Configuration Handbook, Volume 1

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

CF5V1-2.1

I.S. EN ISO 9001

Contents

Chapter Revision Dates

The chapters in this book, *Configuration Handbook, Volume 1*, were revised on the following dates. Where chapters or groups of chapters are available separately, part numbers are listed.

Chapter 1. Configuring Altera FPGAs
Revised: *July 2004*
Part number: *CF51001-2.0*

Chapter 2. Configuring Stratix II Devices
Revised: *January 2005*
Part number: *SII52007-2.1*

Chapter 3. Configuring Stratix & Stratix GX Devices
Revised: *September 2004*
Part number: *S52013-3.1*

Chapter 4. Configuring Cyclone II Devices
Revised: *November 2004*
Part number: *CII51013-1.1*

Chapter 5. Configuring Cyclone FPGAs
Revised: *February 2005*
Part number: *C51013-1.3*

Chapter 6. Configuring APEX II Devices
Revised: *July 2004*
Part number: *CF51004-2.0*

Chapter 7. Configuring APEX 20KE & APEX 20KC Devices
Revised: *February 2005*
Part number: *CF51005-2.1*

Chapter 8. Configuring Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K Devices
Revised: *July 2004*
Part number: *CF51006-2.0*

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	lliterature@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, 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. Altera FPGAs

This section provides information on how to configure Altera® FPGAs. The following chapters contain descriptions of the supported configuration schemes and configuration pins.

This section includes the following chapters:

Revision History

The table below shows the revision history for Chapters 1 through 8.

Chapter	Date/Version	Changes Made
1	July 2004, v2.0	● Added Stratix II and Cyclone II Device information throughout chapter. ● Updated Figure 1–2.
	September 2003, v1.0	Initial Release.

Chapter	Date/Version	Changes Made
2	January, 2005, v2.1	Minor edits.
	December, v1.2	Corrected data rate for FPP configuration using a MAX II device without Stratix II decompression nor the design security feature.
	July 2004, v1.1	● Removed reference to PLMSEPC-8 adapter from "Programming Serial Configuration Devices" section. ● Updated Tables 2–7 and 2–15. ● Updated Figure 2–2. ● Updated "VCCSEL Pin", "Configuration Devices", and "Design Security Using Configuration Bitstream Encryption" sections. ● Added timing specifications for `CONF_DONE` high to user mode with `CLKUSR` option on.
	February 2004, v1.0	Shared Stratix II Device Handbook, Volume 2, Chapter 7.
3	September 2004, v3.1	● Corrected spelling error.
	April 2004, v3.0	● In the "PORSEL Pins" section and the "nIO_PULLUP Pins" section, several pull-down resistors were changed to pull-up resistors. ● Updated notes in Figure 3–3. ● Two vertical V_{CC} lines removed in Figures 3–12 to 3–14. ● Three paragraphs added regarding the `CONF_DONE` and `INIT_DONE` pins in the "PS Configuration with a Microprocessor" section. ● Value in Note 1 changed in Tables 3–8 and 3–9. ● Deleted reference to AS in Table 3–15 because Stratix does not support AS mode. ● Text added before callout of Figure 3–7.
	July 2003, v2.0	● Shared Stratix Device Handbook, Volume 2, Chapter 13. ● Updated to new template layout.
4	November 2004, v1.1	● Updated "Configuration Stage" section in "Single Device AS Configuration" section. ● Updated "Initialization Stage" section in "Single Device AS Configuration" section. ● Updated Figure 4–8. ● Updated "Initialization Stage" section in "Single Device PS Configuration Using a MAX II Device as an External Host" section. ● Updated Table 4–7. ● Updated "Single Device PS Configuration Using a Configuration Device" section. ● Updated "Initialization Stage" section in "Single Device PS Configuration Using a Configuration Device" section. ● Updated Figure 4–18. ● Updated "Single Device JTAG Configuration" section.
	June 2004, v1.0	Shared Cyclone II Device Handbook, Volume 1, Chapter 13.

Chapter	Date/Version	Changes Made
5	February 2005, v1.3	Updated Figures 5–13, 5–18, and 5–19.
	December 2004, v1.2	Minor edits.
	August 2004, v1.2	● Deleted sections: Programming Configuration Devices, Connecting the JTAG Chain, Passive Serial and JTAG, Device Options, Device Configuration Files, Configuration Reliability, and Board Layout Tips. ● Deleted figures: Embedded System Block Diagram, Combining PS & JTAG Configuration, Configuration Options Dialog Box. ● Deleted table: Cyclone Configuration Option Bits. ● Added: USB Blaster to cable list; new Figure 5–13; text on pages 13-14, 13-29, and 13-30, and information to Table 5–6. ● Changes to Figures 5–14 to 5–16, 5–19, 5–20, 5–25; numbers changed in EP1C4 row of Table 5–3. ● Added extensive descriptions of configuration methods under the “Configuring Multiple Devices with the Same Data” section.
	July 2003, v1.1	Shared Cyclone Device Handbook, Volume 1, Chapter 13.
6	July 2004, v2.0	● Updated .rbf sizes in Table 6–2. ● Updated nCEO description in Table 6–8.
	September 2003, v1.0	Initial Release.
7	February 2005, v2.1	Formatting changes.
	July 2004, v2.0	● Updated resistor values in Figure 7–25. ● Updated nCEO description in Table 7–8.
	September 2003, v1.0	Initial Release.
8	July 2004, v2.0	Updated nCEO description in Table 8–19.
	September 2003, v1.0	Initial Release.

1. Configuring Altera FPGAs

CF51001-2.0

Introduction

Stratix® series, Cyclone™ series, APEX™ II, APEX 20K (including APEX 20KE and APEX 20KC), Mercury™, ACEX® 1K, FLEX® 10K (including FLEX 10KE and FLEX 10KA), and FLEX 6000 devices can be configured using one of seven configuration schemes. Table 1–1 shows which device families support which configuration schemes.

Table 1–1. Configuration Scheme Device Family Support

Configuration Scheme	Device Family									
	Stratix II	Stratix, Stratix GX	Cyclone II	Cyclone	APEX II	APEX 20K, APEX 20KE, APEX 20KC	Mercury	ACEX 1K	FLEX 10K, FLEX 10KE, FLEX 10KA	FLEX 6000
Passive Serial (PS)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Active Serial (AS)	✓		✓	✓						
Fast Passive Parallel (FPP)	✓	✓			✓					
Passive Parallel Synchronous (PPS)						✓	✓	✓	✓	
Passive Parallel Asynchronous (PPA)	✓	✓			✓	✓	✓	✓	✓	
Passive Serial Asynchronous (PSA)										✓
Joint Test Action Group (JTAG)	✓	✓	✓	✓	✓	✓	✓	✓	✓	(1)

Note to Table 1–1:

(1) Although you cannot configure FLEX 6000 devices through the JTAG pins, you can perform JTAG boundary-scan testing.

All configuration schemes use either an intelligent host or a configuration device(s) (see Table 1–2).

Table 1–2. Configuration Schemes

Configuration Scheme	Typical Use
Passive Serial (PS)	Configuration with the enhanced configuration devices (EPC16, EPC8, and EPC4), EPC2, EPC1, EPC1441 configuration devices, serial synchronous microprocessor interface, the USB Blaster USB Port Download Cable, MasterBlaste™r communications cable, ByteBlaster™ II parallel download cable or ByteBlasterMV™ parallel port download cable.
Active Serial (AS)	Configuration with the serial configuration devices (EPCS1 and EPCS4).
Passive Parallel Synchronous (PPS)	Configuration with a parallel synchronous microprocessor interface.
Fast Passive Parallel (FPP)	Configuration with an enhanced configuration device or parallel synchronous microprocessor interface where 8 bits of configuration data are loaded on every clock cycle. Eight times faster than PPS.
Passive Parallel Asynchronous (PPA)	Configuration with a parallel asynchronous microprocessor interface. In this scheme, the microprocessor treats the target device as memory.
Passive Serial Asynchronous (PSA)	Configuration with a serial asynchronous microprocessor interface.
Joint Test Action Group (JTAG)	Configuration through the IEEE Std. 1149.1 (JTAG) pins. (1)

The following chapters discuss how to configure one or more Stratix series, Cyclone series, APEX II, APEX 20K (including APEX 20KE and APEX 20KC), Mercury, ACEX 1K, FLEX 10K (including FLEX 10KE and FLEX 10KA), and FLEX 6000 devices. The following chapters should be used in conjunction with the following documents:

- Stratix II Device Handbook
- Stratix Device Handbook
- Stratix GX FPGA Family Data Sheet
- Cyclone II Device Handbook
- Cyclone Device Handbook
- APEX II Programmable Logic Device Family Data Sheet
- APEX 20K Programmable Logic Device Family Data Sheet
- APEX 20KC Programmable Logic Device Data Sheet
- Mercury Programmable Logic Device Family Data Sheet
- ACEX 1K Programmable Logic Device Family Data Sheet
- FLEX 10K Embedded Programmable Logic Family Data Sheet
- FLEX 10KE Embedded Programmable Logic Family Data Sheet
- FLEX 6000 Programmable Logic Device Family Data Sheet

Volume I will cover how to configure Altera FPGAs, where each chapter covers a different device family. Each subsection describes how to configure the devices with the following configuration schemes:

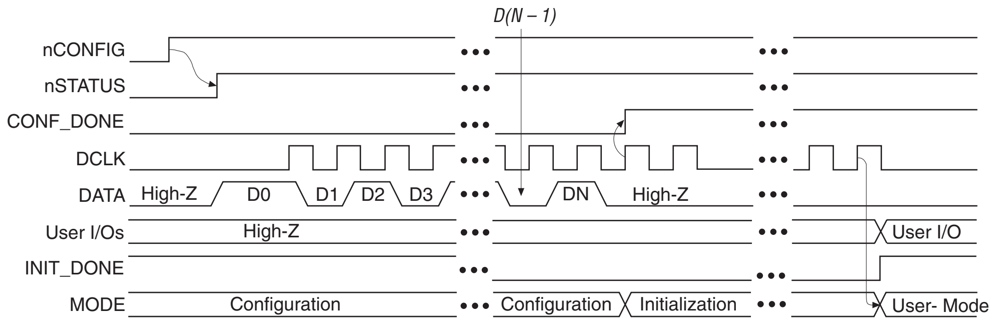
- PS Configuration
  - Using a Configuration Device
  - Using a Microprocessor
  - Using a Download Cable
- AS Configuration (Stratix II FPGAs and the Cyclone Series Only)
- FPP Configuration (Stratix Series and APEX II Devices Only)
  - Using an enhanced Configuration Device
  - Using a Microprocessor
- PPS Configuration (APEX 20K, Mercury, ACEX 1K, and FLEX 10K Devices Only)
- PPA Configuration (Stratix Series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K Devices Only)
- PSA Configuration (FLEX 6000 Devices Only)
- JTAG Programming and Configuration (Stratix Series, Cyclone Series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K Devices Only)

Volume II contains information that is relevant for all Altera FPGAs discussed in this handbook. Information about configuration devices and combining different Altera device families in the same configuration chain can be found in this volume.

Device Configuration Overview for Passive Schemes

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. After the device is configured, its registers and I/O pins must be initialized. After initialization, the device enters user mode for in-system operation. Figure 1–1 shows the waveform of the configuration pins during configuration, initialization, and user-mode.

Figure 1–1. Configuration Cycle Waveform



The low-to-high transition of nCONFIG on the FPGA begins the configuration cycle. The configuration cycle consists of 3 stages: reset, configuration, and initialization. While nCONFIG is low, the device is in reset. When the device comes out of reset, nCONFIG must be at a logic high level in order for the device to release the open-drain nSTATUS pin. Once nSTATUS is released, it is pulled high by a pull-up resistor and the FPGA is ready to receive configuration data. Before and during configuration all user I/O pins are tri-stated. Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the I/O pins which are on before and during configuration.

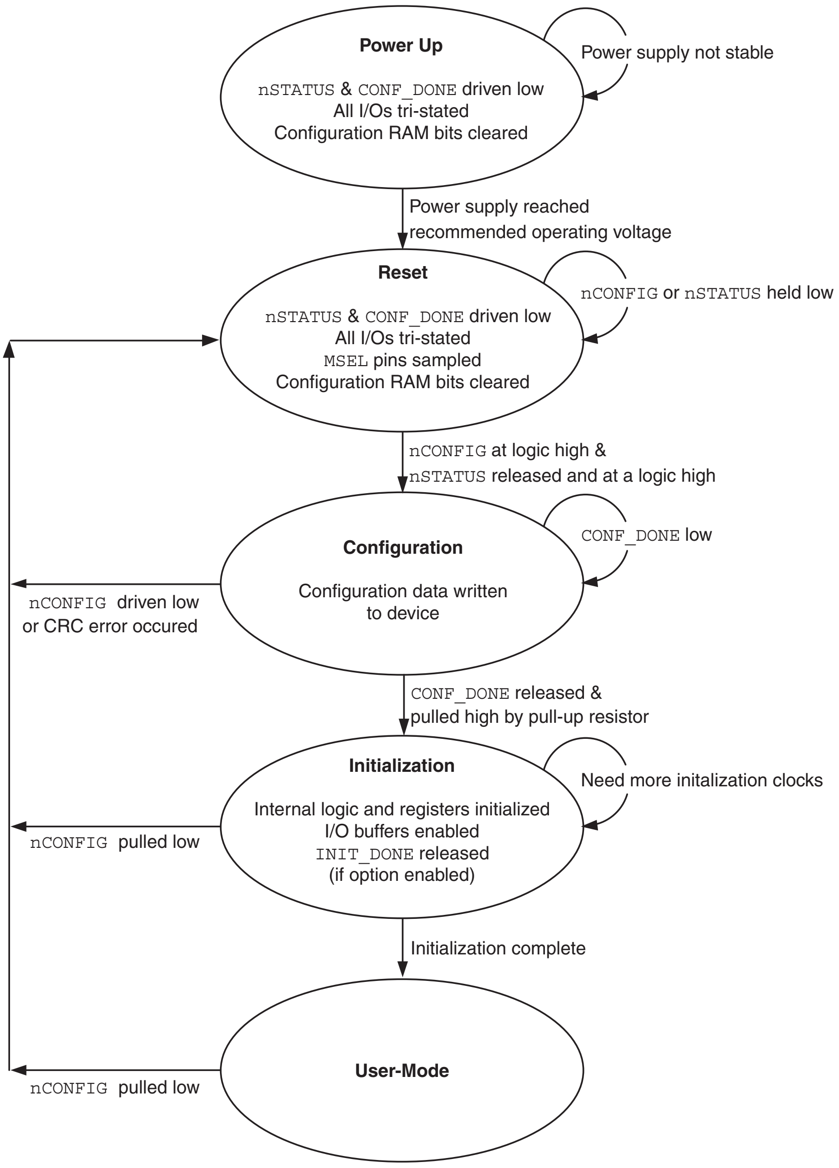
nCONFIG and nSTATUS must be at a logic high level in order for the configuration stage to begin. The device receives configuration data on its DATA pin(s) and (for synchronous configuration schemes) the clock source on the DCLK pin. Configuration data is latched into the FPGA on the rising edge of DCLK. After the FPGA has received all configuration data successfully it releases the CONF_DONE pin, which is pulled high by a pull-up resistor. A low to high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode. During initialization, internal logic, internal and I/O registers are initialized and I/O buffers are enabled. When initialization is finished, the INIT_DONE pin is released and pulled high by an external pull-up resistor. Once in user-mode, the user I/O pins will no longer have a weak pull-up and will function as assigned in your design. The DCLK, DATA (FLEX 6000), and DATA0 (Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10KE) pins should not be left floating after configuration; they should be driven high or low, whichever is convenient, on your board.

A reconfiguration is initiated by toggling the nCONFIG pin from high to low and then back to high. When nCONFIG is pulled low, nSTATUS and CONF_DONE are also pulled low and all I/O pins are tri-stated. Once nCONFIG and nSTATUS return to a logic high level, configuration begins.

Figure 1–2 shows a simple state diagram of the configuration process.

Figure 1–2. Configuration Cycle State Machine


Power Up
nSTATUS & CONF_DONE driven low
All I/Os tri-stated
Configuration RAM bits cleared
Power supply not stable
Power supply reached recommended operating voltage
Reset
nSTATUS & CONF_DONE driven low
All I/Os tri-stated
MSEL pins sampled
Configuration RAM bits cleared
nCONFIG or nSTATUS held low
nCONFIG at logic high & nSTATUS released and at a logic high
Configuration
Configuration data written to device
CONF_DONE low
nCONFIG driven low or CRC error occured
CONF_DONE released & pulled high by pull-up resistor
Initialization
Internal logic and registers initialized
I/O buffers enabled
INIT_DONE released
(if option enabled)
Need more initalization clocks
nCONFIG pulled low
Initialization complete
User-Mode
nCONFIG pulled low

Selecting a Configuration Scheme

The configuration data for Altera devices can be loaded using an active, passive or JTAG configuration scheme. When using an active configuration scheme with a serial configuration device, the target FPGA generates the control and synchronization signals. When both devices are ready to begin configuration, the serial configuration device sends data to the FPGA.

When using any passive configuration scheme, the Altera device is incorporated into a system with an Altera configuration device or an intelligent host, such as a microprocessor, that controls the configuration process. The configuration device or host supplies configuration data from a storage device (a configuration device(s), a hard disk, RAM, or other system memory). When using passive configuration, you can change the target device's functionality while the system is in operation by reconfiguring it.

Altera devices support a number of configuration schemes. Not all device families support all configuration schemes. Table 1–1 and the individual device family sections should be referenced to determine if your target device family supports your intended configuration scheme. Once you have decided on the appropriate configuration scheme for your system, you will need to drive the dedicated mode select control pins, `MSEL`, of the FPGA to set the configuration mode.

For further details on how to set the `MSEL` pins for your target device, refer to the appropriate device family chapters.

Below is a brief description of each configuration scheme. For detailed information, consult the appropriate sections.

Passive Serial Configuration

The PS configuration scheme is supported in the Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, FLEX 10K, and FLEX 6000 device families. PS configuration can be performed by using an Altera download cable, an Altera enhanced configuration device or configuration device, or an intelligent host, such as a microprocessor. During PS configuration, configuration data is transferred from a storage device, such as a configuration device or flash memory, to the FPGA on the `DATA` (FLEX 6000) or `DATA0` (Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K) pin. This configuration data is latched into the FPGA on the rising edge of `DCLK`. Configuration data is transferred one bit per clock cycle.

Active Serial Configuration

The AS configuration scheme is supported in the Stratix II and Cyclone series device families. AS configuration can be performed by using an Altera Serial Configuration device. During AS configuration, the Stratix II or Cyclone series device is the master and the configuration device is the slave. Configuration data is transferred to the FPGA on the `DATA0` pin. This configuration data is synchronized to the `DCLK` input. Configuration data is transferred one bit per clock cycle.

Passive Parallel Synchronous Configuration

The PPS configuration scheme is supported in the APEX 20K, Mercury, ACEX 1K and FLEX 10K device families. PPS configuration can be performed by using an intelligent host, such as a microprocessor. During PPS configuration, configuration data is transferred from a storage device, such as flash memory, to the FPGA on the `DATA[7..0]` pins. This configuration data is synchronized to the `DCLK` input. On the first rising edge of `DCLK`, a byte of configuration data is latched into the FPGA. The next 8 falling edges of `DCLK` are needed to internally serialize the data in the FPGA.

Fast Passive Parallel Configuration

The FPP configuration scheme is supported in the Stratix series and APEX II device families. FPP configuration can be performed by using an Altera enhanced configuration device, or an intelligent host, such as a microprocessor. During FPP configuration, configuration data is transferred from a storage device, such as an enhanced configuration device or flash memory, to the FPGA on the `DATA[7..0]` pins. This configuration data is latched into the FPGA on the rising edge of `DCLK`. Configuration data is transferred one byte per clock cycle.

Passive Parallel Asynchronous Configuration

The PPA configuration scheme is supported in the Stratix series, APEX II, APEX 20K, Mercury, ACEX 1K and FLEX 10K device families. PPA configuration can be performed by using an intelligent host, such as a microprocessor. During PPA configuration, configuration data is transferred from a storage device, such as a configuration device or flash memory, to the FPGA on the `DATA[7..0]` pins. Since this configuration scheme is asynchronous, control signals are used to regulate the configuration cycle.

Passive Serial Asynchronous Configuration

The PSA configuration scheme is supported in the FLEX 6000 device family. PSA configuration can be performed by using an intelligent host, such as a microprocessor. During PSA configuration, configuration data is transferred from a storage device, such as a configuration device or flash memory, to the FPGA on the DATA pin. Since this configuration scheme is asynchronous, control signals are used to regulate the configuration cycle.

JTAG Configuration

The JTAG configuration scheme is supported in the Stratix series, Cyclone series, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K device families. JTAG configuration uses the IEEE Std 1149.1 JTAG interface pins and supports the JAM STAPL standard. JTAG configuration can be performed by using an Altera download cable or an intelligent host, such as a microprocessor.

2. Configuring Stratix II Devices

SII52007-2.1

Introduction

Stratix® II devices use SRAM cells to store configuration data. Since SRAM memory is volatile, configuration data must be downloaded to Stratix II devices each time the device powers up. Stratix II devices can be configured using one of five configuration schemes: the fast passive parallel (FPP), active serial (AS), passive serial (PS), passive parallel asynchronous (PPA), and Joint Test Action Group (JTAG) configuration schemes. All configuration schemes use either an external controller (for example, a MAX® II device or microprocessor) or a configuration device.

Configuration Devices

The Altera enhanced configuration devices (EPC16, EPC8, and EPC4) support a single-device configuration solution for high-density devices and can be used in the FPP and PS configuration schemes. They are ISP-capable through its JTAG interface. The enhanced configuration devices are divided into two major blocks, the controller and the flash memory.

For information on enhanced configuration devices, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* chapter and the *Using Altera Enhanced Configuration Devices* chapter in the *Configuration Handbook*.

The Altera serial configuration devices (EPCS4 and EPCS1) support a single-device configuration solution for Stratix II devices and are used in the AS configuration scheme. Serial configuration devices offer a low cost, low pin count configuration solution.

For information on serial configuration devices, see the *Serial Configuration Devices (EPCS1 & EPCS4) Data Sheet* chapter.

The EPC2 and EPC1 configuration devices provide configuration support for the PS configuration scheme. The EPC2 device is ISP-capable through its JTAG interface. The EPC2 and EPC1 can be cascaded to hold large configuration files.

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

The configuration scheme is selected by driving the Stratix II device `MSEL` pins either high or low as shown in Table 2–1. The `MSEL` pins are powered by the V_{CCPD} power supply of the bank they reside in. The `MSEL[3..0]` pins have 5-kΩ internal pull-down resistors that are always active. During POR and during reconfiguration, the `MSEL` pins have to be at LVTTL V_{IL} and V_{IH} levels to be considered a logic low and logic high.

☞ To avoid any problems with detecting an incorrect configuration scheme, hard-wire the `MSEL[]` pins to V_{CCPD} and GND, without any pull-up or pull-down resistors. Do not drive the `MSEL[]` pins by a microprocessor or another device.

Table 2–1. Stratix II Configuration Schemes

Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
Fast passive parallel (FPP)	0	0	0	0
Passive parallel asynchronous (PPA)	0	0	0	1
Passive serial (PS)	0	0	1	0
Remote system upgrade FPP *(1)*	0	1	0	0
Remote system upgrade PPA *(1)*	0	1	0	1
Remote system upgrade PS *(1)*	0	1	1	0
Fast AS (40 MHz) *(2)*	1	0	0	0
Remote system upgrade fast AS (40 MHz) *(2)*	1	0	0	1
FPP with decompression and/or design security feature enabled *(3)*	1	0	1	1
Remote system upgrade FPP with decompression and/or design security feature enabled *(1)*, *(3)*	1	1	0	0
AS (20 MHz) *(2)*	1	1	0	1
Remote system upgrade AS (20 MHz) *(2)*	1	1	1	0
JTAG-based configuration *(5)*	*(4)*	*(4)*	*(4)*	*(4)*

Notes to Table 2–1:

(1) These schemes require that you drive the `RUnLU` pin to specify either remote update or local update. For more information about remote system upgrades in Stratix II devices, see *Chapter 8, Remote System Upgrades with Stratix II Devices* in Volume 2 of the *Stratix II Device Handbook*.

(2) Only the EPCS16 and EPCS64 devices support up to a 40 MHz `DCLK`. Other EPCS devices support up to a 20 MHz `DCLK`. See the *Serial Configuration Devices Data Sheet* for more information.

(3) These modes are only supported when using a MAX II device or a microprocessor with flash memory for configuration. In these modes, the host system must output a `DCLK` that is 4× the data rate.

(4) Do not leave the `MSEL` pins floating. Connect them to V_{CCPD} or ground. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used, you should connect the `MSEL` pins to ground.

(5) JTAG-based configuration takes precedence over other configuration schemes, which means `MSEL` pin settings are ignored.

Stratix II devices offer the design security, decompression, and remote system upgrade features. Design security using configuration bitstream encryption is available in Stratix II devices, which protects your designs. Stratix II devices can receive a compressed configuration bit stream and decompress this data in real-time, reducing storage requirements and configuration time. You can make real-time system upgrades from remote locations of your Stratix II designs by using the remote system upgrade feature.

Table 2–2 shows the approximate uncompressed configuration file sizes for Stratix II devices.

Table 2–2. Stratix II .rbf Sizes *Notes (1), (2)*

Device	Data Size (MBits)	Data Size (MBytes)
EP2S15	5.0	0.625
EP2S30	10.1	1.2625
EP2S60	17.1	2.1375
EP2S90	27.5	3.4375
EP2S130	39.6	4.95
EP2S180	52.4	6.55

Notes to Table 2–2:
(1) These values are preliminary.
(2) **.rbf**: Raw Binary File.

Use the data in Table 2–2 to estimate the file size before design compilation. Different configuration file formats, such as a Hexidecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II software, any design targeted for the same device will have the same uncompressed configuration file size. If you are using compression, the file size can vary after each compilation since the compression ratio is dependent on the design.

This chapter explains the Stratix II device configuration features and describes how to configure Stratix II devices using the supported configuration schemes. This chapter configuration pin descriptions and the Stratix II device configuration file format. In this chapter, the generic term device(s) includes all Stratix II devices.

For more information on setting device configuration options or creating configuration files, see *Software Settings* in Volume 2 of the *Configuration Handbook*.

Configuration Features

Stratix II devices offer configuration data decompression to reduce configuration file storage, design security using data encryption to protect your designs, and remote system upgrades to allow for remotely updating your Stratix II designs. Table 2–3 summarizes which configuration features can be used in each configuration scheme.

Table 2–3. Stratix II Configuration Features

Configuration Scheme	Configuration Method	Design Security	Decompression	Remote System Upgrade
FPP	MAX II device or a Microprocessor with flash memory	✓ (1)	✓ (1)	✓
	Enhanced Configuration Device		✓ (2)	✓
AS	Serial Configuration Device	✓	✓	✓ (3)
PS	MAX II device or a Microprocessor with flash memory	✓	✓	✓
	Enhanced Configuration Device	✓	✓	✓
	Download cable	✓	✓	
PPA	MAX II device or a Microprocessor with flash memory			✓
JTAG	MAX II device or a Microprocessor with flash memory			

Notes to Table 2–3:
(1) In these modes, the host system must send a DCLK that is 4× the data rate.
(2) The enhanced configuration device decompression feature is available, while the Stratix II decompression feature is not available.
(3) Only remote update mode is supported when using the AS configuration scheme. Local update mode is not supported.

Configuration Data Decompression

Stratix II devices support configuration data decompression, which saves configuration memory space and time. This feature allows you to store compressed configuration data in configuration devices or other memory and transmit this compressed bit stream to Stratix II devices. During configuration, the Stratix II device decompresses the bit stream in real time and programs its SRAM cells.

Preliminary data indicates that compression typically reduces configuration bit stream size by 35 to 55%.

Stratix II devices support decompression in the FPP (when using a MAX II device/microprocessor + flash), AS, and PS configuration schemes. Decompression is not supported in the PPA configuration scheme nor in JTAG-based configuration.

☞ When using FPP mode, the intelligent host must provide a `DCLK` that is 4× the data rate. Therefore, the configuration data must be valid for four `DCLK` cycles.

The decompression feature supported by Stratix II devices is different from the decompression feature in enhanced configuration devices (EPC16, EPC8, and EPC4 devices), although they both use the same compression algorithm. The data decompression feature in the enhanced configuration devices allows them to store compressed data and decompress the bitstream before transmitting it to the target devices. When using Stratix II devices in FPP mode with enhanced configuration devices, the Stratix II decompression feature is not available, but the enhanced configuration device decompression feature is.

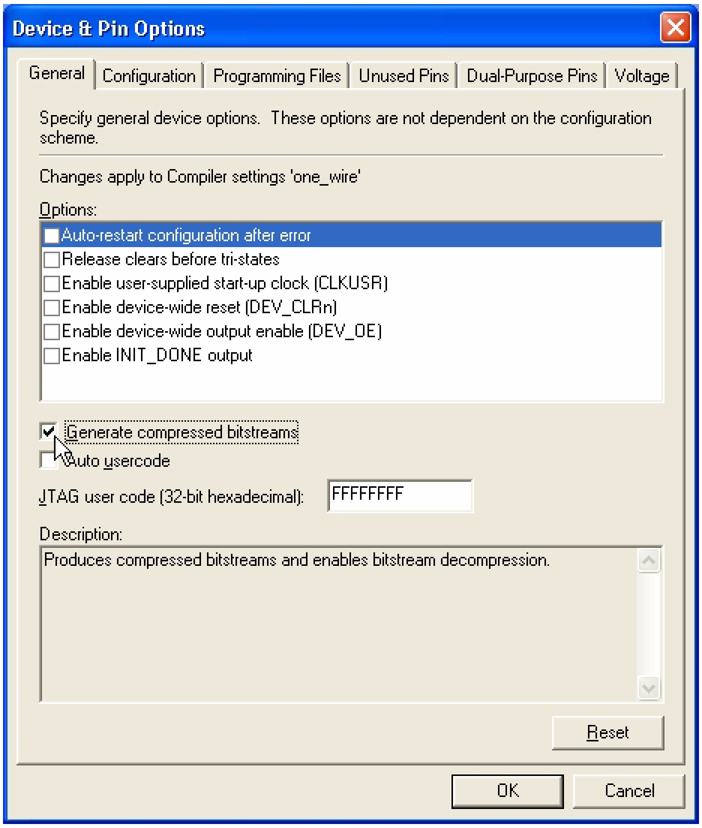
In PS mode, you should use the Stratix II decompression feature since sending compressed configuration data reduces configuration time. You should not use both the Stratix II device and the enhanced configuration device decompression features simultaneously. The compression algorithm is not intended to be recursive and could expand the configuration file instead of compressing it further.

When you enable compression, the Quartus II software generates configuration files with compressed configuration data. This compressed file reduces the storage requirements in the configuration device or flash memory, and decreases the time needed to transmit the bitstream to the Stratix II device. The time required by a Stratix II device to decompress a configuration file is less than the time needed to transmit the configuration data to the device.

There are two methods to enable compression for Stratix II bitstreams: before design compilation (in the **Compiler Settings** menu) and after design compilation (in the **Convert Programming Files** window).

To enable compression in the project's compiler settings, select **Device** under the **Assignments** menu to bring up the **Settings** window. After selecting your Stratix II device, open the **Device & Pin Options** window, and in the **General** settings tab enable the check box for **Generate compressed bitstreams** (as shown in Figure 2–1).

Figure 2–1. Enabling Compression for Stratix II Bitstreams in Compiler Settings



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

Changes apply to Compiler settings 'one_wire'

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:

Produces compressed bitstreams and enables bitstream decompression.

Compression can also be enabled when creating programming files from the **Convert Programming Files** window.

1. Click **Convert Programming Files** (File menu).
2. Select the programming file type (POF, SRAM HEXOUT, RBF, or TTF).
3. For POF output files, select a configuration device.
4. In the **Input files to convert** box, select **SOF Data**.
5. Select **Add File** and add a Stratix II device SOF(s).

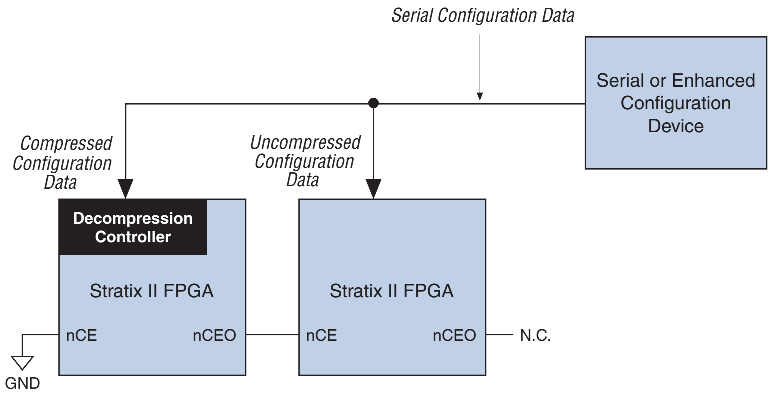
6. Select the name of the file you added to the **SOF Data** area and click **Properties**.

7. Check the **Compression** check box.

When multiple Stratix II devices are cascaded, the compression feature can be selectively enabled for each device in the chain if you are using a serial configuration scheme. Figure 2–2 depicts a chain of two Stratix II devices. The first Stratix II device has compression enabled and therefore receives a compressed bit stream from the configuration device. The second Stratix II device has the compression feature disabled and receives uncompressed data.

In a multi-device FPP configuration chain all Stratix II devices in the chain must either enable of disable the decompression feature. You can not selectively enable the compression feature for each device in the chain because of the `DATA` and `DCLK` relationship.

Figure 2–2. Compressed and Uncompressed Configuration Data in the Same Configuration File


Serial Configuration Data
Serial or Enhanced Configuration Device
Compressed Configuration Data
Uncompressed Configuration Data
Decompression Controller
Stratix II FPGA
nCE
nCEO
Stratix II FPGA
nCE
nCEO
N.C.
GND

You can generate programming files for this setup from the **Convert Programming Files** window (File menu) in the Quartus II software.

Design Security Using Configuration Bitstream Encryption

Stratix II devices are the industry's first devices with the ability to decrypt a configuration bitstream using the Advanced Encryption Standard (AES) algorithm—the most advanced encryption algorithm available today. When using the design security feature, a 128-bit security key is stored in the Stratix II device. In order to successfully configure a Stratix II device which has the design security feature enabled, it must be configured with a configuration file that was encrypted using the same 128-bit security key. The security key can be stored in non-volatile memory inside the Stratix II device. This non-volatile memory does not require any external devices, such as a battery back-up, for storage.

☞ An encryption configuration file is the same size as a non-encryption configuration file. When using a serial configuration scheme such as passive serial (PS) or active serial (AS), configuration time is the same whether or not the design security feature is enabled. If the fast passive parallel (FPP) scheme us used with the design security or decompression feature, a 4× `DCLK` is required. This results in a slower configuration time when compared to the configuration time of an FPGA that has neither the design security, nor decompression feature enabled. For more information about this feature, contact Altera applications.

Remote System Upgrade

Stratix II devices feature remote and local update. For more information about this feature, see Chapter 8, Remote System Upgrades with Stratix II Devices in Volume 2 of the *Stratix II Device Handbook.*

V_{CCPD} Pins

Stratix II devices also offer a new power supply, V_{CCPD}, which must be connected to 3.3-V in order to power the 3.3-V/2.5-V buffer available on the configuration input pins and JTAG pins. V_{CCPD} applies to all the JTAG input pins (`TCK`, `TMS`, `TDI`, and `TRST`) and the configuration pins: `nCONFIG`, `DCLK` (when used as an input), `nIO_Pullup`, `DATA[7..0]`, `RUnLU`, `nCE`, `nWS`, `nRS`, `CS`, `nCS` and `CLKUSR`.

☞ V_{CCPD} must ramp-up from 0-V to 3.3-V within 100 ms. If V_{CCPD} is not ramped up within this specified time, your Stratix II device will not configure successfully. If your system does not allow for a V_{CCPD} ramp-up time of 100 ms or less, you must hold `nCONFIG` low until all power supplies are stable.

VCCSEL Pin

The `VCCSEL` pin allows the V_{CCIO} setting (of the banks where the configuration inputs reside) to be independent of the voltage required by the configuration inputs. Therefore, when selecting V_{CCIO}, the V_{IL} and V_{IH} levels driven to the configuration inputs are not a factor.

The configuration input pins (nCONFIG, DCLK (when used as an input), nIO_Pullup, RUnLU, nCE, nWS, nRS, CS, nCS, and CLKUSR) have a dual buffer design. These pins have a 3.3-V/2.5-V input buffer and a 1.8-V/1.5-V input buffer. The VCCSEL input pin selects which input buffer is used during configuration. After configuration, the dual-purpose configuration pins are powered by the V_{CCIO} pins. The 3.3-V/2.5-V input buffer is powered by V_{CCPD}, while the 1.8-V/1.5-V input buffer is powered by V_{CCIO}.

VCCSEL is sampled during power-up. Therefore, the VCCSEL setting cannot change on the fly or during a reconfiguration. The VCCSEL input buffer is powered by V_{CCINT} and has an internal 5-kΩ pull-down resistor that is always active.

VCCSEL must be hardwired to V_{CCPD} or GND.

A logic high selects the 1.8-V/1.5-V input buffer, and a logic low selects the 3.3-V/2.5-V input buffer. VCCSEL should be set to comply with the logic levels driven out of the configuration device or MAX II device or a microprocessor with flash memory.

If you need to support 3.3-V or 2.5-V configuration input voltages, set VCCSEL low. You can set the bank V_{CCIO} that contains the configuration inputs to any supported voltage. If you need to support 1.8-V or 1.5-V configuration input voltages, set VCCSEL to a logic high and the V_{CCIO} of the bank that contains the configuration inputs to 1.8 or 1.5-V.

VCCSEL also sets the POR trip point for I/O bank 8 to ensure that this I/O bank has powered up to the appropriate voltage levels before configuration begins. Upon power-up, the device will not release nSTATUS until V_{CCINT} and V_{CCIO} of bank 8 is above its POR trip point. If you set VCCSEL to ground (logic low), this sets the POR trip point for bank 8 to a voltage consistent with 3.3-V/2.5-V signaling, which means the POR trip point for these I/O banks may be as high as 1.8V. If V_{CCIO} of any of the configuration banks is set to 1.8-V or 1.5-V, the voltage supplied to this I/O bank(s) may never reach the POR trip point, which will cause the device to never begin configuration.

If the V_{CCIO} of I/O bank 8 is set to 1.5-V or 1.8-V and the configuration signals used require 3.3-V or 2.5-V signaling, you should set VCCSEL to V_{CCPD} (logic high) in order to lower the POR trip point to enable successful configuration.

Table 2–4 shows how you should set the VCCSEL depending on the V_{CCIO} setting of bank 8 and your configuration input signaling voltages.

Table 2–4. VCCSEL Setting

VCCIO (bank 8)	Configuration Input Signaling Voltage	VCCSEL
3.3-V/2.5-V	3.3-V/2.5-V	GND
1.8-V/1.5-V	3.3-V/2.5-V/1.8-V/1.5-V	VCCPD
3.3-V/2.5-V	1.8-V/1.5-V	Not Supported

The VCCSEL signal does not control TDO, nCEO, or any of the dual-purpose pins, including the dual-purpose configuration pins, such as the DATA[7..0] and PPA pins (nWS, nRS, CS, nCS, and RDYnBSY). During configuration, these pins will drive out voltage levels corresponding to the V_{CCIO} supply voltage that powers the I/O bank containing the pin. After configuration, the dual-purpose pins inherit the I/O standards specified in the design.

For more information on multi-volt support, including information on using TDO and nCEO in multi-volt systems, refer to the "MultiVolt I/O Interface" section of the *Stratix II Architecture* chapter in Volume 1 of the *Stratix II Handbook*.

Fast Passive Parallel Configuration

Fast passive parallel (FPP) configuration in Stratix II devices is designed to meet the continuously increasing demand for faster configuration times. Stratix II devices are designed with the capability of receiving byte-wide configuration data per clock cycle. Table 2–5 shows the MSEL pin settings when using the FFP configuration scheme.

Table 2–5. Stratix II MSEL Pin Settings for FPP Configuration Schemes (Part 1 of 2)

Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
FPP when not using remote system upgrade or decompression and/or design security feature	0	0	0	0
FPP when using remote system upgrade *(1)*	0	1	0	0
FPP with decompression and/or design security feature enabled *(2)*	1	0	1	1

Table 2–5. Stratix II MSEL Pin Settings for FPP Configuration Schemes (Part 2 of 2)

Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
FPP when using remote system upgrade and decompression and/or design security feature *(1)*, *(2)*	1	1	0	0

Notes to Table 2–5:

(1) These schemes require that you drive the RUnLU pin to specify either remote update or local update. For more information about remote system upgrade in Stratix II devices, see the *Chapter 8, Remote System Upgrades with Stratix II Devices* in Volume 2 of the *Stratix II Device Handbook*.

(2) These modes are only supported when using a MAX II device or a microprocessor with flash memory for configuration. In these modes, the host system must output a DCLK that is 4× the data rate.

FPP configuration of Stratix II devices can be performed using an intelligent host, such as a MAX II device, or microprocessor, or an Altera enhanced configuration device.

FPP Configuration Using a MAX II Device as an External Host

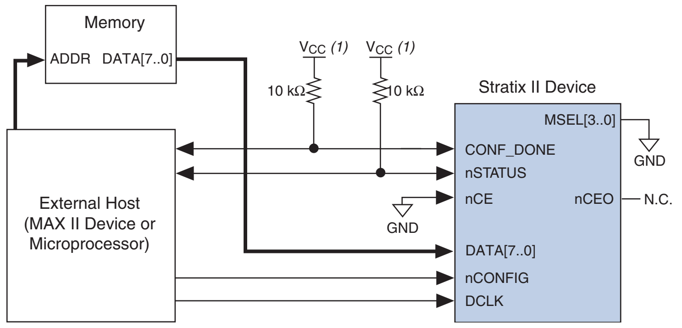
FPP configuration using compression and an external host provides the fastest method to configure Stratix II devices. In the FPP configuration scheme, a MAX II device can be used as an intelligent host that controls the transfer of configuration data from a storage device, such as flash memory, to the target Stratix II device. Configuration data can be stored in RBF, HEX, or TTF format. When using the MAX II devices as an intelligent host, a design that controls the configuration process, such as fetching the data from flash memory and sending it to the device, must be stored in the MAX II device.

☞ If you are using the Stratix II decompression and/or design security feature, the external host must be able to send a DCLK frequency that is 4× the data rate.

The 4× DCLK signal does not require an additional pin and is sent on the DCLK pin. The maximum DCLK frequency is 100 MHz, which results in a maximum data rate of 200 Mbps. If you are not using the Stratix II decompression nor are using the design security feature, the data rate is 8× the DCLK frequency.

Figure 2–3 shows the configuration interface connections between the Stratix II device and a MAX II device for single device configuration.

Figure 2–3. Single Device FPP Configuration Using an External Host



Note to Figure 2–3:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

Upon power-up, the Stratix II device goes through a Power-On Reset (POR). The POR delay is dependent on the PORSEL pin setting; when PORSEL is driven low, the POR time is approximately 100 ms, if PORSEL is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold nSTATUS low, and tri-state all user I/O pins. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If nIO_pullup is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If nIO_pullup is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *DC & Switching Characteristic* chapter in the *Stratix II Device Handbook*.

The configuration cycle consists of three stages: reset, configuration and initialization. While nCONFIG or nSTATUS are low, the device is in the reset stage. To initiate configuration, the MAX II device must drive the nCONFIG pin from low-to-high.

V_{CCINT}, V_{CCIO}, and V_{CCPD} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once nSTATUS is released, the device is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the MAX II device places the configuration data one byte at a time on the DATA[7..0] pins.

☞ The Stratix II device receives configuration data on its DATA[7..0] pins and the clock is received on the DCLK pin. Data is latched into the device on the rising edge of DCLK. If you are using the Stratix II decompression and/or design security feature, configuration data is latched on the rising edge of every fourth DCLK cycle. After the configuration data is latched in, it is processed during the following three DCLK cycles.

Data is continuously clocked into the target device until CONF_DONE goes high. The CONF_DONE pin will go high one byte early in parallel configuration (FPP and PPA) modes. The last byte is required for serial configuration (AS and PS) modes. After the device has received the next to last byte of the configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Stratix II devices, the initialization clock source is either the Stratix II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Stratix II device will provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You can also synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. The **Enable user-supplied start-up clock** (**CLKUSR**) option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. The CONF_DONE pin will go high one byte early in parallel configuration (FPP and PPA) modes. The last byte is required for serial configuration (AS and PS) modes. After the CONF_DONE pin transitions high, CLKUSR will be enabled after the time specified as t_{CD2CU}. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. The MAX II device must be able to detect this low-to-high transition which signals the device has entered user mode. When initialization is complete, the device enters user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

To ensure DCLK and DATA[7..0] are not left floating at the end of configuration, the MAX II device must drive them either high or low, whichever is convenient on your board. The DATA[7..0] pins are available as user I/O pins after configuration. When you select the FPP scheme in the Quartus II software, as a default, these I/O pins are tri-stated in user mode and should be driven by the MAX II device. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

The configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

☞ If you are using the Stratix II decompression and/or design security feature and need to stop DCLK, it can only be stopped three clock cycles after the last data byte was latched into the Stratix II device.

By stopping DCLK, the configuration circuit allows enough clock cycles to process the last byte of latched configuration data. When the clock restarts, the MAX II device must provide data on the DATA[7..0] pins prior to sending the first DCLK rising edge.

If an error occurs during configuration, the device drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the MAX II device that there is an error. If the **Auto-restart configuration after error** option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** (dialog box) is turned on, the device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the MAX II device can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

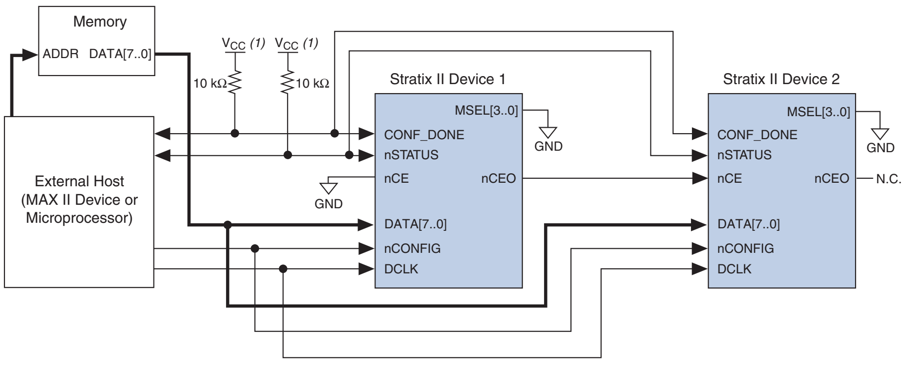
The MAX II device can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the MAX II device to detect errors and determine when programming completes. If all configuration data is sent, but the CONF_DONE or INIT_DONE signals have not gone high, the MAX II device will reconfigure the target device.

☞ If the optional CLKUSR pin is used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the device is in user-mode, initiating a reconfiguration is done by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 40 µs. When nCONFIG is pulled low, the device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the device, reconfiguration begins.

Figure 2–4 shows how to configure multiple devices using a MAX II device. This circuit is similar to the FPP configuration circuit for a single device, except the Stratix II devices are cascaded for multi-device configuration.

Figure 2–4. Multi-Device FPP Configuration Using an External Host



Note to Figure 2–4:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O standard on the device and the external host.

In multi-device FPP configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the MAX II device. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. The configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

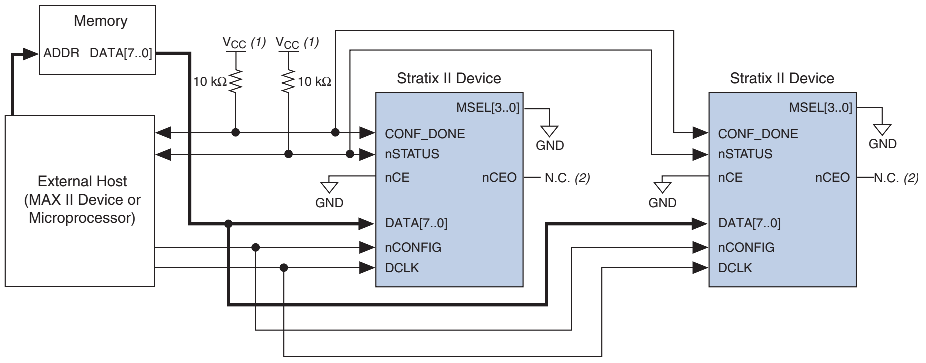
All nSTATUS and CONF_DONE pins are tied together and if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first device flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single device detecting an error.

If the **Auto-restart configuration after error** option is turned on, the devices release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the MAX II device can try to reconfigure the chain without pulsing nCONFIG low. If this option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

In a multi-device FPP configuration chain, all Stratix II devices in the chain must either enable or disable the decompression and/or design security feature. You can not selectively enable the decompression and/or design security feature for each device in the chain because of the DATA and DCLK relationship. If the chain contains devices that do not support design security, you should use a serial configuration scheme.

If a system has multiple devices that contain the same configuration data, tie all device nCE inputs to GND, and leave nCEO pins floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. Configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 2–5 shows multi-device FPP configuration when both Stratix II devices are receiving the same configuration data.

Figure 2–5. Multiple-Device FPP Configuration Using an External Host When Both Devices Receive the Same Data



Notes to Figure 2–5:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

(2) The `nCEO` pins of both Stratix II devices are left unconnected when configuring the same configuration data into multiple devices.

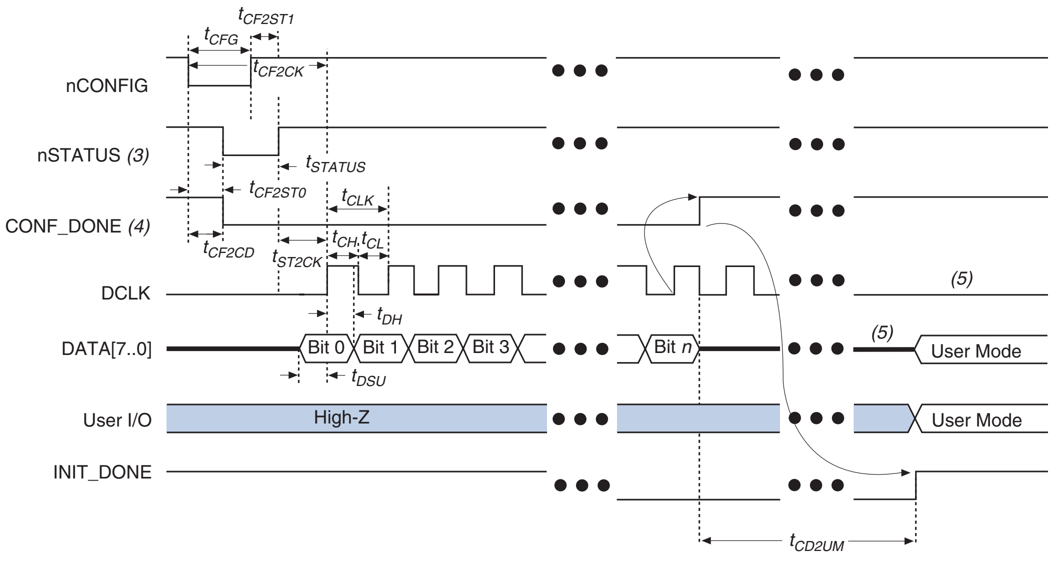
You can use a single configuration chain to configure Stratix II devices with other Altera devices that support FPP configuration, such as Stratix devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, tie all of the device `CONF_DONE` and `nSTATUS` pins together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera device Chains* in the *Configuration Handbook*.

FPP Configuration Timing

Figure 2–6 shows the timing waveform for FPP configuration when using a MAX II device as an external host. This waveform shows the timing when the decompression and the design security feature are not enabled.

Figure 2–6. FPP Configuration Timing Waveform Notes (1), (2)



Notes to Figure 2–6:

(1) This timing waveform should be used when the decompression and design security feature are not used.
(2) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS`, and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.
(3) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.
(4) Upon power-up, before and during configuration, `CONF_DONE` is low.
(5) `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..0]` are available as user I/O pins after configuration and the state of these pins depends on the dual-purpose pin settings.

Table 2–6 defines the timing parameters for Stratix II devices for FPP configuration when the decompression and the design security feature are not enabled.

Table 2–6. FPP Timing Parameters for Stratix II Devices (Part 1 of 2) Notes (1), (2)

Symbol	Parameter	Min	Max	Units
t_{POR}	POR delay	12	100	ms
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		800	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		800	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	10	100 (3)	µs

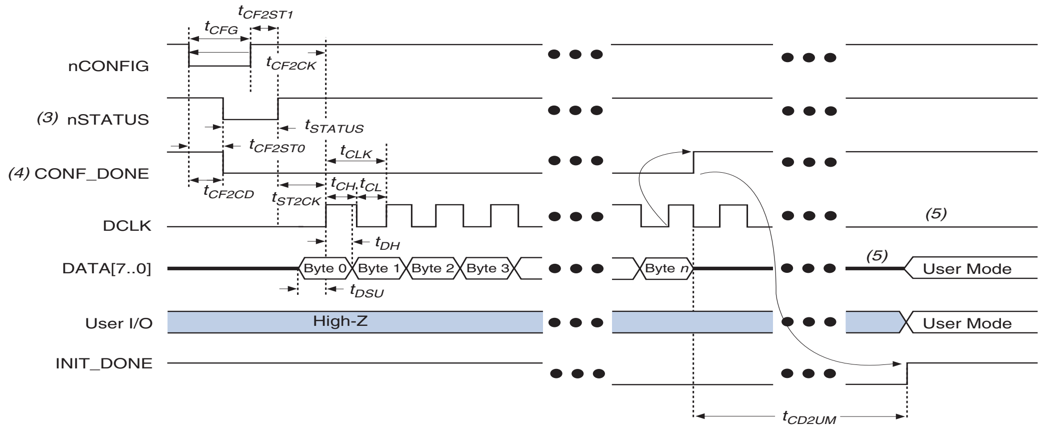
Table 2–6. FPP Timing Parameters for Stratix II Devices (Part 2 of 2) *Notes (1), (2)*

Symbol	Parameter	Min	Max	Units
t_{CF2ST1}	nCONFIG high to nSTATUS high		100 *(3)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	100		µs
t_{ST2CK}	nSTATUS high to first rising edge of DCLK	2		µs
t_{DSU}	Data setup time before rising edge on DCLK	7		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK frequency		100	MHz
t_R	Input rise time		40	ns
t_F	Input fall time		40	ns
t_{CD2UM}	CONF_DONE high to user mode *(4)*	20	40	µs
t_{CD2CU}	CONF_DONE high to CLKUSR enabled	4 × maximum DCLK period		
t_{CD2UMC}	CONF_DONE high to user mode with CLKUSR option on	t_{CD2CU} + (299 × CLKUSR period)		

Notes to Table 2–6:
(1) This information is preliminary.
(2) These timing parameters should be used when the decompression and design security feature are not used.
(3) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.
(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device.

Figure 2–7 shows the timing waveform for FPP configuration when using a MAX II device as an external host. This waveform shows the timing when the decompression and/or the design security feature are enabled.

Figure 2–7. FPP Configuration Timing Waveform With Decompression or Design Security Feature Enabled Notes (1), (2)



Notes to Figure 2–7:

(1) This timing waveform should be used when the decompression and/or design security feature are used.

(2) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS` and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.

(3) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.

(4) Upon power-up, before and during configuration, `CONF_DONE` is low.

(5) `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..0]` are available as user I/O pins after configuration and the state of these pins depends on the dual-purpose pin settings.

(6) If needed, `DCLK` can be paused by holding it low. When `DCLK` restarts, the external host must provide data on the `DATA[7..0]` pins prior to sending the first `DCLK` rising edge.

Table 2–7 defines the timing parameters for Stratix II devices for FPP configuration when the decompression and/or the design security feature are enabled.

Table 2–7. FPP Timing Parameters for Stratix II Devices With Decompression or Design Security Feature Enabled *Notes (1), (2)*

Symbol	Parameter	Min	Max	Units
t_{POR}	POR delay	12	100	ms
t_{CF2CD}	nCONFIG low to CONF_DONE low		800	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		800	ns
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	10	100 *(3)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		100 *(3)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	100		µs
t_{ST2CK}	nSTATUS high to first rising edge of DCLK	2		µs
t_{DSU}	Data setup time before rising edge on DCLK	7		ns
t_{DH}	Data hold time after rising edge on DCLK	30		ns
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK frequency		100	MHz
t_{DATA}	Data rate		200	Mbps
t_R	Input rise time		40	ns
t_F	Input fall time		40	ns
t_{CD2UM}	CONF_DONE high to user mode *(4)*	20	40	µs
t_{CD2CU}	CONF_DONE high to CLKUSR enabled	4 × maximum DCLK period		
t_{CD2UMC}	CONF_DONE high to user mode with CLKUSR option on	t_{CD2CU} + (299 × CLKUSR period)		

Notes to Table 2–7:
(1) This information is preliminary.
(2) These timing parameters should be used when the decompression and design security feature are used.
(3) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.
(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* chapter in the *Configuration Handbook.*

FPP Configuration Using a Microprocessor

In the FPP configuration scheme, a microprocessor can control the transfer of configuration data from a storage device, such as flash memory, to the target Stratix II device.

All information in "FPP Configuration Using a MAX II Device as an External Host" on page 2–11 is also applicable when using a microprocessor as an external host. Refer to that section for all configuration and timing information.

FPP Configuration Using an Enhanced Configuration Device

In the FPP configuration scheme, an enhanced configuration device sends a byte of configuration data every `DCLK` cycle to the Stratix II device. Configuration data is stored in the configuration device.

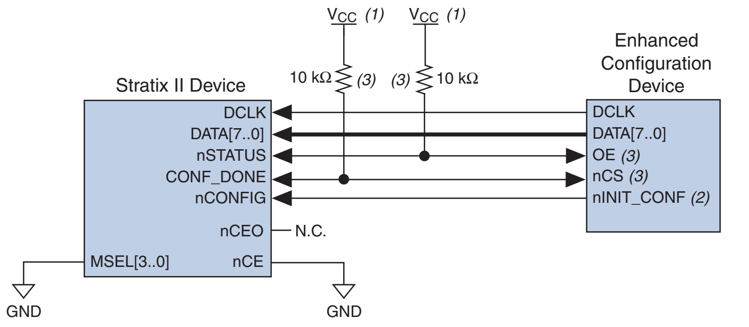
☞ When configuring your Stratix II device using FPP mode and an enhanced configuration device, the enhanced configuration device decompression feature is available while the Stratix II decompression feature and design security feature are not.

Figure 2–8 shows the configuration interface connections between the Stratix II device and the enhanced configuration device for single device configuration.

☞ The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the device.

For more information on the enhanced configuration device and flash interface pins, such as `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`, refer to the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet.*

Figure 2–8. Single Device FPP Configuration Using an Enhanced Configuration Device



Notes to Figure 2–8:

(1) The pull-up resistor 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 should not be used 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.

The value of the internal pull-up resistors on the enhanced configuration devices can be found in the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet.*

When using enhanced configuration devices, you can connect the device's `nCONFIG` pin to `nINIT_CONF` pin of the enhanced configuration device, which allows the `INIT_CONF` JTAG instruction to initiate device configuration. 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. An internal pull-up resistor on the `nINIT_CONF` pin is always active in the enhanced configuration devices, which means an external pull-up resistor should not be used if `nCONFIG` is tied to `nINIT_CONF`.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the `PORSEL` pin setting; when `PORSEL` is driven low, the POR time is approximately 100 ms, if `PORSEL` is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold `nSTATUS` low, and tri-state all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for enhanced configuration devices can be set to either 100 ms or 2 ms, depending on its `PORSEL` pin setting. If the `PORSEL` pin is connected to GND, the POR delay is 100 ms. If the `PORSEL` pin is connected to V_{CC}, the POR delay is 2 ms. During this time, the configuration device drives its `OE` pin low. This low signal delays configuration because the `OE` pin is connected to the target device's `nSTATUS` pin.

When selecting a POR time, you need to ensure that the device completes power-up before the enhanced configuration device exits POR. Altera recommends that you use a 12-ms POR time for the Stratix II device, and use a 100-ms POR time for the enhanced configuration device.

When both devices complete POR, they release their open-drain `OE` or `nSTATUS` pin, which is then pulled high by a pull-up resistor. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If `nIO_pullup` is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If `nIO_pullup` is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *Stratix II Device Handbook***.**

When the power supplies have reached the appropriate operating voltages, the target device senses the low-to-high transition on `nCONFIG` and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. The beginning of configuration can be delayed by holding the `nCONFIG` or `nSTATUS` pin low.

V_{CCINT}, V_{CCIO} and V_{CCPD} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the `nSTATUS` pin, which is pulled high by a pull-up resistor. Enhanced configuration devices have an optional internal pull-up resistor on the `OE` pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 10-kΩ pull-up resistor on the `OE-nSTATUS` line is required. Once `nSTATUS` is released, the device is ready to receive configuration data and the configuration stage begins.

When nSTATUS is pulled high, the configuration device's OE pin also goes high and the configuration device clocks data out to the device using its internal oscillator. The Stratix II device receives configuration data on its DATA[7..0] pins and the clock is received on the DCLK pin. A byte of data is latched into the device on each rising edge of DCLK.

After the device has received all configuration data successfully, it releases the open-drain CONF_DONE pin which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration devices have an optional internal pull-up resistor on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 10-kΩ pull-up resistor on the nCS-CONF_DONE line is required. A low to high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Stratix II devices, the initialization clock source is either the Stratix II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Stratix II device will provide itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. The **Enable user-supplied start-up clock** (**CLKUSR**) option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, CLKUSR will be enabled after the time specified as t_{CD2CU}. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. The enhanced configuration device will drive DCLK low and DATA[7..0] high at the end of configuration.

If an error occurs during configuration, the device drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the **Auto-restart configuration after error** option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the device will automatically initiate reconfiguration if an error occurs. The Stratix II device will release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the device has not configured successfully. Enhanced configuration devices wait for 64 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. If the Auto-restart configuration after error option is set in the software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns to a logic high level, the configuration device will try to reconfigure the device.

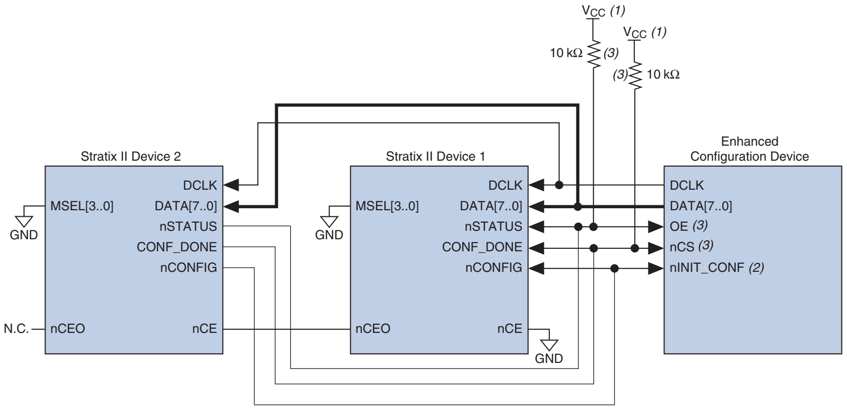
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully. Therefore, your system should not pull CONF_DONE low to delay initialization. Instead, you should use the CLKUSR option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

If the optional CLKUSR pin is used and nCONFIG is pulled low to restart configuration during device initialization, ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the device is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 40 µs. When nCONFIG is pulled low, the device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this will activate the configuration device as it will see its nCS pin drive low. Once nCONFIG returns to a logic high level and nSTATUS is released by the device, reconfiguration begins.

Figure 2–9 shows how to configure multiple Stratix II devices with an enhanced configuration device. This circuit is similar to the configuration device circuit for a single device, except the Stratix II devices are cascaded for multi-device configuration.

Figure 2–9. Multi-Device FPP Configuration Using an Enhanced Configuration Device



Notes to Figure 2–9:

(1) The pull-up resistor 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 should not be used 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-up resistors on configuration device** option when generating programming files.

Enhanced configuration devices cannot be cascaded.

When performing multi-device configuration, you must generate the configuration device's POF from each project's SOF. You can combine multiple SOFs using the **Convert Programming Files** window in the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, see *Software Settings* in Volume 2 of the *Configuration Handbook*.

In multi-device FPP configuration, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. Pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device.

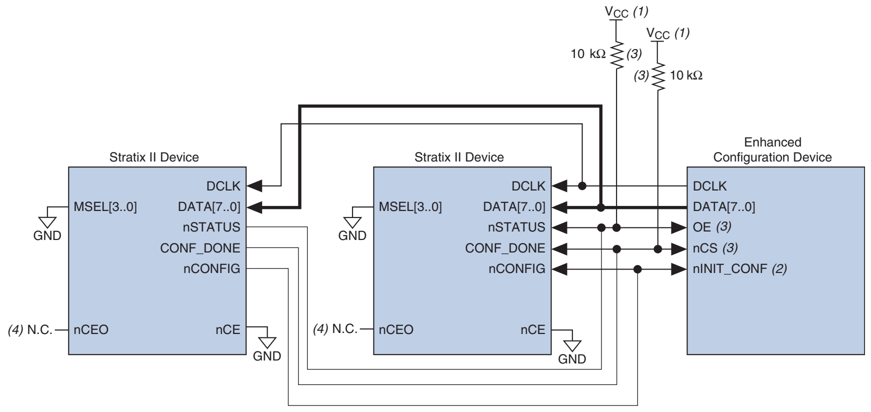
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first device flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the enhanced configuration device and drives nSTATUS low on all devices, which causes them to enter a reset state. This behavior is similar to a single device detecting an error.

If the **Auto-restart configuration after error** option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The devices will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the **Auto-restart configuration after error** option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

Your system may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. Configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 2–10 shows multi-device FPP configuration when both Stratix II devices are receiving the same configuration data.

Figure 2–10. Multiple-Device FPP Configuration Using an Enhanced Configuration Device When Both devices Receive the Same Data



Notes to Figure 2–10:

(1) The pull-up resistor 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 should not be used 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.

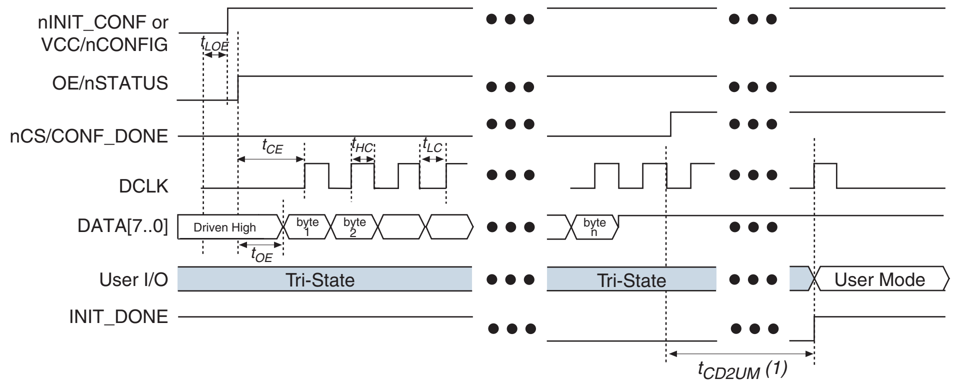
(4) The `nCEO` pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single enhanced configuration chain to configure multiple Stratix II devices with other Altera devices that support FPP configuration, such as Stratix and Stratix GX devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera device Chains* in the *Configuration Handbook*.

Figure 2–11 shows the timing waveform for the FPP configuration scheme using an enhanced configuration device.

Figure 2–11. Stratix II FPP Configuration Using an Enhanced Configuration Device Timing Waveform



Note to Figure 2–11:

(1) The initialization clock can come from the Stratix II internal oscillator or the `CLKUSR` pin.

For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* in the *Configuration Handbook*.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* section in of the *Configuration Handbook*.

Active Serial Configuration (Serial Configuration Devices)

In the AS configuration scheme, Stratix II devices are configured using a serial configuration device. These configuration devices are low cost devices with non-volatile memory that feature a simple four-pin interface and a small form factor. These features make serial configuration devices an ideal low-cost configuration solution.

For more information on serial configuration devices, see the *Serial Configuration Devices Data Sheet* in the *Configuration Handbook*.

Serial configuration devices provide a serial interface to access configuration data. During device configuration, Stratix II devices read configuration data via the serial interface, decompress data if necessary, and configure their SRAM cells. This scheme is referred to as the AS configuration scheme because the device controls the configuration interface. This scheme contrast the PS configuration scheme where the configuration device controls the interface.

☞ The Stratix II decompression and design security feature are fully available when configuring your Stratix II device using AS mode.

Table 2–8 shows the MSEL pin settings when using the AS configuration scheme.

Table 2–8. Stratix II MSEL Pin Settings for AS Configuration Schemes

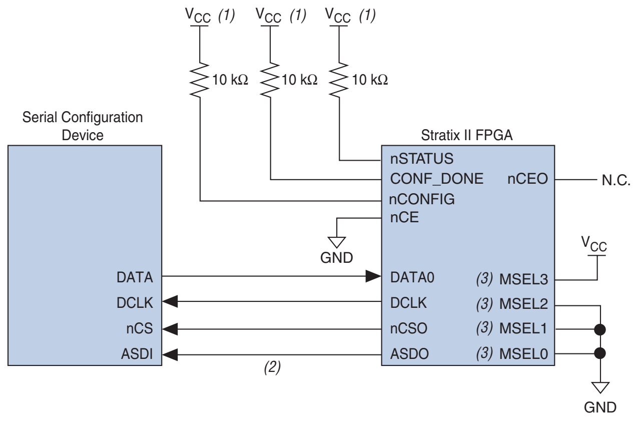
Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
Fast AS (40 MHz) *(1)*	1	0	0	0
Remote system upgrade fast AS (40 MHz) *(1)*	1	0	0	1
AS (20 MHz) *(1)*	1	1	0	1
Remote system upgrade AS (20 MHz) *(1)*	1	1	1	0

Note to Table 2–8:

(1) Only the EPCS16 and EPCS64 devices support a DCLK up to 40 MHz clock; other EPCS devices support a DCLK up to 20 MHz. See the *Serial Configuration Devices Data Sheet* for more information.

Serial configuration devices have a four-pin interface: serial clock input (DCLK), serial data output (DATA), AS data input (ASDI), and an active-low chip select (nCS). This four-pin interface connects to Stratix II device pins, as shown in Figure 2–12.

Figure 2–12. Single Device AS Configuration



Notes to Figure 2–12:

(1) Connect the pull-up resistors to a 3.3-V supply.

(2) Stratix II devices use the `ASDO` to `ASDI` path to control the configuration device.

(3) If using an EPCS4 device, `MSEL[3..0]` should be set to 1101. See Table 2–8 for more details.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the `PORSEL` pin setting. When `PORSEL` is driven low, the POR time is approximately 100 ms. If `PORSEL` is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold `nSTATUS` and `CONF_DONE` low, and tri-state all user I/O pins. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If `nIO_pullup` is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If `nIO_pullup` is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *Operating Conditions table* in the *Stratix II Device Handbook*.

The configuration cycle consists of three stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. After POR, the Stratix II device releases `nSTATUS`, which is pulled high by an external 10-kΩ pull-up resistor, and enters configuration mode.

To begin configuration, power the V_{CCINT}, V_{CCIO}, and V_{CCPD} voltages (for the banks where the configuration and JTAG pins reside) to the appropriate voltage levels.

The serial clock (DCLK) generated by the Stratix II device controls the entire configuration cycle and provides the timing for the serial interface. Stratix II devices use an internal oscillator to generate DCLK. Using the MSEL[] pins, you can select to use either a 40- or 20-MHz oscillator.

Only the EPCS16 and EPCS64 devices support a DCLK up to 40-MHz clock; other EPCS devices support a DCLK up to 20-MHz. See the *Serial Configuration Devices Data Sheet* for more information.

Table 2–9 shows the active serial DCLK output frequencies.

Table 2–9. Active Serial DCLK Output Frequency *Note (1)*

Oscillator	Minimum	Typical	Maximum	Units
40 MHz *(2)*	20	26	40	MHz
20 MHz	10	13	20	MHz

Notes to Table 2–9:

(1) These values are preliminary.

(2) Only the EPCS16 and EPCS64 devices support a DCLK up to 40-MHz clock; other EPCS devices support a DCLK up to 20-MHz. See the *Serial Configuration Devices Data Sheet* for more information.

The serial configuration device latches input/control signals on the rising edge of DCLK and drives out configuration data on the falling edge. Stratix II devices drive out control signals on the falling edge of DCLK and latch configuration data on the rising edge of DCLK.

In configuration mode, the Stratix II device enables the serial configuration device by driving its nCSO output pin low, which connects to the chip select (nCS) pin of the configuration device. The Stratix II device uses the serial clock (DCLK) and serial data output (ASDO) pins to send operation commands and/or read address signals to the serial configuration device. The configuration device provides data on its serial data output (DATA) pin, which connects to the DATA0 input of the Stratix II device.

After all configuration bits are received by the Stratix II device, it releases the open-drain CONF_DONE pin, which is pulled high by an external 10-kΩ resistor. Initialization begins only after the CONF_DONE signal reaches a logic high level. All AS configuration pins, DATA0, DCLK, nCSO, and ASDO, have weak internal pull-up resistors, which are always active. Therefore, after configuration these pins will be driven high.

In Stratix II devices, the initialization clock source is either the Stratix II 10-MHz (typical) internal oscillator (separate from the active serial internal oscillator) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Stratix II device will provide itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. The **Enable user-supplied start-up clock** (**CLKUSR**) option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. When you **Enable** the **user supplied start-up clock** option, the CLKUSR pin is the initialization clock source. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, CLKUSR will be enabled after 600 ns. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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 that the device has entered user mode. When initialization is complete, the device enters user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

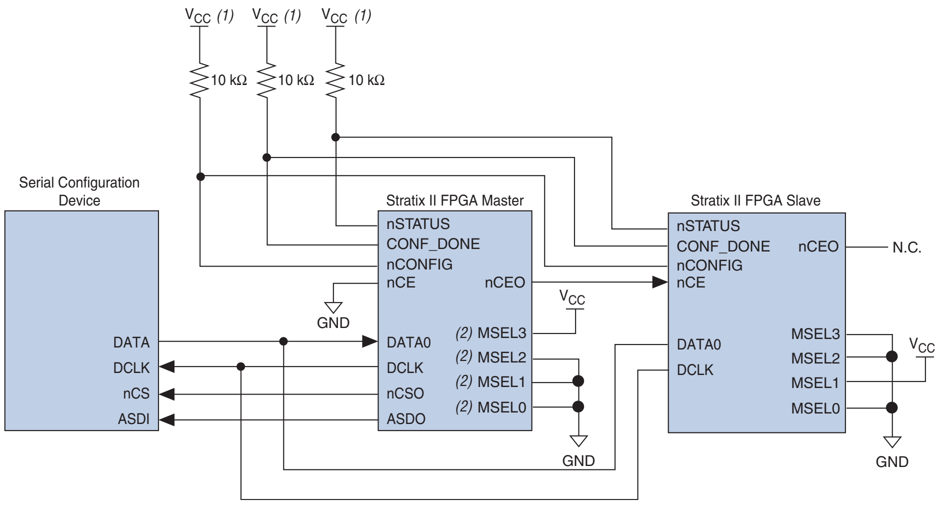
If an error occurs during configuration, the Stratix II device asserts the nSTATUS signal low indicating a data frame error, and the CONF_DONE signal will stay low. If the **Auto-restart configuration after error** option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the Stratix II device resets the configuration device by pulsing nCSO, releases nSTATUS after a reset time-out period (about 40 µs), and retries configuration. If this option is turned off, the system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 µs to restart configuration.

When the Stratix II device is in user mode, you can initiate reconfiguration by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 40 µs. When nCONFIG is pulled low, the device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the Stratix II device, reconfiguration begins.

You can configure multiple Stratix II devices using a single serial configuration device. You can cascade multiple Stratix II devices using the chip-enable (nCE) and chip-enable-out (nCEO) pins. The first device in the chain must have its nCE pin connected to ground. You must connect its nCEO pin to the nCE pin of the next device in the chain. When the first device captures all of its configuration data from the bit stream, it drives the nCEO pin low, enabling the next device in the chain. You must leave the nCEO pin of the last device unconnected. The nCONFIG, nSTATUS, CONF_DONE, DCLK, and DATA0 pins of each device in the chain are connected (see Figure 2–13).

This first Stratix II device in the chain is the configuration master and controls configuration of the entire chain. You must connect its MSEL pins to select the AS configuration scheme. The remaining Stratix II devices are configuration slaves and you must connect their MSEL pins to select the PS configuration scheme. Any other Altera device that supports PS configuration can also be part of the chain as a configuration slave. Figure 2–13 shows the pin connections for this setup.

Figure 2–13. Multi-Device AS Configuration



Notes to Figure 2–13:
(1) Connect the pull-up resistors to a 3.3-V supply.
(2) If using an EPCS4 device, MSEL[3..0] should be set to 1101. See Table 2–8 for more details.

As shown in Figure 2–13, the nSTATUS and CONF_DONE pins on all target devices are connected together with external pull-up resistors. These pins are open-drain bidirectional pins on the devices. When the first device asserts nCEO (after receiving all of its configuration data), it releases its CONF_DONE pin. But the subsequent devices in the chain keep this shared CONF_DONE line low until they have received their configuration data. When all target devices in the chain have received their configuration data and have released CONF_DONE, the pull-up resistor drives a high level on this line and all devices simultaneously enter initialization mode.

If an error occurs at any point during configuration, the nSTATUS line is driven low by the failing device. If you enable the Auto-restart configuration after error option, reconfiguration of the entire chain begins after a reset time-out period (a maximum of 40 µs). If the **Auto-restart configuration after error** option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}.

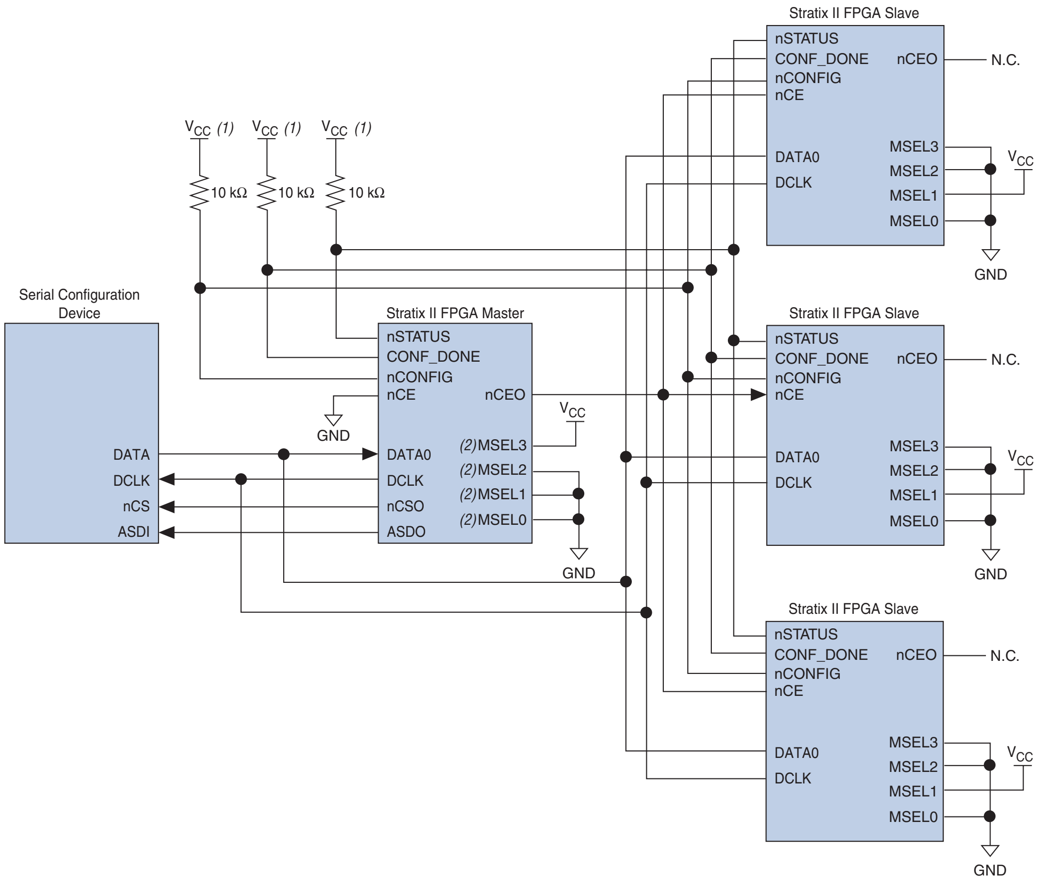
☞ While you can cascade Stratix II devices, serial configuration devices cannot be cascaded or chained together.

If the configuration bit stream size exceeds the capacity of a serial configuration device, you must select a larger configuration device and/or enable the compression feature. When configuring multiple devices, the size of the bitstream is the sum of the individual devices' configuration bitstreams.

A system may have multiple devices that contain the same configuration data. In active serial chains, this can be implemented by storing two copies of the SOF in the serial configuration device. The first copy would configure the master Stratix II device, and the second copy would configure all remaining slave devices concurrently. All slave devices must be the same density and package. The setup is similar to Figure 2–13, where the master is setup in active serial mode and the slave devices are setup in passive serial mode.

To configure four identical Stratix II devices with the same SOF, you could setup the chain similar to the example shown in Figure 2–14. The first device is the master device and its MSEL pins should be set to select AS configuration. The other three slave devices are set up for concurrent configuration and its MSEL pins should be set to select PS configuration. The nCEO pin from the master device drives the nCE input pins on all three slave devices, and the DATA and DCLK pins connect in parallel to all four devices. During the first configuration cycle, the master device reads its configuration data from the serial configuration device while holding nCEO high. After completing its configuration cycle, the master drives nCE low and transmits the second copy of the configuration data to all three slave devices, configuring them simultaneously.

Figure 2–14. Multi-Device AS Configuration When devices Receive the Same Data



Notes to Figure 2–14:

(1) Connect the pull-up resistors to a 3.3-V supply.

(2) If using an EPCS4 device, MSEL[3..0] should be set to 1101. See Table 2–8 for more details.

Estimating Active Serial Configuration Time

Active serial configuration time is dominated by the time it takes to transfer data from the serial configuration device to the Stratix II device. This serial interface is clocked by the Stratix II `DCLK` output (generated from an internal oscillator). As listed in Table 2–9, the `DCLK` minimum frequency when choosing to use the 40-MHz oscillator is 20 MHz (50 ns). Therefore, the maximum configuration time estimate for an EP2S15 device (5 MBits of uncompressed data) is:

RBF Size (minimum `DCLK` period / 1 bit per `DCLK` cycle) = estimated maximum configuration time

5 Mbits × (50 ns / 1 bit) = 250 ms

To estimate the typical configuration time, use the typical `DCLK` period as listed in Table 2–9. With a typical `DCLK` period of 38.46 ns, the typical configuration time is 192 ms. Enabling compression reduces the amount of configuration data that is transmitted to the Stratix II device, which also reduces configuration time. On average, compression reduces configuration time by 50%.

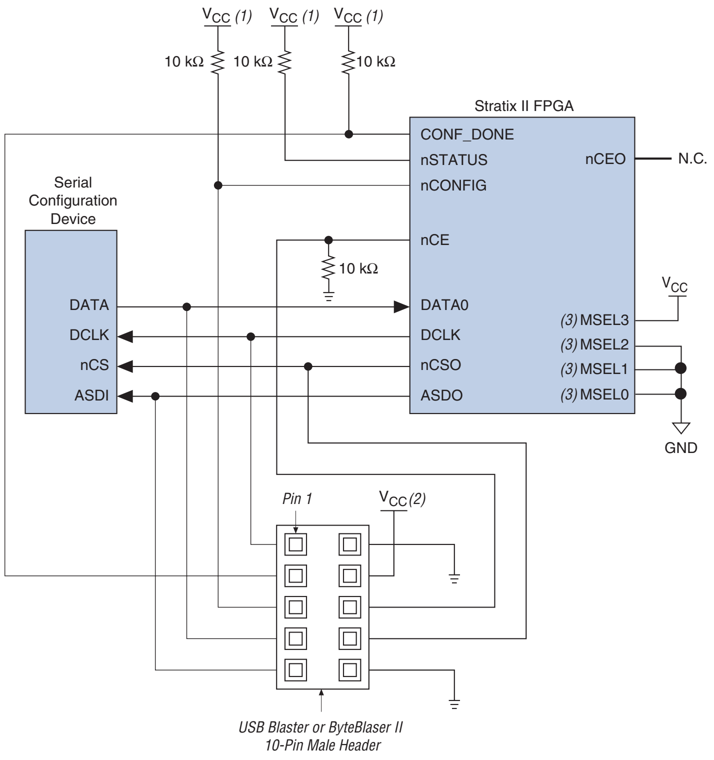
Programming Serial Configuration Devices

Serial configuration devices are non-volatile, flash-memory-based devices. You can program these devices in-system using the USB-Blaster™ or ByteBlaster™ II download cable. Alternatively, you can program them using the Altera Programming Unit (APU), supported third-party programmers, or a microprocessor with the SRunner software driver.

You can perform in-system programming of serial configuration devices via the AS programming interface. During in-system programming, the download cable disables device access to the AS interface by driving the `nCE` pin high. Stratix II devices are also held in reset by a low level on `nCONFIG`. After programming is complete, the download cable releases `nCE` and `nCONFIG`, allowing the pull-down and pull-up resistors to drive GND and V_{CC}, respectively. Figure 2–15 shows the download cable connections to the serial configuration device.

For more information on the USB Blaster download cable, see the *USB-Blaster USB Port Download Cable Data Sheet*. For more information on the ByteBlaster II cable, see the *ByteBlaster II Download Cable Data Sheet*.

Figure 2–15. In-System Programming of Serial Configuration Devices



Notes to Figure 2–15:

(1) Connect these pull-up resistors to 3.3-V supply.

(2) Power up the ByteBlaster II cable's V_{CC} with a 3.3-V supply.

(3) If using an EPCS4 device, `MSEL[3..0]` should be set to `1101`. See Table 2–8 for more details.

You can program serial configuration devices by using the Quartus II software with the Altera programming hardware (APU) and the appropriate configuration device programming adapter. The EPCS1 and EPCS4 devices are offered in an eight-pin small outline integrated circuit (SOIC) package.

In production environments, serial configuration devices can be programmed using multiple methods. Altera programming hardware or other third-party programming hardware can be used to program blank serial configuration devices before they are mounted onto printed circuit boards (PCBs). Alternatively, you can use an on-board microprocessor to program the serial configuration device in-system using C-based software drivers provided by Altera.

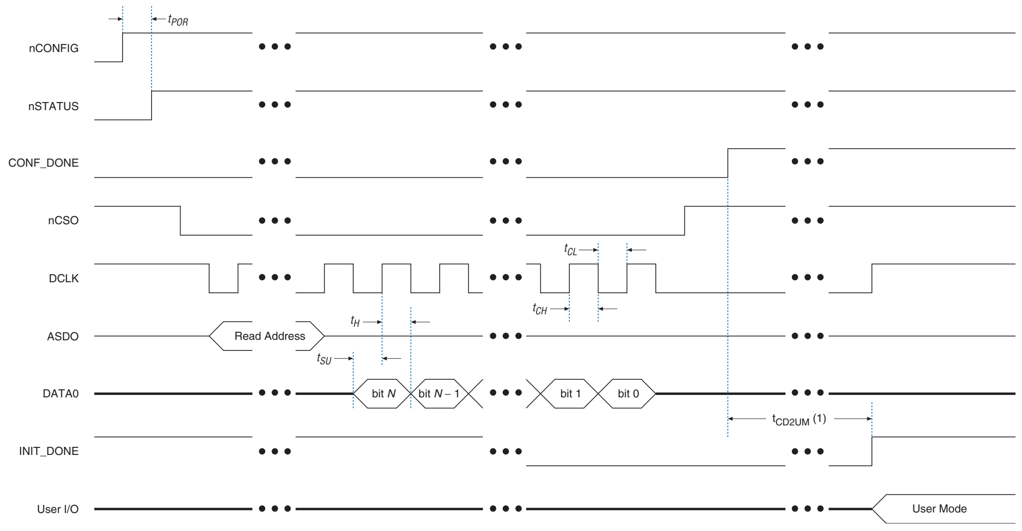
A serial configuration device can be programmed in-system by an external microprocessor using SRunner. SRunner is a software driver developed for embedded serial configuration device programming, which can be easily customized to fit in different embedded systems. SRunner is able to read a raw programming data (**.rpd**) file and write to the serial configuration devices. The serial configuration device programming time using SRunner is comparable to the programming time with the Quartus II software.

For more information about SRunner, see the *SRunner: An Embedded Solution for EPCS Programming* White Paper and the source code on the Altera web site at **www.altera.com**.

For more information on programming serial configuration devices, see the *Serial Configuration Devices (EPCS1 & EPCS4) Data Sheet* in the *Configuration Handbook*.

Figure 2–16 shows the timing waveform for the AS configuration scheme using a serial configuration device.

Figure 2–16. AS Configuration Timing



Note to Figure 2–16:

(1) The initialization clock can come from the Stratix II internal oscillator or the `CLKUSR` pin.

Passive Serial Configuration

PS configuration of Stratix II devices can be performed using an intelligent host, such as a MAX II device or microprocessor with flash memory, an Altera configuration device, or a download cable. In the PS scheme, an external host (MAX II device, embedded processor, configuration device, or host PC) controls configuration. Configuration data is clocked into the target Stratix II devices via the `DATA0` pin at each rising edge of `DCLK`.

☞ The Stratix II decompression and design security feature are fully available when configuring your Stratix II device using PS mode.

Table 2–10 shows the `MSEL` pin settings when using the PS configuration scheme.

Table 2–10. Stratix II MSEL Pin Settings for PS Configuration Schemes

Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
PS	0	0	1	0
PS when using Remote System Upgrade *(1)*	0	1	1	0

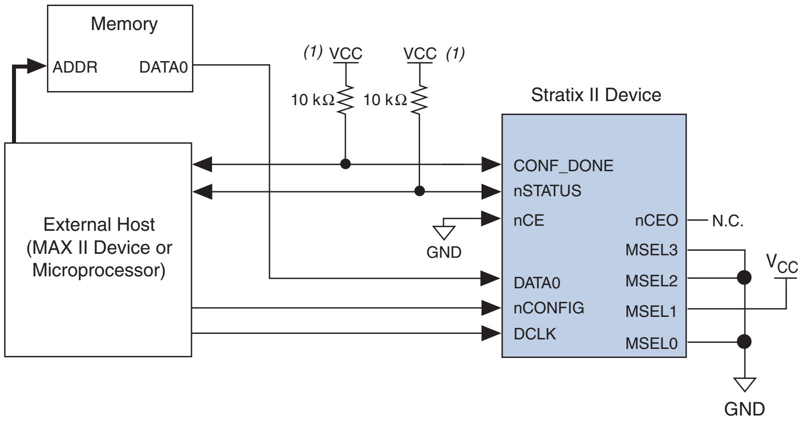
Note to Table 2–10:

(1) This scheme requires that you drive the `RUnLU` pin to specify either remote update or local update. For more information about remote system upgrade in Stratix II devices, see *Chapter 8, Remote System Upgrades with Stratix II Devices* in Volume 2 of the *Stratix II Device Handbook*.

PS Configuration Using a MAX II Device as an External Host

In the PS configuration scheme, a MAX II device can be used as an intelligent host that controls the transfer of configuration data from a storage device, such as flash memory, to the target Stratix II device. Configuration data can be stored in RBF, HEX, or TTF format. Figure 2–17 shows the configuration interface connections between the Stratix II device and a MAX II device for single device configuration.

Figure 2–17. Single Device PS Configuration Using an External Host



Note to Figure 2–17:

(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the `PORSEL` pin setting; when `PORSEL` is driven low, the POR time is approximately 100 ms, if `PORSEL` is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold `nSTATUS` low, and tri-state all user I/O pins. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If `nIO_pullup` is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If `nIO_pullup` is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *Stratix II Device Handbook.*

The configuration cycle consists of three stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the MAX II device must generate a low-to-high transition on the `nCONFIG` pin.

V_{CCINT}, V_{CCIO}, and V_{CCPD} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once nSTATUS is released, the device is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the MAX II device should place the configuration data one bit at a time on the DATA0 pin. The least significant bit (LSB) of each data byte must be sent first. For example, if the RBF contains the byte sequence 02 1B EE 01 FA, the serial bitstream you should transmit to the device is 0100-0000 1101-1000 0111-0111 1000-0000 0101-1111.

The Stratix II device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the device on the rising edge of DCLK. Data is continuously clocked into the target device until CONF_DONE goes high. After the device has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Stratix II devices, the initialization clock source is either the Stratix II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Stratix II device will provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. The **Enable user-supplied start-up clock** (**CLKUSR**) option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, CLKUSR will be enabled after the time specified as t_{CD2CU}. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. The MAX II device must be able to detect this low-to-high transition which signals the device has entered user mode. When initialization is complete, the device enters user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

To ensure DCLK and DATA0 are not left floating at the end of configuration, the MAX II device must drive them either high or low, whichever is convenient on your board. The DATA[0] pin is available as a user I/O pin after configuration. When the PS scheme is chosen in the Quartus II software, as a default this I/O pin is tri-stated in user mode and should be driven by the MAX II device. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

The configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the device drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the MAX II device that there is an error. If the **Auto-restart configuration after error** option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the Stratix II device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the MAX II device can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

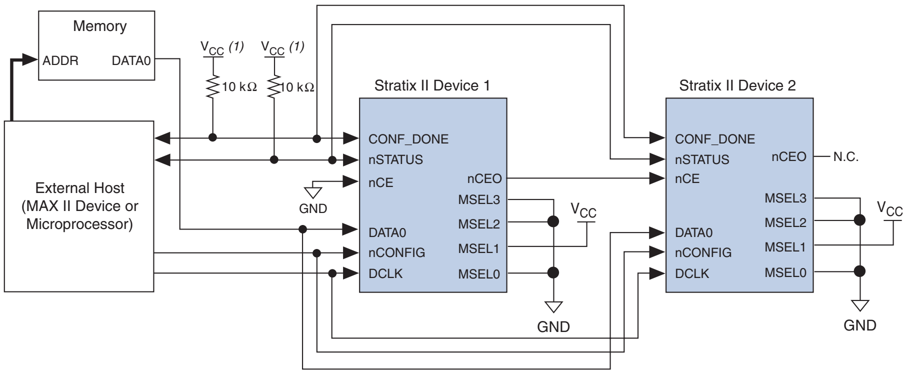
The MAX II device can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the MAX II device to detect errors and determine when programming completes. If all configuration data is sent, but CONF_DONE or INIT_DONE have not gone high, the MAX II device must reconfigure the target device.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the device is in user-mode, you can initiate a reconfiguration by transitioning the nCONFIG pin low-to-high. The nCONFIG pin must be low for at least 40 µs. When nCONFIG is pulled low, the device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the device, reconfiguration begins.

Figure 2–18 shows how to configure multiple devices using a MAX II device. This circuit is similar to the PS configuration circuit for a single device, except Stratix II devices are cascaded for multi-device configuration.

Figure 2–18. Multi-Device PS Configuration Using an External Host



Note to Figure 2–18:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

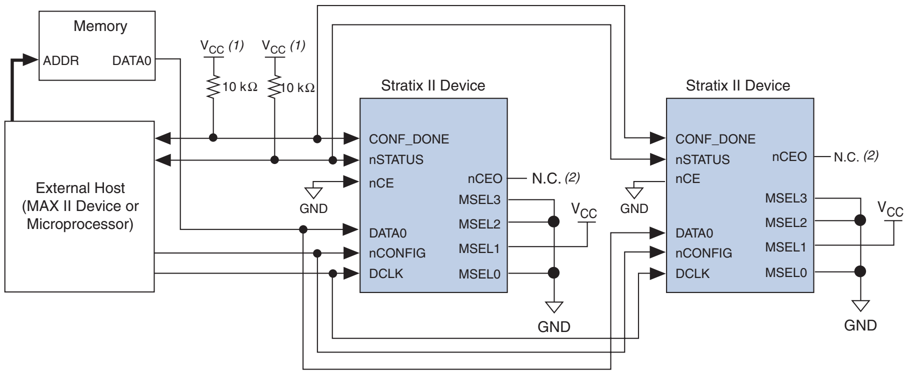
In multi-device PS configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle. Therefore, the transfer of data destinations is transparent to the MAX II device. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. Configuration signals can require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first device flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single device detecting an error.

If the **Auto-restart configuration after error** option is turned on, the devices release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the MAX II device can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

In your system, you can have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. Configuration signals can require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 2–19 shows multi-device PS configuration when both Stratix II devices are receiving the same configuration data.

Figure 2–19. Multiple-Device PS Configuration When Both devices Receive the Same Data



Notes to Figure 2–19:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

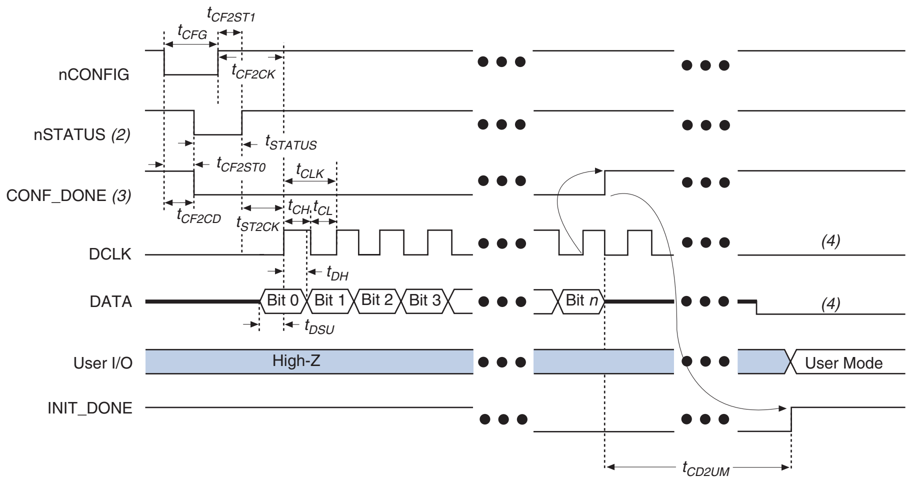
You can use a single configuration chain to configure Stratix II devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera device Chains* in the *Configuration Handbook*.

PS Configuration Timing

Figure 2–20 shows the timing waveform for PS configuration when using a MAX II device as an external host.

Figure 2–20. PS Configuration Timing Waveform *Note (1)*



Notes to Figure 2–20:

(1) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS` and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.
(2) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.
(3) Upon power-up, before and during configuration, `CONF_DONE` is low.
(4) `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[0]` is available as a user I/O pin after configuration and the state of this pin depends on the dual-purpose pin settings.

Table 2–11 defines the timing parameters for Stratix II devices for PS configuration.

Table 2–11. PS Timing Parameters for Stratix II Devices *Note (1)*

Symbol	Parameter	Min	Max	Units
t_{POR}	POR delay	12	100	ms
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		800	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		800	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	10	100 *(2)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		100 *(2)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	100		µs
t_{ST2CK}	`nSTATUS` high to first rising edge of `DCLK`	2		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	7		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	4		ns
t_{CL}	`DCLK` low time	4		ns
t_{CLK}	`DCLK` period	10		ns
f_{MAX}	`DCLK` frequency		100	MHz
t_R	Input rise time		40	ns
t_F	Input fall time		40	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	20	40	µs
t_{CD2CU}	`CONF_DONE` high to `CLKUSR` enabled	4 × maximum `DCLK` period		
t_{CD2UMC}	`CONF_DONE` high to user mode with CLKUSR option on	t_{CD2CU} + (299 × CLKUSR period)		

Notes to Table 2–11:
(1) This information is preliminary.
(2) This value is applicable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting the device.

Device configuration options and how to create configuration files are discussed further in *Software Settings* in Volume 2 of the *Configuration Handbook*.

An example PS design that uses a MAX II device as the external host for configuration will be available when devices are available.

PS Configuration Using a Microprocessor

In the PS configuration scheme, a microprocessor can control the transfer of configuration data from a storage device, such as flash memory, to the target Stratix II device.

All information in the "PS Configuration Using a MAX II Device as an External Host" section is also applicable when using a microprocessor as an external host. Refer to that section for all configuration and timing information.

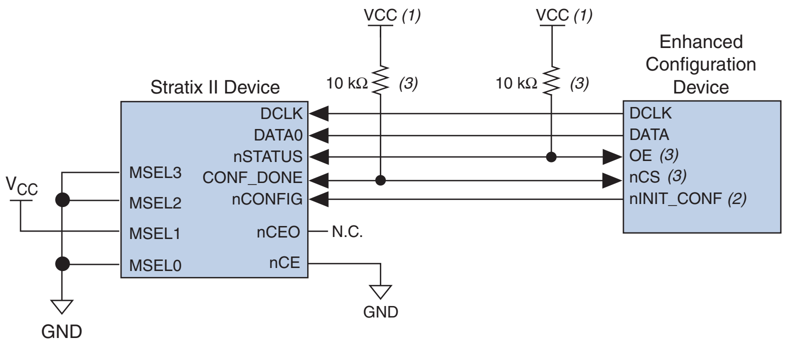
PS Configuration Using a Configuration Device

You can use an Altera configuration device, such as an enhanced configuration device, EPC2, or EPC1 device, to configure Stratix II devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 2–21 shows the configuration interface connections between the Stratix II device and a configuration device.

☞ The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the device.

For more information on the enhanced configuration device and flash interface pins (such as `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`), see the *Enhanced Configuration Devices* (*EPC4, EPC8, & EPC16*) *Data Sheet*.

Figure 2–21. Single Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 2–21:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used 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.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the Operating Conditions table of the *Enhanced Configuration Devices* (*EPC4, EPC8, & EPC16*) *Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet*.

When using enhanced configuration devices or EPC2 devices, nCONFIG of the device can be connected to nINIT_CONF of the configuration device, which allows the INIT_CONF JTAG instruction to initiate device configuration. 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor. An internal pull-up resistor on the nINIT_CONF pin is always active in enhanced configuration devices and EPC2 devices, which means an external pull-up resistor should not be used if nCONFIG is tied to nINIT_CONF.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the PORSEL pin setting. When PORSEL is driven low, the POR time is approximately 100 ms. If PORSEL is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold nSTATUS low, and tri-state all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The

POR time for EPC2 and EPC1 devices is 200 ms (maximum). The POR time for enhanced configuration devices can be set to either 100 ms or 2 ms, depending on its PORSEL pin setting. If the PORSEL pin is connected to GND, the POR delay is 100 ms. If the PORSEL pin is connected to V_{CC}, the POR delay is 2 ms. During this time, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin.

When selecting a POR time, you need to ensure that the device completes power-up before the enhanced configuration device exits POR. Altera recommends that you choose a POR time for the Stratix II device of 12 ms, while selecting a POR time for the enhanced configuration device of 100 ms.

When both devices complete POR, they release their open-drain OE or nSTATUS pin, which is then pulled high by a pull-up resistor. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If nIO_pullup is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If nIO_pullup is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *DC & Switching Characteristics* chapter in the *Stratix II Device Handbook*.

When the power supplies have reached the appropriate operating voltages, the target device senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. The beginning of configuration can be delayed by holding the nCONFIG or nSTATUS pin low.

To begin configuration, power the V_{CCINT}, V_{CCIO}, and V_{CCPD} voltages (for the banks where the configuration and JTAG pins reside) to the appropriate voltage levels.

When nCONFIG goes high, the device comes out of reset and releases the nSTATUS pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the OE pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 10-kΩ pull-up resistor on the OE-nSTATUS line is required. Once nSTATUS is released, the device is ready to receive configuration data and the configuration stage begins.

When nSTATUS is pulled high, OE of the configuration device also goes high and the configuration device clocks data out serially to the device using its internal oscillator. The Stratix II device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the device on the rising edge of DCLK.

After the device has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 10-kΩ pull-up resistor on the nCS-CONF_DONE line is required. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Stratix II devices, the initialization clock source is either the Stratix II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If you are using internal oscillator, the Stratix II device will supply itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. You can turn on the **Enable user-supplied start-up clock** (**CLKUSR**) option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, CLKUSR will be enabled after the time specified as t_{CD2CU}. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If you are using the INIT_DONE pin, it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the device has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. Enhanced configuration devices and EPC2 devices drive `DCLK` low and `DATA0` high (EPC1 devices drive `DCLK` low and tri-state `DATA`) at the end of configuration.

If an error occurs during configuration, the device drives its `nSTATUS` pin low, resetting itself internally. Since the `nSTATUS` pin is tied to `OE`, the configuration device will also be reset. If the **Auto-restart configuration after error** option, available in the Quartus II software, from the **General** tab of the **Device & Pin Options** dialog box is turned on, the device automatically initiates reconfiguration if an error occurs. The Stratix II device will release its `nSTATUS` pin after a reset time-out period (maximum of 40 µs). When the `nSTATUS` pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor `nSTATUS` for errors and then pulse `nCONFIG` low for at least 40 µs to restart configuration. The external system can pulse `nCONFIG` if `nCONFIG` is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that `CONF_DONE` has not gone high, it recognizes that the device has not configured successfully. 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. In this case, the configuration device pulls its `OE` pin low, driving the target device's `nSTATUS` pin low. If the **Auto-restart configuration after error** option is set in the software, the target device resets and then releases its `nSTATUS` pin after a reset time-out period (maximum of 40 µs). When `nSTATUS` returns to a logic high level, the configuration device tries to reconfigure the device.

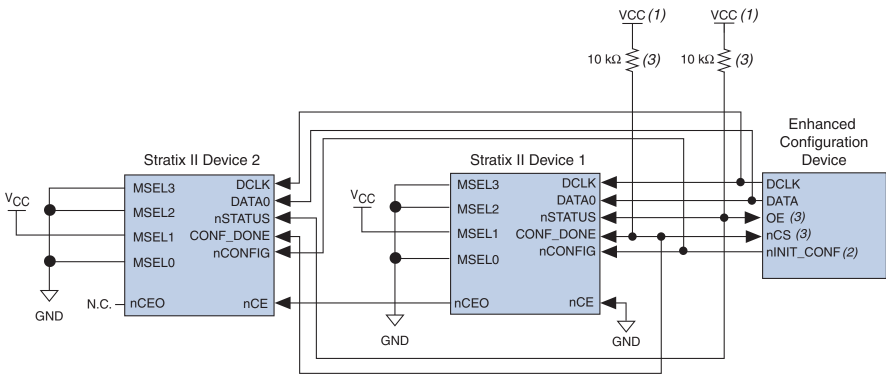
When `CONF_DONE` is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully. Therefore, your system should not pull `CONF_DONE` low to delay initialization. Instead, use the `CLKUSR` option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their `CONF_DONE` pins are tied together.

☞ If you are using the optional `CLKUSR` pin and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure that `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 µs).

When the device is in user-mode, pulling the `nCONFIG` pin low will initiate a reconfiguration. The `nCONFIG` pin should be low for at least 40 µs. When `nCONFIG` is pulled low, the device also pulls `nSTATUS` and `CONF_DONE` low and all I/O pins are tri-stated. Since `CONF_DONE` is pulled low, this will activate the configuration device since it will see its `nCS` pin drive low. Once `nCONFIG` returns to a logic high level and `nSTATUS` is released by the device, reconfiguration begins.

Figure 2–22 shows how to configure multiple devices with an enhanced configuration device. This circuit is similar to the configuration device circuit for a single device, except Stratix II devices are cascaded for multi-device configuration.

Figure 2–22. Multi-Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 2–22:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used 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.

Enhanced configuration devices cannot be cascaded.

When performing multi-device configuration, you must generate the configuration device's POF from each project's SOF. You can combine multiple SOFs using the **Convert Programming Files** window in the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, see the *Software Settings* chapter of the *Configuration Handbook*.

In multi-device PS configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, prompting the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. Configuration signals can require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DCLK and DATA lines are buffered for every fourth device.

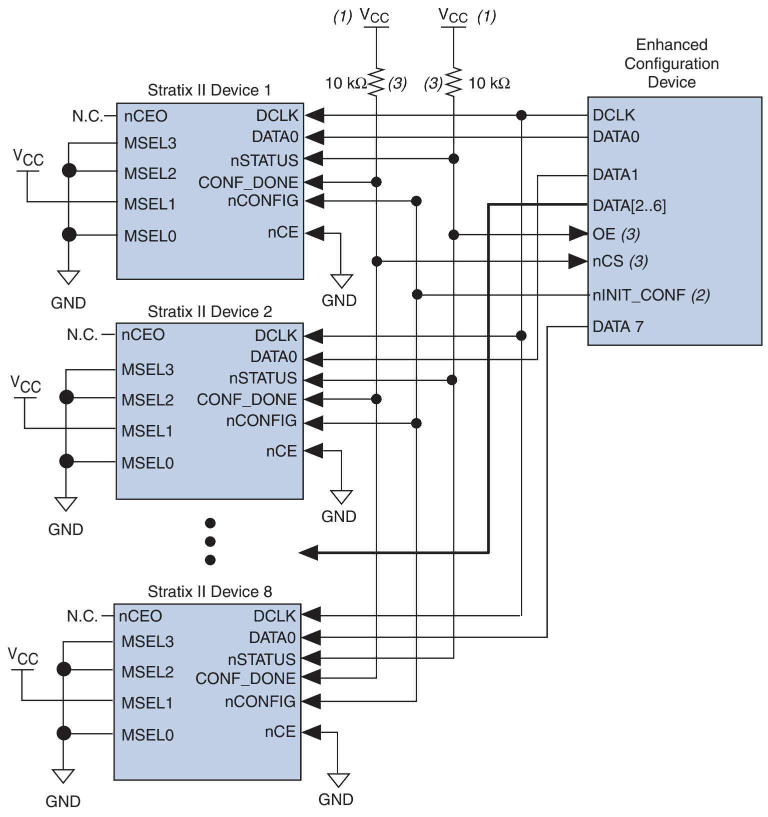
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first device flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the enhanced configuration device and drives nSTATUS low on all devices, causing them to enter a reset state. This behavior is similar to a single device detecting an error.

If the **Auto-restart configuration after error** option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The devices will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the **Auto-restart configuration after error** option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

The enhanced configuration devices also support parallel configuration of up to eight devices. The n-bit (*n* = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure devices or a chain of devices. In addition, these devices do not have to be the same device family or density as they can be any combination of Altera 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. Figure 2–23 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 2–23. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device



Notes to Figure 2–23:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used 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.

The Quartus II software only allows the selection of n-bit PS configuration modes, where n must be 1, 2, 4, or 8. 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 2–12 to select the appropriate configuration mode for the fastest configuration times.

Table 2–12. 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 2–12:

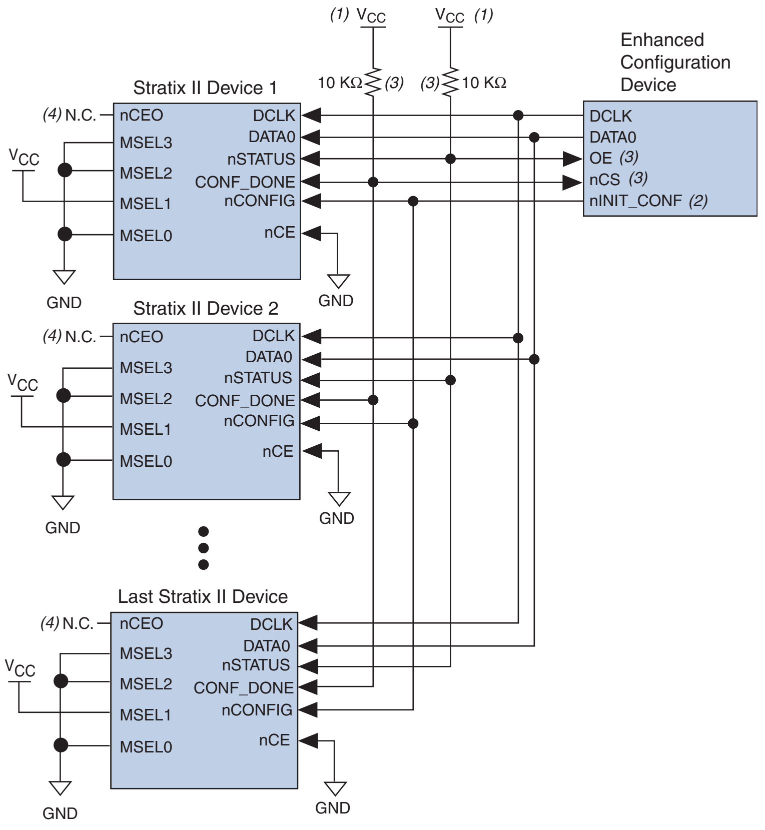
(1) Assume that each DATA line is only configuring one device, not a daisy chain of devices.

For example, if you configure three devices, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding SOF data is transmitted from the configuration device to the device. For `DATA3`, you can leave the corresponding Bit3 line blank in the Quartus II software. On the PCB, leave the `DATA3` line from the enhanced configuration device unconnected.

Alternatively, you can daisy chain two 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 devices with `DATA` Bit0 (two EP2S10 devices) and the third device (EP2S20 device) with `DATA` Bit1. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but can increase the total system configuration time.

A system may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. Configuration signals can require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the `DCLK` and `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 2–24 shows multi-device PS configuration when the Stratix II devices are receiving the same configuration data.

Figure 2–24. Multiple-Device PS Configuration Using an Enhanced Configuration Device When devices Receive the Same Data



Notes to Figure 2–24:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used 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) The `nCEO` pins of all devices are left unconnected when configuring the same configuration data into multiple devices.

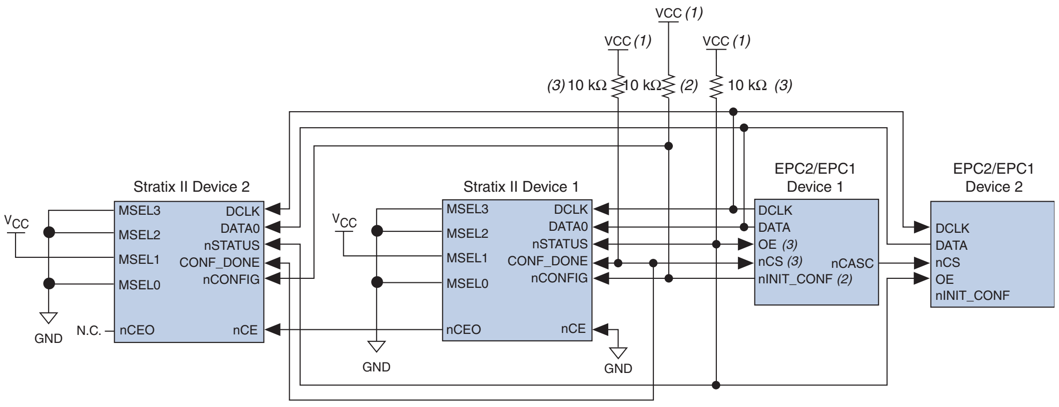
You can cascade several EPC2 or EPC1 devices to configure multiple Stratix II devices. The first configuration device in the chain is the master configuration device, while the subsequent devices are the slave devices. The master configuration device sends DCLK to the Stratix II devices and to the slave configuration devices. The first EPC device's nCS pin is connected to the CONF_DONE pins of the devices, while its nCASC pin is connected to nCS of the next configuration device in the chain. The last device's nCS input comes from the previous device, while its nCASC pin is left floating. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the next configuration device. Since a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

☞ Enhanced configuration devices cannot be cascaded.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect CONF_DONE going high at the end of configuration, it resets the entire chain by pulling its OE pin low. This low signal drives the OE pin low on the slave configuration device(s) and drives nSTATUS low on all devices, causing them to enter a reset state. This behavior is similar to the device detecting an error in the configuration data.

Figure 2–25 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 2–25. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 2–25:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor should not be used 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. External 10-kΩ pull-up resistors should be used. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

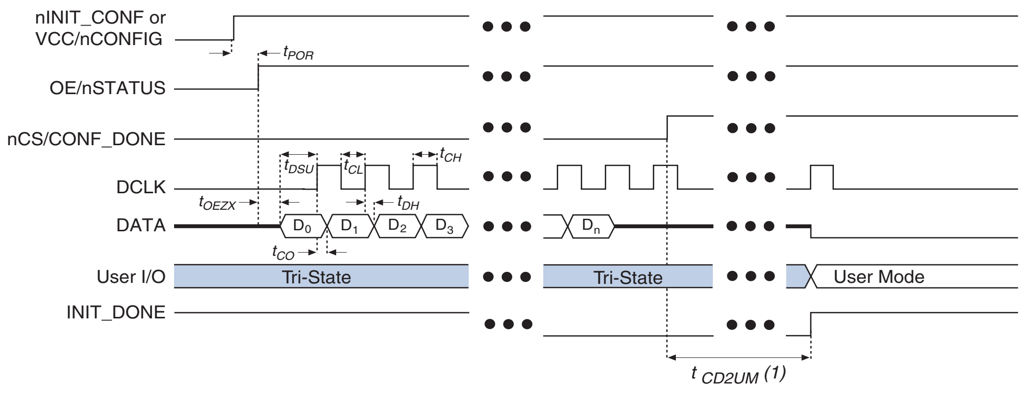
When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the device can be connected to `nINIT_CONF` of the configuration device, allowing the `INIT_CONF` JTAG instruction to initiate device configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If you are not using `nINIT_CONF`, or if it isn't available(e.g. on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up resistor on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices, which means that you shouldn't be using an external pull-up resistor if `nCONFIG` is tied to `nINIT_CONF`. If you are using multiple EPC2 devices to configure a Stratix II device(s), only the first EPC2 has its `nINIT_CONF` pin tied to the device's `nCONFIG` pin.

You can use a single configuration chain to configure Stratix II devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera device Chains* chapter in the *Configuration Handbook*.

Figure 2–26 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 2–26. Stratix II PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 2–26:

(1) The initialization clock can come from the Stratix II internal oscillator or the CLKUSR pin.

For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* or the *Configuration Devices for SRAM-Based LUT Devices Data Sheet* in the *Configuration Handbook*.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* chapter of the *Configuration Handbook*.

PS Configuration Using a Download Cable

In this section, the generic term "download cable" includes the Altera USB-Blaster™ universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster™ II parallel port download cable, and the ByteBlaster MV parallel port download cable.

In PS configuration with a download cable, an intelligent host (such as a PC) transfers data from a storage device to the device via the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the PORSEL pin setting. When PORSEL is driven low, the POR time is approximately 100 ms. If PORSEL is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold nSTATUS low, and tri-state all user I/O pins. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If nIO_pullup is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If nIO_pullup is driven high, the weak pull-up resistors are disabled.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *DC & Switching Characteristics* chapter in the *Stratix II Device Handbook*.

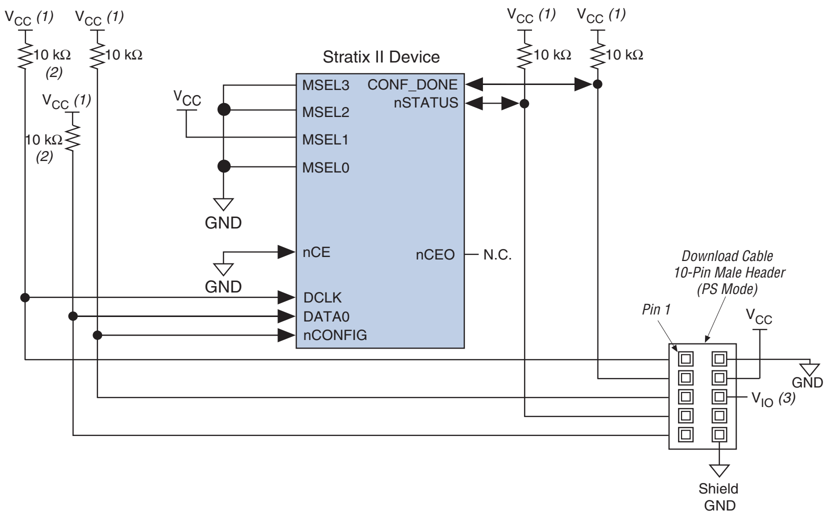
The configuration cycle consists of three stages: reset, configuration and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the nCONFIG pin.

To begin configuration, power the V_{CCINT}, V_{CCIO}, and V_{CCPD} voltages (for the banks where the configuration and JTAG pins reside) to the appropriate voltage levels.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once nSTATUS is released the device is ready to receive configuration data and the configuration stage begins. The programming hardware or download cable then places the configuration data one bit at a time on the device's DATA0 pin. The configuration data is clocked into the target device until CONF_DONE goes high.

When using a download cable, setting the **Auto-restart configuration after error** option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs. Additionally, the **Enable user-supplied start-up clock** (**CLKUSR**) option has no affect on the device initialization since this option is disabled in the SOF when programming the device using the Quartus II programmer and download cable. Therefore, if you turn on the CLKUSR option, you do not need to provide a clock on CLKUSR when you are configuring the device with the Quartus II programmer and a download cable. Figure 2–27 shows PS configuration for Stratix II devices using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Figure 2–27. PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV Cable



Notes to Figure 2–27:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (V_{IO} pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This ensures that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

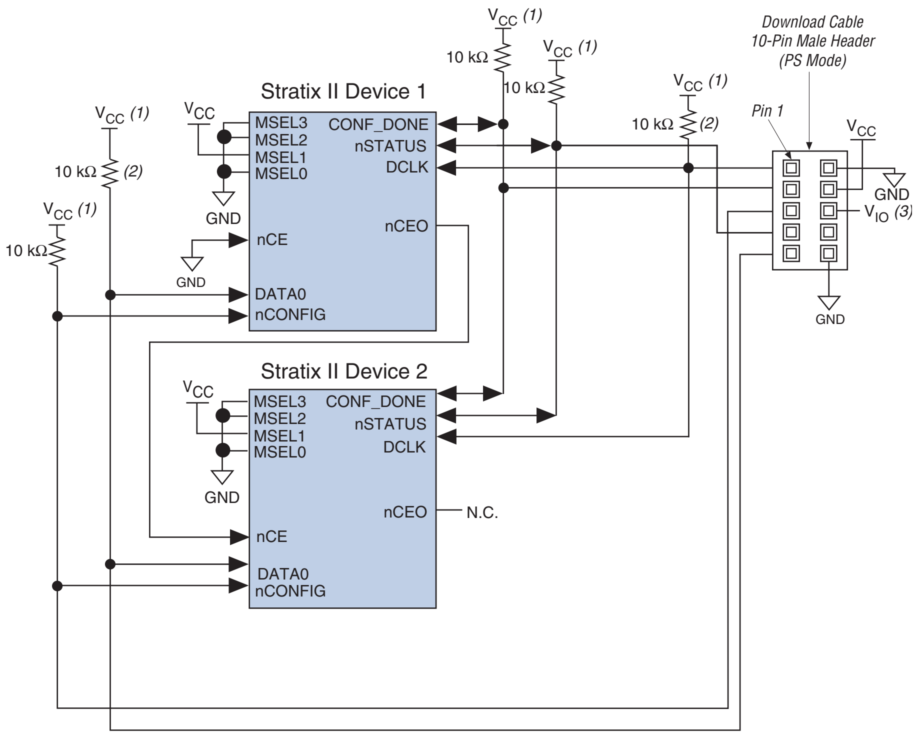
(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to nCE when it is used for active serial programming, otherwise it is a no connect.

You can use a download cable to configure multiple Stratix II devices by connecting each device's nCEO pin to the subsequent device's nCE pin. The first device's nCE pin is connected to GND while its nCEO pin is connected to the nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. All other configuration pins, nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE are connected to every device in the chain. Because all CONF_DONE pins are tied together, all devices in the chain initialize and enter user mode at the same time.

In addition, because the nSTATUS pins are tied together, the entire chain halts configuration if any device detects an error. The **Auto-restart configuration after error** option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs.

Figure 2–28 shows how to configure multiple Stratix II devices with a download cable.

Figure 2–28. Multi-Device PS Configuration using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV Cable



Notes to Figure 2–28:

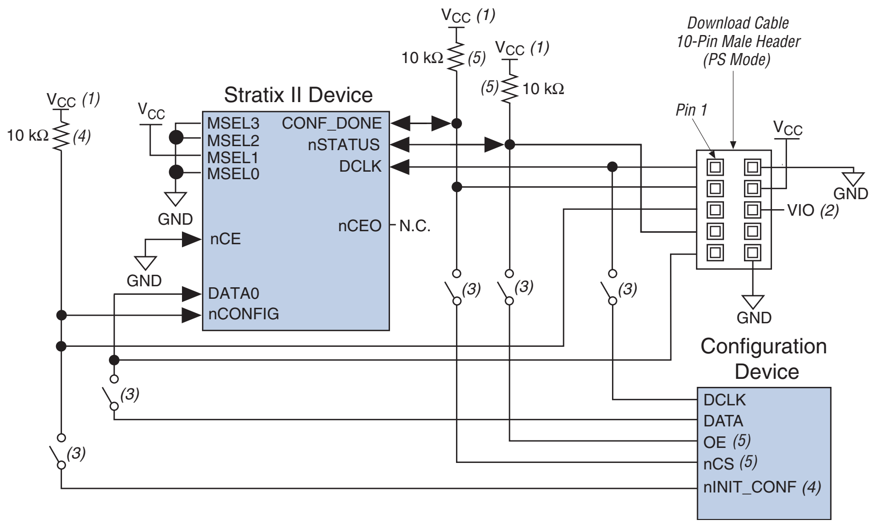
(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (V_{IO} pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to `nCE` when it is used for active serial programming, otherwise it is a no connect.

If you are using a download cable to configure device(s) on a board that also has configuration devices, electrically isolate the configuration device from the target device(s) and cable. One way of isolating the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer chip allows bidirectional transfers on the nSTATUS and CONF_DONE signals. Another option is to add switches to the five common signals (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) between the cable and the configuration device. The last option is to remove the configuration device from the board when configuring the device with the cable. Figure 2–29 shows a combination of a configuration device and a download cable to configure an device.

Figure 2–29. PS Configuration with a Download Cable & Configuration Device Circuit



Notes to Figure 2–29:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to nCE when it is used for active serial programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to a Stratix II device. Instead, 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.

(4) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active. This means an external pull-up resistor should not be used 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(5) The enhanced configuration devices' and EPC2 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-up resistors on configuration device** option when generating programming files.

For more information on how to use the USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cables, refer to the following data sheets:

- 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

Passive Parallel Asynchronous Configuration

Passive parallel asynchronous (PPA) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target Stratix II device.

Configuration data can be stored in RBF, HEX, or TTF format. The host system outputs byte-wide data and the accompanying strobe signals to the device. When using PPA, pull the DCLK pin high through a 10-kΩ pull-up resistor to prevent unused configuration input pins from floating.

You cannot use the Stratix II decompression and design security feature if you are configuring your Stratix II device using PPA mode.

Table 2–13 shows the MSEL pin settings when using the PS configuration scheme.

Table 2–13. Stratix II MSEL Pin Settings for PPA Configuration Schemes

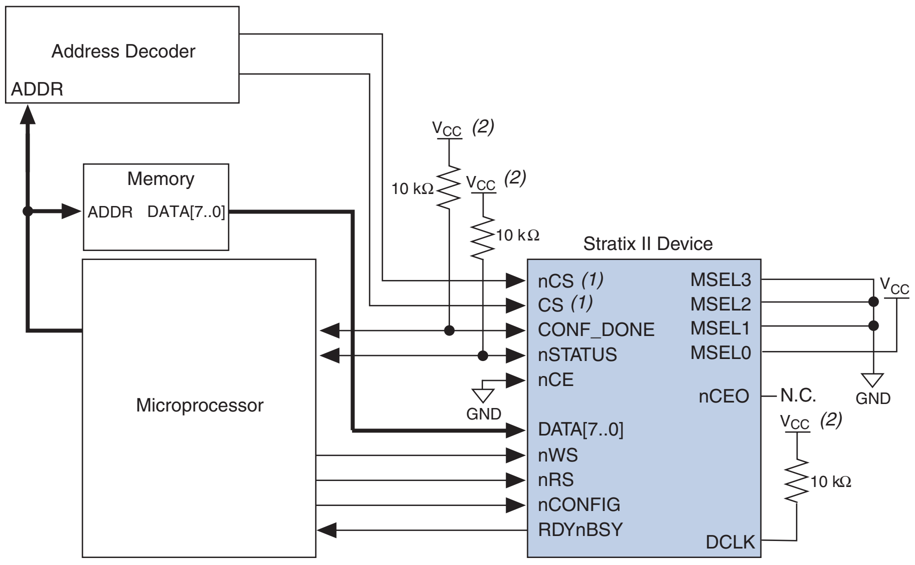
Configuration Scheme	MSEL3	MSEL2	MSEL1	MSEL0
PPA	0	0	0	1
Remote System Upgrade PPA *(1)*	0	1	0	1

Notes to Table 2–13:

(1) This scheme requires that you drive the RUnLU pin to specify either remote update or local update. For more information about remote system upgrade in Stratix II devices, see the *Chapter 8, Remote System Upgrades with Stratix II Devices* in Volume 2 of the *Stratix II Device Handbook.*

Figure 2–30 shows the configuration interface connections between the device and a microprocessor for single device PPA configuration. The microprocessor or an optional address decoder can control the device's chip select pins, nCS and CS. The address decoder allows the microprocessor to select the Stratix II device by accessing a particular address, which simplifies the configuration process. Hold the nCS and CS pins active during configuration and initialization.

Figure 2–30. Single Device PPA Configuration Using a Microprocessor



Notes to Figure 2–30:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

During PPA configuration, it is only required to use either the nCS or CS pin. Therefore, if you are using only one chip-select input, the other must be tied to the active state. For example, nCS can be tied to ground while CS is toggled to control configuration. The device's nCS or CS pins can be toggled during PPA configuration if the design meets the specifications set for t_{CSSU}, t_{WSP}, and t_{CSH} listed in Table 2–14.

Upon power-up, the Stratix II device goes through a POR. The POR delay is dependent on the PORSEL pin setting. When PORSEL is driven low, the POR time is approximately 100 ms. If PORSEL is driven high, the POR time is approximately 12 ms. During POR, the device will reset, hold nSTATUS low, and tri-state all user I/O pins. Once the device successfully exits POR, all user I/O pins continue to be tri-stated. If nIO_pullup is driven low during power-up and configuration, the user I/O pins and dual-purpose I/O pins will have weak pull-up resistors which are on (after POR) before and during configuration. If nIO_pullup is driven high, the weak pull-up resistors are disabled.

☞ The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *DC & Switching Characteristics* chapter in the *Stratix II Device Handbook*.

The configuration cycle consists of three stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

☞ To begin configuration, power the V_{CCINT}, V_{CCIO}, and V_{CCPD} voltages (for the banks where the configuration and JTAG pins reside) to the appropriate voltage levels.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` is released the device is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should then assert the target device's `nCS` pin low and/or `CS` pin high. Next, the microprocessor places an 8-bit configuration word (one byte) on the target device's `DATA[7..0]` pins and pulses the `nWS` pin low.

On the rising edge of `nWS`, the target device latches in a byte of configuration data and drives its `RDYnBSY` signal low, which indicates it is processing the byte of configuration data. The microprocessor can then perform other system functions while the Stratix II device is processing the byte of configuration data.

During the time `RDYnBSY` is low, the Stratix II device internally processes the configuration data using its internal oscillator (typically 100 MHz). When the device is ready for the next byte of configuration data, it will drive `RDYnBSY` high. If the microprocessor senses a high signal when it polls `RDYnBSY`, the microprocessor sends the next byte of configuration data to the device.

Alternatively, the `nRS` signal can be strobed low, causing the `RDYnBSY` signal to appear on `DATA7`. Because `RDYnBSY` does not need to be monitored, this pin doesn't need to be connected to the microprocessor. Do not drive data onto the data bus while `nRS` is low because it will cause contention on the `DATA7` pin. If you are not using the `nRS` pin to monitor configuration, it should be tied high.

To simplify configuration and save an I/O port, the microprocessor can wait for the total time of t_{BUSY} (max) + t_{RDY2WS} + t_{W2SB} before sending the next data byte. In this set-up, `nRS` should be tied high and `RDYnBSY` does not need to be connected to the microprocessor. The t_{BUSY}, t_{RDY2WS}, and t_{W2SB} timing specifications are listed in Table 2–14 on page 2–75.

Next, the microprocessor checks nSTATUS and CONF_DONE. If nSTATUS is not low and CONF_DONE is not high, the microprocessor sends the next data byte. However, if nSTATUS is not low and all the configuration data has been received, the device is ready for initialization. The CONF_DONE pin will go high one byte early in parallel configuration (FPP and PPA) modes. The last byte is required for serial configuration (AS and PS) modes. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin. The open-drain CONF_DONE pin is pulled high by an external 10-kΩ pull-up resistor.

In Stratix II devices, the initialization clock source is either the Stratix II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Stratix II device will provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device.

You also have the flexibility to synchronize initialization of multiple devices or to delay initialization by using the CLKUSR option. The **Enable user-supplied start-up clock** (**CLKUSR**) option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR does not affect the configuration process. After CONF_DONE goes high, CLKUSR will be enabled after the time specifiedas t_{CD2CU}. After this time period elapses, the Stratix II devices require 299 clock cycles to initialize properly and enter user mode. Stratix II devices support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This **Enable INIT_DONE Output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the device has entered user mode. When initialization is complete, the device enters user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

To ensure DATA[7..0] is not left floating at the end of configuration, the microprocessor must drive them either high or low, whichever is convenient on your board. After configuration, the nCS, CS, nRS, nWS, RDYnBSY, and DATA[7..0] pins can be used as user I/O pins. When choosing the PPA scheme in the Quartus II software as a default, these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

If an error occurs during configuration, the device drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the **Auto-restart configuration after error** option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

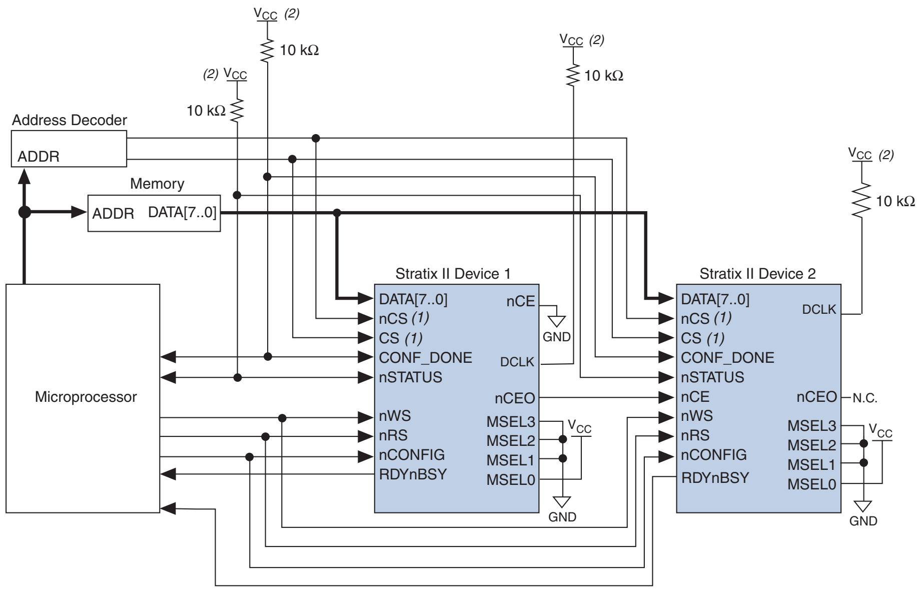
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. To detect errors and determine when programming completes, monitor the CONF_DONE pin with the microprocessor. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE has not gone high, the microprocessor must reconfigure the target device.

☞ If you are using the optional CLKUSR pin and nCONFIG is pulled low to restart configuration during device initialization, ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the device is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should go low for at least 40 µs. When nCONFIG is pulled low, the device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the device, reconfiguration begins.

Figure 2–31 shows how to configure multiple Stratix II devices using a microprocessor. This circuit is similar to the PPA configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 2–31. Multi-Device PPA Configuration Using a Microprocessor



Notes to Figure 2–31:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

In multi-device PPA configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle. Therefore, the transfer of data destinations is transparent to the microprocessor.

Each device's RDYnBSY pin can have a separate input to the microprocessor. Alternatively, if the microprocessor is pin limited, all the RDYnBSY pins can feed an AND gate and the output of the AND gate can feed the microprocessor. For example, if you have two devices in a PPA configuration chain, the second device's RDYnBSY pin will be high during the time that the first device is being configured. When the first device has been successfully configured, it will drive nCEO low to activate the next device in the chain and drive its RDYnBSY pin high. Therefore, since

RDYnBSY signal is driven high before configuration and after configuration before entering user-mode, the device being configured will govern the output of the AND gate.

The nRS signal can be used in multi-device PPA chain since the Stratix II device will tri-state its DATA[7..0] pins before configuration and after configuration before entering user-mode to avoid contention. Therefore, only the device that is currently being configured will respond to the nRS strobe by asserting DATA7.

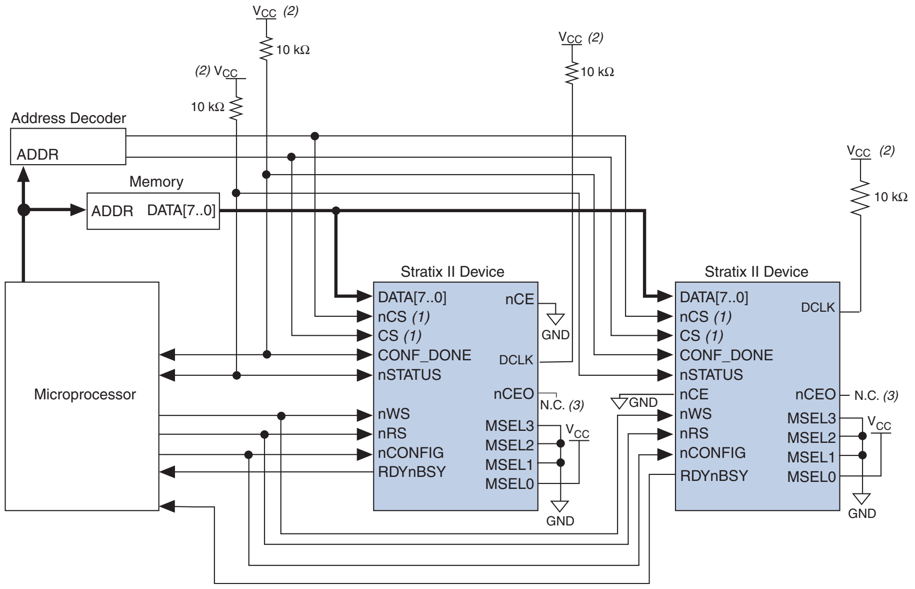
All other configuration pins (nCONFIG, nSTATUS, DATA[7..0], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. It is not necessary to tie nCS and CS together for every device in the chain, as each device's nCS and CS input can be driven by a separate source. Configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first device flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single device detecting an error.

If the **Auto-restart configuration after error** option is turned on, the devices release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 40 µs) on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DATA[7..0], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. Configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Ensure that the DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 2–32 shows multi-device PPA configuration when both devices are receiving the same configuration data.

Figure 2–32. Multiple-Device PPA Configuration Using a Microprocessor When Both devices Receive the Same Data



Notes to Figure 2–32:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device and the external host.

(3) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

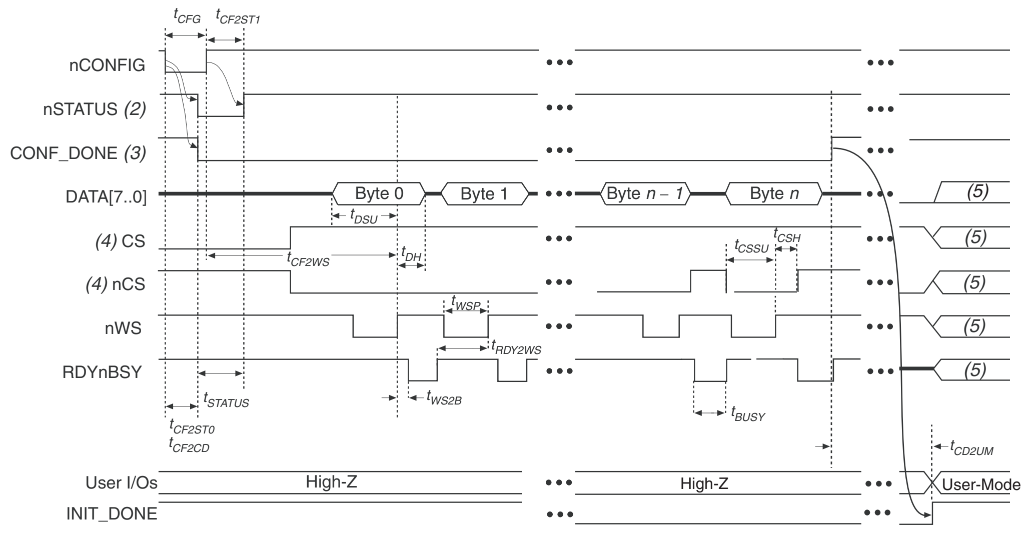
You can use a single configuration chain to configure Stratix II devices with other Altera devices that support PPA configuration, such as Stratix, Mercury™, APEX™ 20K, ACEX® 1K, and FLEX® 10KE devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera Device Chains* chapter in the *Configuration Handbook*.

PPA Configuration Timing

Figure 2–33 shows the timing waveform for the PPA configuration scheme using a microprocessor.

Figure 2–33. Stratix II PPA Configuration Timing Waveform Using nWS *Note (1)*



Notes to Figure 2–33:

(1) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS` and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.

(2) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.

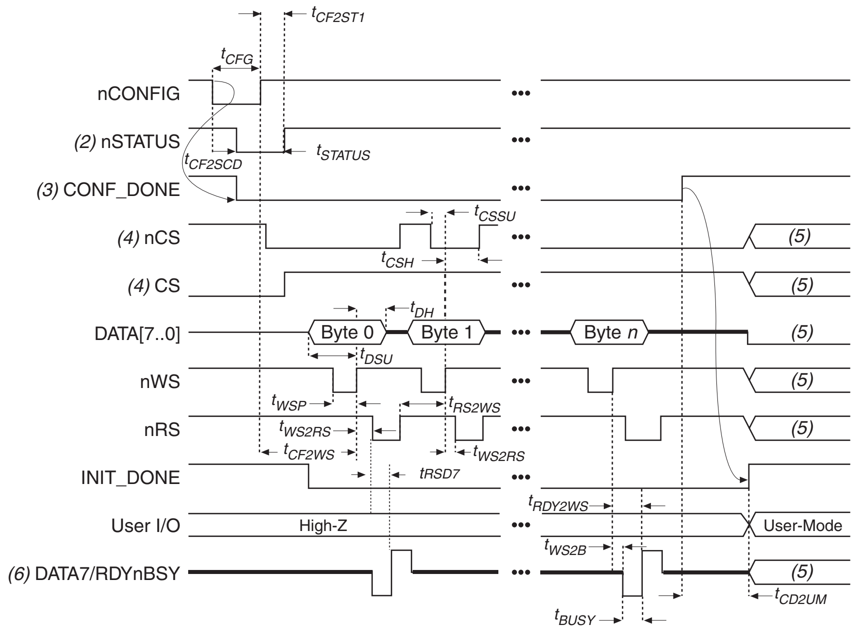
(3) Upon power-up, before and during configuration, `CONF_DONE` is low.

(4) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(5) `DATA[7..0]`, `CS`, `nCS`, `nWS`, `nRS`, and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

Figure 2–34 shows the timing waveform for the PPA configuration scheme when using a strobed `nRS` and `nWS` signal.

Figure 2–34. Stratix II PPA Configuration Timing Waveform Using nRS & nWS Note (1)



Notes to Figure 2–34:

(1) The beginning of this waveform shows the device in user-mode. In user-mode, nCONFIG, nSTATUS and CONF_DONE are at logic high levels. When nCONFIG is pulled low, a reconfiguration cycle begins.
(2) Upon power-up, the Stratix II device holds nSTATUS low for the time of the POR delay.
(3) Upon power-up, before and during configuration, CONF_DONE is low.
(4) The user can toggle nCS or CS during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.
(5) DATA[7..0], CS, nCS, nWS, nRS, and RDYnBSY are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.
(6) DATA7 is a bidirectional pin. It is an input for configuration data input, but it is an output to show the status of RDYnBSY.

Table 2–14 defines the timing parameters for Stratix II devices for PPA configuration.

Table 2–14. PPA Timing Parameters for Stratix II Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{POR}	POR delay	12	100	ms
t_{CF2CD}	nCONFIG low to CONF_DONE low		800	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		800	ns
t_{CFG}	nCONFIG low pulse width	2		µs

Table 2–14. PPA Timing Parameters for Stratix II Devices (Part 2 of 2) *Note (1)*

Symbol	Parameter	Min	Max	Units
t_{STATUS}	nSTATUS low pulse width	10	100 *(2)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		100 *(2)*	µs
t_{CSSU}	Chip select setup time before rising edge on nWS	10		ns
t_{CSH}	Chip select hold time after rising edge on nWS	0		ns
t_{CF2WS}	nCONFIG high to first rising edge on nWS	100		µs
t_{DSU}	Data setup time before rising edge on nWS	10		ns
t_{DH}	Data hold time after rising edge on nWS	0		ns
t_{WSP}	nWS low pulse width	15		ns
t_{WS2B}	nWS rising edge to RDYnBSY low		20	ns
t_{BUSY}	RDYnBSY low pulse width	7	45	ns
t_{RDY2WS}	RDYnBSY rising edge to nWS rising edge	15		ns
t_{WS2RS}	nWS rising edge to nRS falling edge	15		ns
t_{RS2WS}	nRS rising edge to nWS rising edge	15		ns
t_{RSD7}	nRS falling edge to DATA7 valid with RDYnBSY signal		20	ns
t_R	Input rise time		40	ns
t_F	Input fall time		40	ns
t_{CD2UM}	CONF_DONE high to user mode *(3)*	20	40	µs
t_{CD2CU}	CONF_DONE high to CLKUSR enabled	40		ns
t_{CD2UMC}	CONF_DONE high to user mode with CLKUSR option on	t_{CD2CU} + (299 × CLKUSR period)		

Notes to Table 2–14:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* chapter in the *Configuration Handbook*.

JTAG Configuration

The JTAG has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The JTAG circuitry can also be used to shift configuration data into the device. The Quartus II software automatically generates SOFs that can be used for JTAG configuration with a download cable in the Quartus II software programmer.

For more information on JTAG boundary-scan testing, see the following documents:

- Chapter 9, IEEE 1149.1 (JTAG) Boundary-Scan Testing for Stratix II Devices in the *Stratix II Device Handbook, Volume II*
- *Jam Programming & Testing Language Specification*

Stratix II devices are designed such that JTAG instructions have precedence over any device configuration modes. Therefore, JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of Stratix II devices during PS configuration, PS configuration will be terminated and JTAG configuration will begin.

You cannot use the Stratix II decompression feature or the design security feature if you are configuring your Stratix II device when using JTAG-based configuration.

A device operating in JTAG mode uses four required pins, `TDI`, `TDO`, `TMS`, and `TCK`, and one optional pin, `TRST`. The `TCK` pin has an internal weak pull-down resistor, while the `TDI`, `TMS`, and `TRST` pins have weak internal pull-up resistors (typically 25 kΩ). The `TDO` output pin is powered by V_{CCIO} in I/O bank 4. All of the JTAG input pins are powered by the 3.3-V V_{CCPD} pin. All user I/O pins are tri-stated during JTAG configuration. Table 2–15 explains each JTAG pin's function.

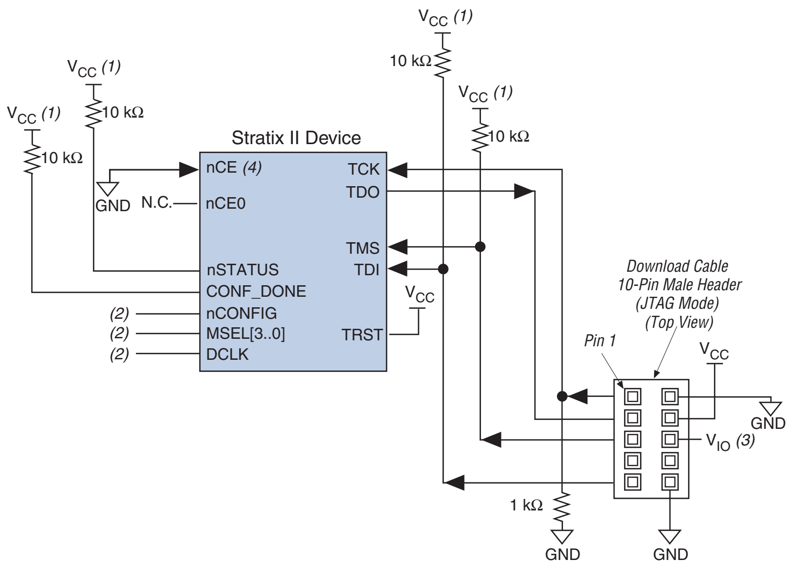
☞ The TDO output is powered by the V_{CCIO} power supply of I/O bank 4. For recommendations on how to connect a JTAG chain with multiple voltages across the devices in the chain, refer to the *IEEE 1149.1 (JTAG) Boundary Scan Testing in Stratix II Devices* chapter in Volume 2 of the *Stratix II Handbook*.

Table 2–15. Dedicated JTAG Pins

Pin Name	Pin Type	Description
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The TRST pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

During JTAG configuration, data can be downloaded to the device on the PCB through the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV download cable. Configuring devices through a cable is similar to programming devices in-system, except the TRST pin should be connected to V_{CC}. This ensures that the TAP controller is not reset. Figure 2–35 shows JTAG configuration of a single Stratix II device.

Figure 2–35. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 2–35:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (V_{IO} pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL[3..0]` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL[3..0]` to ground. Pull `DCLK` either high or low, whichever is convenient on your board.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to `nCE` when it is used for active serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

To configure a single device in a JTAG chain, the programming software places all other devices in bypass mode. In bypass mode, devices pass programming data from the `TDI` pin to the `TDO` pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the `TDO` pin one clock cycle later.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, the software checks the state of `CONF_DONE` through the JTAG port. When Quartus II generates a (**.jam**) file for a multi-device chain, it contains instructions so that all the devices in the chain will be initialized at the same time. If `CONF_DONE` is not high, the Quartus II software indicates that configuration has failed. If

CONF_DONE is high, the software indicates that configuration was successful. After the configuration bit stream is transmitted serially via the JTAG TDI port, the TCK port is clocked an additional 299 cycles to perform device initialization.

Stratix II devices have dedicated JTAG pins that always function as JTAG pins. Not only can you perform JTAG testing on Stratix II devices before and after, but also during configuration. While other device families do not support JTAG testing during configuration, Stratix II devices support the bypass, idcode, and sample instructions during configuration without interrupting configuration. All other JTAG instructions may only be issued by first interrupting configuration and reprogramming I/O pins using the CONFIG_IO instruction.

The CONFIG_IO instruction allows I/O buffers to be configured via the JTAG port and when issued, interrupts configuration. This instruction allows you to perform board-level testing prior to configuring the Stratix II device or waiting for a configuration device to complete configuration. Once configuration has been interrupted and JTAG testing is complete, the part must be reconfigured via JTAG (PULSE_CONFIG instruction) or by pulsing nCONFIG low.

The chip-wide reset (DEV_CLRn) and chip-wide output enable (DEV_OE) pins on Stratix II devices do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of Stratix II devices, consider the dedicated configuration pins. Table 2–16 shows how these pins should be connected during JTAG configuration.

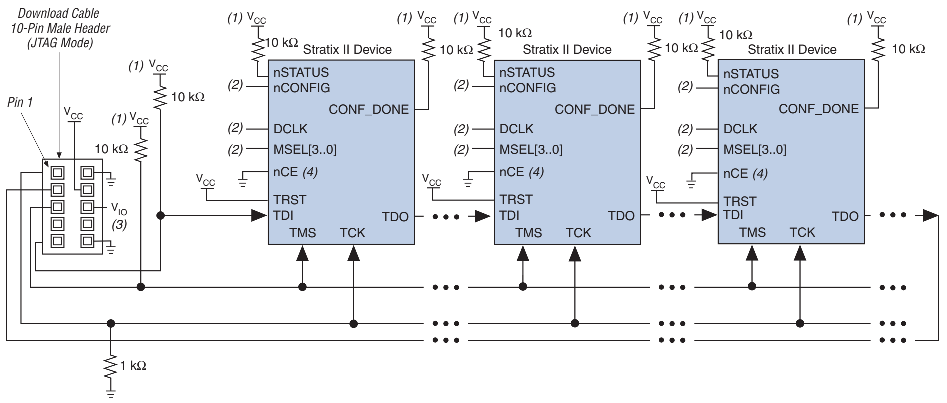
When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. When four or more devices are connected in a JTAG chain, Altera recommends buffering the TCK, TDI, and TMS pins with an on-board buffer.

JTAG-chain device programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 2–36 shows multi-device JTAG configuration.

Table 2–16. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all Stratix II devices in the chain, nCE should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device FPP, AS, PS, or PPA configuration chains, the nCE pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all Stratix II devices in the chain, nCEO can be left floating or connected to the nCE of the next device.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, tie these pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling up via a resistor, or driven high by some control circuitry.
nSTATUS	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each nSTATUS pin should be pulled up to V_{CC} individually.
CONF_DONE	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each CONF_DONE pin should be pulled up to V_{CC} individually. CONF_DONE going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.

Figure 2–36. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 2–36:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (V_{IO} pin), ByteBlaster II or ByteBlasterMV cable.

(2) The `nCONFIG, MSEL[3..0]` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL[3..0]` to ground. Pull `DCLK` either high or low, whichever is convenient on your board.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to `nCE` when it is used for active serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to GND or driven low during JTAG configuration. In multi-device FPP, AS, PS and PPA configuration chains, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating.

After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, make sure the `nCE` pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device will drive `nCE` of the next device low when it has successfully been JTAG configured.

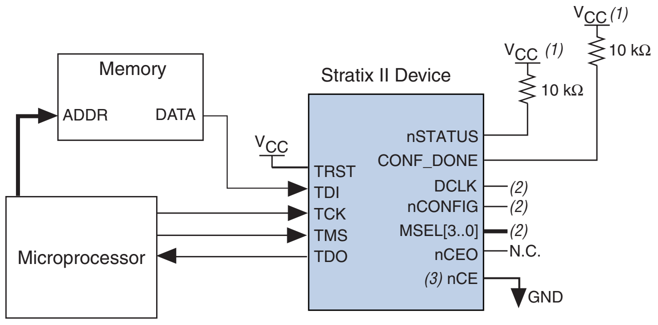
Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

Stratix, Stratix II, Cyclone, and Cyclone II devices must be within the first 17 devices in a JTAG chain. All of these devices have the same JTAG controller. If any of the Stratix, Stratix II, Cyclone, and Cyclone II devices are in the 18th or after they will fail configuration. This does not affect SignalTap II.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera device Chains* chapter in the *Configuration Handbook*.

Figure 2–37 shows JTAG configuration of a Stratix II device with a microprocessor.

Figure 2–37. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 2–37:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the device.

(2) The `nCONFIG`, `MSEL[3..0]` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL[3..0]` to ground. Pull `DCLK` either high or low, whichever is convenient on your board.

(3) `nCE` must be connected to GND or driven low for successful JTAG configuration.

Jam STAPL

Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, see *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the jam player, visit the Altera web site at **www.altera.com**

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the Stratix II device. Table 2–17 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 1 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
V_{CCPD}	N/A	All	Power	Dedicated power pin. This pin is used to power the I/O pre-drivers and a 3.3-V/2.5-V buffer available on the configuration input pins and JTAG pins. V_{CCPD} applies to all the JTAG input pins (TCK, TMS, TDI, and TRST) and the configuration pins; nCONFIG, DCLK (when used as an input), nIO_Pullup, DATA[7..0], RUnLU, nCE, nWS, nRS, CS, nCS and CLKUSR. This pin must be connected to 3.3-V. V_{CCPD} must ramp-up from 0-V to 3.3-V within 100 ms. If V_{CCPD} is not ramped up within this specified time, your Stratix II device will not configure successfully. If your system does not allow for a V_{CCPD} ramp-up time of 100 ms or less, you must hold nCONFIG low until all power supplies are stable.
VCCSEL	N/A	All	Input	Dedicated input that selects which input buffer is used on the configuration input pins; nCONFIG, DCLK (when used as an input), RUnLU, nCE, nWS, nRS, CS, nCS, and CLKUSR. The 3.3-V/2.5-V input buffer is powered by V_{CCPD}, while the 1.8-V/1.5-V input buffer is powered by V_{CCIO}. The VCCSEL input buffer has an internal 5-kΩ pull-down resistor that is always active. The VCCSEL input buffer is powered by V_{CCINT} and must be hardwired to V_{CCPD} or ground. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) selects the 1.8-V/1.5-V input buffer, and a logic low selects the 3.3-V/2.5-V input buffer. For more information, see "VCCSEL Pin" section.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 2 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`PORSEL`	N/A	All	Input	Dedicated input which selects between a POR time of 12 ms or 100 ms. A logic high (1.5 V, 1.8 V, 2.5 V, 3.3 V) selects a POR time of about 12 ms and a logic low selects POR time of about 100 ms. The `PORSEL` input buffer is powered by V_{CCINT} and has an internal 5-kΩ pull-down resistor that is always active. The `PORSEL` pin should be tied directly to V_{CCPD} or GND.
`nIO_PULLUP`	N/A	All	Input	Dedicated input that chooses whether the internal pull-up resistors on the user I/O pins and dual-purpose I/O pins (`nCSO`, `nASDO`, `DATA[7..0]`, `nWS`, `nRS`, `RDYnBSY`, `nCS`, `CS`, `RUnLU`, `PGM[]`, `CLKUSR`, `INIT_DONE`, `DEV_OE`, `DEV_CLR`) are on or off before and during configuration. A logic high (1.5 V, 1.8 V, 2.5 V, 3.3 V) turns off the weak internal pull-up resistors, while a logic low turns them on. The `nIO_PULLUP` pin has an internal 5-kΩ pull-down resistor that is always active.
`MSEL[3..0]`	N/A	All	Input	The `nIO-PULLUP` input buffer is powered by V_{CCPD} and has an internal 5-kΩ pull-down resistor that is always active. The `nIO-PULLUP` can be tied directly to V_{CCPD} or use a 1-kΩ pull-up resistor or tied directly to GND. The MSEL[3..0] pins have internal 5-kΩ pull-down resistors that are always active. 4-bit configuration input that sets the Stratix II device configuration scheme. See Table 2–1 for the appropriate connections. These pins must be hard-wired to V_{CCPD} or GND.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 3 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCONFIG	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode will cause the device to lose its configuration data, enter a reset state, tri-state all I/O pins. Returning this pin to a logic high level will initiate a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, nCONFIG can be tied directly to V_{CC} or to the configuration device's nINIT_CONF pin.
nSTATUS	N/A	All	Bidirectional open-drain	The device drives nSTATUS low immediately after power-up and releases it after the POR time. Status output. If an error occurs during configuration, nSTATUS is pulled low by the target device. Status input. If an external source drives the nSTATUS pin low during configuration or initialization, the target device enters an error state. Driving nSTATUS low after configuration and initialization does not affect the configured device. If a configuration device is used, driving nSTATUS low will cause the configuration device to attempt to configure the device, but since the device ignores transitions on nSTATUS in user-mode, the device does not reconfigure. To initiate a reconfiguration, nCONFIG must be pulled low. The enhanced configuration devices' and EPC2 devices' OE and nCS pins have optional internal programmable pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, external 10-kΩ pull-up resistors should not be used on these pins. When using EPC2 devices, only external 10-kΩ pull-up resistors should be used.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 4 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
CONF_DONE	N/A	All	Bidirectional open-drain	Status output. The target device drives the CONF_DONE pin low before and during configuration. Once all configuration data is received without error and the initialization cycle starts, the target device releases CONF_DONE. Status input. After all data is received and CONF_DONE goes high, the target device initializes and enters user mode. Driving CONF_DONE low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' OE and nCS pins have optional internal programmable pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, external 10-kΩ pull-up resistors should not be used on these pins. When using EPC2 devices, only external 10-kΩ pull-up resistors should be used.
nCE	N/A	All	Input	Active-low chip enable. The nCE pin activates the device with a low signal to allow configuration. The nCE pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, nCE of the first device is tied low while its nCEO pin is connected to nCE of the next device in the chain. The nCE pin must also be held low for successful JTAG programming of the device.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 5 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCEO	N/A	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's nCE pin. The nCEO of the last device in the chain is left floating. The nCEO pin is powered by V_{CCIO} in I/O bank 7. For recommendations on how to connect nCEO in a chain with multiple voltages across the devices in the chain, refer to the *MultiVolt I/O Interface* section in the *Stratix II Architecture* chapter in Volume 1 of the *Stratix II Handbook*.
ASDO	N/A in AS mode I/O in non-AS mode	AS	Output	Control signal from the Stratix II device to the serial configuration device in AS mode used to read out configuration data. In AS mode, ASDO has an internal pull-up resistor that is always active.
nCSO	N/A in AS mode I/O in non-AS mode	AS	Output	Output control signal from the Stratix II device to the serial configuration device in AS mode that enables the configuration device. In AS mode, nCSO has an internal pull-up resistor that is always active.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 6 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`DCLK`	N/A	Synchronous configuration schemes (PS, FPP, AS)	Input (PS, FPP) Output (AS)	In PS and FPP configuration, `DCLK` is the clock input used to clock data from an external source into the target device. Data is latched into the device on the rising edge of `DCLK`. In AS mode, `DCLK` is an output from the Stratix II device that provides timing for the configuration interface. In AS mode, `DCLK` has an internal pull-up resistor (typically 25 kΩ) that is always active. In PPA mode, `DCLK` should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, `DCLK` will be driven low after configuration is done. In schemes that use a control host, `DCLK` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
`DATA0`	I/O	PS, FPP, PPA, AS	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the `DATA0` pin. The V_{IH} and V_{IL} levels for this pin are dependent on the V_{CCIO} of the I/O bank that this pin resides in. In AS mode, `DATA0` has an internal pull-up resistor that is always active. After configuration, `DATA0` is available as a user I/O pin and the state of this pin depends on the **Dual-Purpose Pin** settings. After configuration, EPC1 and EPC1441 devices tri-state this pin, while enhanced configuration and EPC2 devices drive this pin high.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 7 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
DATA[7..1]	I/O	Parallel configuration schemes (FPP and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on DATA[7..0]. The V_{IH} and V_{IL} levels for this pin are dependent on the V_{CCIO} of the I/O bank that this pin resides in. In serial configuration schemes, they function as user I/O pins during configuration, which means they are tri-stated. After PPA or FPP configuration, DATA[7..1] are available as user I/O pins and the state of these pin depends on the **Dual-Purpose Pin** settings.
DATA7	I/O	PPA	Bidirectional	In the PPA configuration scheme, the DATA7 pin presents the RDYnBSY signal after the nRS signal has been strobed low. The V_{IH} and V_{IL} levels for this pin are dependent on the V_{CCIO} of the I/O bank that this pin resides in. In serial configuration schemes, it functions as a user I/O pin during configuration, which means it is tri-stated. After PPA configuration, DATA7 is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings.
nWS	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the DATA[7..0] pins. In non-PPA schemes, it functions as a user I/O pin during configuration, which means it is tri-stated. After PPA configuration, nWS is available as a user I/O pins and the state of this pin depends on the **Dual-Purpose Pin** settings.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 8 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nRS	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the RDYnBSY signal on the DATA7 pin. If the nRS pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nRS is available as a user I/O pin and the state of this pin depends on the **Dual-Purpose Pin** settings.
RDYnBSY	I/O	PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPA configuration schemes, this pin will drive out high after power-up, before configuration and after configuration before entering user-mode. In non-PPA schemes, it functions as a user I/O pin during configuration, which means it is tri-stated. After PPA configuration, RDYnBSY is available as a user I/O pin and the state of this pin depends on the **Dual-Purpose Pin** settings.

Table 2–17. Dedicated Configuration Pins on the Stratix II Device (Part 9 of 9)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nCS/CS`	I/O	PPA	Input	Chip-select inputs. A low on `nCS` and a high on `CS` select the target device for configuration. The `nCS` and `CS` pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the `nCS` or `CS` pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, `nCS` can be tied to ground while `CS` is toggled to control configuration. In non-PPA schemes, it functions as a user I/O pin during configuration, which means it is tri-stated. After PPA configuration, `nCS` and `CS` are available as user I/O pins and the state of these pins depends on the **Dual-Purpose Pin** settings.
`RUnLU`	N/A if using Remote System Upgrade I/O if not	Remote System Upgrade in FPP, PS or PPA	Input	Input that selects between remote update and local update. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) selects remote update and a logic low selects local update. When not using remote update or local update configuration modes, this pin is available as general-purpose user I/O pin. When using remote system upgrade in AS more, the `RUnLU` pin is available as a general-purpose I/O pin.
`PGM[2..0]`	N/A if using Remote System Upgrade I/O if not using	Remote System Upgrade in FPP, PS or PPA	Input	These output pins select one of eight pages in the memory (either flash or enhanced configuration device) when using a remote system upgrade mode. When not using remote update or local update configuration modes, these pins are available as general-purpose user I/O pins. When using remote system upgrade in AS more, the `PGM[]` pins are available as general-purpose I/O pins.

Table 2–18 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration, these pins function as user I/O pins and are tri-stated with weak pull-up resistors.

Table 2–18. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
`CLKUSR`	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input synchronizes the initialization of one or more devices. This pin is enabled by turning on the **Enable user-supplied start-up clock** (**CLKUSR**) option in the Quartus II software.
`INIT_DONE`	N/A if option is on. I/O if option is off.	Output open-drain	Status pin can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 10-kΩ pull-up resistor. 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 and the device enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the **Enable INIT_DONE output** option in the Quartus II software.
`DEV_OE`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the **Enable device-wide output enable** (**DEV_OE**) option in the Quartus II software.
`DEV_CLRn`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the **Enable device-wide reset** (**DEV_CLRn**) option in the Quartus II software.

Table 2–19 describes the dedicated JTAG pins. JTAG pins must be kept stable before and during configuration to prevent accidental loading of JTAG instructions. The `TDI`, `TMS,` and `TRST` have weak internal pull-up resistors (typically 25 kΩ) while `TCK` has a weak internal pull-down

resistor. If you plan to use the SignalTap® embedded logic array analyzer, you need to connect the JTAG pins of the Stratix II device to a JTAG header on your board.

Table 2–19. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
`TDI`	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. The `TDI` pin is powered by the 3.3-V V_{CCPD} supply. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TDO`	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. The `TDO` pin is powered by V_{CCIO} in I/O bank 4. For recommendations on connecting a JTAG chain with multiple voltages across the devices in the chain, refer to the *I/O Voltage Support in JTAG Chain* section in the *IEEE 1149.1 (JTAG) Boundary Scan Testing in Stratix II Devices* chapter in Volume 2 of the *Stratix II Handbook*. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, TMS must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. The `TMS` pin is powered by the 3.3-V V_{CCPD} supply. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TCK`	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. The `TCK` pin is powered by the 3.3-V V_{CCPD} supply. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting `TCK` to GND.
`TRST`	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. The `TRST` pin is powered by the 3.3-V V_{CCPD} supply. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting the `TRST` pin to GND.

Conclusion

Stratix II devices can be configured in a number of different schemes to fit your system's need. In addition, configuration bitstream encryption, configuration data decompression, and remote system upgrade support supplement the Stratix II configuration solution.

3. Configuring Stratix & Stratix GX Devices

S52013-3.1

Introduction

You can configure Stratix® and Stratix GX devices using one of several configuration schemes. All configuration schemes use either a microprocessor, configuration device, or a download cable. See Table 3–1.

Table 3–1. Stratix & Stratix GX Device Configuration Schemes

Configuration Scheme	Typical Use
Fast passive parallel (FPP)	Configuration with a parallel synchronous configuration device or microprocessor interface where eight bits of configuration data are loaded on every clock cycle.
Passive serial (PS)	Configuration with a serial synchronous microprocessor interface or the MasterBlaster™ communications cable, USB Blaster, ByteBlaster™ II, or ByteBlasterMV parallel port download cable.
Passive parallel asynchronous (PPA)	Configuration with a parallel asynchronous microprocessor interface. In this scheme, the microprocessor treats the target device as memory.
Remote/local update FPP	Configuration using a Nios™ (16-bit ISA) and Nios® II (32-bit ISA) or other embedded processor. Allows you to update the Stratix or Stratix GX device configuration remotely using the FPP scheme to load data.
Remote/local update PS	Passive serial synchronous configuration using a Nios or other embedded processor. Allows you to update the Stratix or Stratix GX device configuration remotely using the PS scheme to load data.
Remote/local update PPA	Passive parallel asynchronous configuration using a Nios or other embedded processor. In this scheme, the Nios microprocessor treats the target device as memory. Allows you to update the Stratix or Stratix GX device configuration remotely using the PPA scheme to load data.
Joint Test Action Group (JTAG)	Configuration through the IEEE Std. 1149.1 JTAG pins. You can perform JTAG configuration with either a download cable or an embedded device. Ability to use SignalTap® II Embedded Logic Analyzer.

This chapter discusses how to configure one or more Stratix or Stratix GX devices. It should be used together with the following documents:

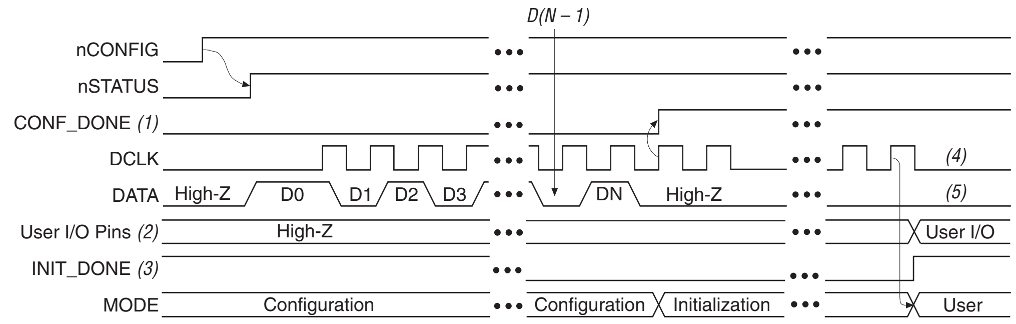
- *MasterBlaster Serial/USB Communications Cable Data Sheet*
- *USB Blaster USB Port Download Cable Development Tools Data Sheet*
- ByteBlaster II Parallel Port Download Cable Data Sheet
- *ByteBlasterMV Parallel Port Download Cable Data Sheets*
- *Configuration Devices for SRAM-Based LUT Devices Data Sheet*
- *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet*
- The *Remote System Configuration with Stratix & Stratix GX Devices* chapter

For more information on setting device configuration options or generating configuration files, see the *Software Setting* chapter in *Volume 2* of the *Configuration Handbook*.

Device Configuration Overview

During device operation, the FPGA stores configuration data in SRAM cells. Because SRAM memory is volatile, you must load the SRAM cells with the configuration data each time the device powers up. After configuration, the device must initialize its registers and I/O pins. After initialization, the device enters user mode. Figure 3–1 shows the state of the device during the configuration, initialization, and user mode.

Figure 3–1. Stratix & Stratix GX Configuration Cycle



Notes to Figure 3–1:

(1) During initial power up and configuration, `CONF_DONE` is low. After configuration, `CONF_DONE` goes high. If the device is reconfigured, `CONF_DONE` goes low after `nCONFIG` is driven low.

(2) User I/O pins are tri-stated during configuration. Stratix and Stratix GX devices also have a weak pull-up resistor on I/O pins during configuration that are enabled by `nIO_PULLUP`. After initialization, the user I/O pins perform the function assigned in the user's design.

(3) If the `INIT_DONE` pin is used, it will be high because of an external 10 kΩ resistor pull-up when `nCONFIG` is low and during the beginning of configuration. 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.

(4) `DCLK` should not be left floating. It should be driven high or low.

(5) `DATA0` should not be left floating. It should be driven high or low.

You can load the configuration data for the Stratix or Stratix GX device using a passive configuration scheme. When using any passive configuration scheme, the Stratix or Stratix GX device is incorporated into a system with an intelligent host, such as a microprocessor, that controls the configuration process. The host supplies configuration data from a storage device (e.g., a hard disk, RAM, or other system memory). When using passive configuration, you can change the target device's functionality while the system is in operation by reconfiguring the device. You can also perform in-field upgrades by distributing a new programming file to system users.

The following sections describe the `MSEL[2..0]`, `VCCSEL`, `PORSEL`, and `nIO_PULLUP` pins used in Stratix and Stratix GX device configuration.

MSEL[2..0] Pins

You can select a Stratix or Stratix GX device configuration scheme by driving its `MSEL2`, `MSEL1`, and `MSEL0` pins either high or low, as shown in Table 3–2.

Table 3–2. Stratix & Stratix GX Device Configuration Schemes

Description	MSEL2	MSEL1	MSEL0
FPP configuration	0	0	0
PPA configuration	0	0	1
PS configuration	0	1	0
Remote/local update FPP *(1)*	1	0	0
Remote/local update PPA *(1)*	1	0	1
Remote/local update PS *(1)*	1	1	0
JTAG-based configuration *(3)*	*(2)*	*(2)*	*(2)*

Notes to Table 3–2:

(1) These schemes require that you drive a secondary pin `RUnLU` to specify whether to perform a remote update or local update.

(2) Do not leave `MSEL` pins floating. Connect them to V_{CCIO} or GND. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used you should connect the `MSEL` pins to ground.

(3) JTAG-based configuration takes precedence over other configuration schemes, which means the `MSEL` pins are ignored.

The `MSEL[]` pins can be tied to V_{CCIO} of the I/O bank they reside in or ground.

V_{CCSEL} Pins

You can configure Stratix and Stratix GX devices using the 3.3-, 2.5-, 1.8-, or 1.5-V LVTTL I/O standard on configuration and JTAG input pins. `VCCSEL` is a dedicated input on Stratix and Stratix GX devices that selects between 3.3-V/2.5-V input buffers and 1.8-V/1.5-V input buffers for dedicated configuration input pins. A logic low supports 3.3-V/2.5-V signaling, and a logic high supports 1.8-V/1.5-V signaling. A logic high can also support 3.3-V/2.5-V signaling. `VCCSEL` affects the configuration related I/O banks (3, 4, 7, and 8) where the following pins reside: `TDI`, `TMS`, `TCK`, `TRST`, `MSEL0`, `MSEL1`, `MSEL2`, `nCONFIG`, `nCE`, `DCLK`, `PLL_ENA`, `CONF_DONE`, `nSTATUS`. The `VCCSEL` pin can be pulled to 1.5, 1.8, 2.5, or

3.3-V for a logic high level. There is an internal 2.5-kΩ pull-down resistor on VCCSEL. Therefore, if you are using a pull-up resister to pull up this signal, you need to use a 1-kΩ resistor.

VCCSEL also sets the power-on-reset (POR) trip point for all the configuration related I/O banks (3, 4, 7, and 8), ensuring that these I/O banks have powered up to the appropriate voltage levels before configuration begins. Upon power-up, the FPGA does not release nSTATUS until V_{CCINT} and all of the V_{CCIO}s of the configuration I/O banks are above their POR trip points. If you set VCCSEL to ground (logic low), this sets the POR trip point for all configuration I/O banks to a voltage consistent with 3.3-V/2.5-V signaling. When VCCSEL = 0, the POR trip point for these I/O banks may be as high as 1.8 V. If V_{CCIO} of any of the configuration banks is set to 1.8 or 1.5 V, the voltage supplied to this I/O bank(s) may never reach the POR trip point, which will not allow the FPGA to begin configuration.

If the V_{CCIO} of I/O banks 3, 4, 7, or 8 is set to 1.5 or 1.8 V and the configuration signals used require 3.3-V or 2.5-V signaling you should set VCCSEL to V_{CC} (logic high) in order to lower the POR trip point to enable successful configuration.

Table 3–3 shows how you should set the VCCSEL depending on the V_{CCIO} setting of the configuration I/O banks and your configuration input signaling voltages.

Table 3–3. VCCSEL Setting

V_{CCIO} (banks 3,4,7,8)	Configuration Input Signaling Voltage	V_{CCSEL}
3.3-V/2.5-V	3.3-V/2.5-V	GND
1.8-V/1.5-V	3.3-V/2.5-V/1.8-V/1.5-V	VCC
3.3-V/2.5-V	1.8-V/1.5-V	Not Supported

The VCCSEL signal does not control any of the dual-purpose pins, including the dual-purpose configuration pins, such as the DATA[7..0] and PPA pins (nWS, nRS, CS, nCS, and RDYnBSY). During configuration, these dual-purpose pins drive out voltage levels corresponding to the V_{CCIO} supply voltage that powers the I/O bank containing the pin. After configuration, the dual-purpose pins inherit the I/O standards specified in the design.

PORSEL Pins

`PORSEL` is a dedicated input pin used to select POR delay times of 2 ms or 100 ms during power-up. When the `PORSEL` pin is connected to ground, the POR time is 100 ms; when the `PORSEL` pin is connected to `VCC`, the POR time is 2 ms. There is an internal 2.5-kΩ pull-down resistor on `VCCSEL`. Therefore if you are using a pull-up resistor to pull up this signal, you need to use a 1-kΩ resistor.

When using enhanced configuration devices to configure Stratix devices, make sure that the `PORSEL` setting of the Stratix device is the same or faster than the `PORSEL` setting of the enhanced configuration device. If the FPGA is not powered up after the enhanced configuration device exits POR, the `CONF_DONE` signal will be high since the pull-up resistor is pulling this signal high. When the enhanced configuration device exits POR, OE of the enhanced configuration device is released and pulled high by a pull-up resistor. Since the enhanced configuration device sees its `nCS/CONF_DONE` signal also high, it enters a test mode. Therefore, you must ensure the FPGA powers up before the enhanced configuration device exits POR.

For more margin, the 100-ms setting can be selected when using an enhanced configuration device to allow the Stratix FPGA to power-up before configuration is attempted (see Table 3–4).

Table 3–4. PORSEL Settings

PORSEL Settings	POR Time (ms)
GND	100
V_{CC}	2

nIO_PULLUP Pins

The `nIO_PULLUP` pin enables a built-in weak pull-up resistor to pull all user I/O pins to VCCIO before and during device configuration. If `nIO_PULLUP` is connected to V_{CC} during configuration, the weak pull-ups on all user I/O pins and all dual-purpose pins are disabled. If connected to ground, the pull-ups are enabled during configuration. The `nIO_PULLUP` pin can be pulled to 1.5, 1.8, 2.5, or 3.3-V for a logic level high. There is an internal 2.5-kΩ pull-down resistor on `VCCSEL`. Therefore, if you are using a pull-up resistor to pull up this signal, you need to use a 1-kΩ resistor.

TDO & nCEO Pins

TDO and nCEO pins drive out the same voltage levels as the V_{CCIO} that powers the I/O bank where the pin resides. You must select the V_{CCIO} supply for the bank containing TDO accordingly. For example, when using the ByteBlasterMV cable, the V_{CCIO} for the bank containing TDO must be powered up at 3.3-V. The current strength for TDO is 12 mA.

Configuration File Size

Tables 3–5 and 3–6 summarize the approximate configuration file size required for each Stratix and Stratix GX device. To calculate the amount of storage space required for multi-device configurations, add the file size of each device together.

Table 3–5. Stratix Configuration File Sizes

Device	Raw Binary File (.rbf) Size (Bits)
EP1S10	3,534,640
EP1S20	5,904,832
EP1S25	7,894,144
EP1S30	10,379,368
EP1S40	12,389,632
EP1S60	17,543,968
EP1S80	23,834,032

Table 3–6. Stratix GX Configuration File Sizes

Device	Raw Binary File Size (Bits)
EP1SGX10C	3,579,928
EP1SGX10D	3,579,928
EP1SGX25C	7,951,248
EP1SGX25D	7,951,248
EP1SGX25F	7,951,248
EP1SGX40D	12,531,440
EP1SGX40G	12,531,440

You should only use the numbers in Tables 3–5 and 3–6 to estimate the file size before design compilation. The exact file size may vary because different Altera® Quartus® II software versions may add a slightly different number of padding bits during programming. However, for any specific version of the Quartus II software, any design targeted for the same device has the same configuration file size.

Altera Configuration Devices

The Altera enhanced configuration devices (EPC16, EPC8, and EPC4 devices) support a single-device configuration solution for high-density FPGAs and can be used in the FPP and PS configuration schemes. They are ISP-capable through its JTAG interface. The enhanced configuration devices are divided into two major blocks, the controller and the flash memory.

For information on enhanced configuration devices, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* and the *Using Altera Enhanced Configuration Devices* chapter in the *Configuration Handbook*.

The EPC2 and EPC1 configuration devices provide configuration support for the PS configuration scheme. The EPC2 device is ISP-capable through its JTAG interface. The EPC2 and EPC1 can be cascaded to hold large configuration files.

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

Configuration Schemes

This section describes how to configure Stratix and Stratix GX devices with the following configuration schemes:

- PS Configuration with Configuration Devices
- PS Configuration with a Download Cable
- PS Configuration with a Microprocessor
- FPP Configuration
- PPA Configuration
- JTAG Programming & Configuration
- JTAG Programming & Configuration of Multiple Devices

PS Configuration

PS configuration of Stratix and Stratix GX devices can be performed using an intelligent host, such as a MAX® device, microprocessor with flash memory, an Altera configuration device, or a download cable. In the PS scheme, an external host (MAX device, embedded processor, configuration device, or host PC) controls configuration. Configuration data is clocked into the target Stratix devices via the `DATA0` pin at each rising edge of `DCLK`.

PS Configuration with Configuration Devices

The configuration device scheme uses an Altera configuration device to supply data to the Stratix or Stratix GX device in a serial bitstream (see Figure 3–3).

In the configuration device scheme, `nCONFIG` is usually tied to V_{CC} (when using EPC16, EPC8, EPC4, or EPC2 devices, `nCONFIG` may be connected to `nINIT_CONF`). Upon device power-up, the target Stratix or Stratix GX device senses the low-to-high transition on `nCONFIG` and initiates configuration. The target device then drives the open-drain `CONF_DONE` pin low, which in-turn drives the configuration device's `nCS` pin low. When exiting power-on reset (POR), both the target and configuration device release the open-drain `nSTATUS` pin.

Before configuration begins, the configuration device goes through a POR delay of up to 200 ms to allow the power supply to stabilize (power the Stratix or Stratix GX device before or during the POR time of the configuration device). This POR delay has a maximum of 200 ms for EPC2 devices. For enhanced configuration devices, you can select between 2 ms and 100 ms by connecting `PORSEL` pin to `VCC` or `GND`, accordingly. During this time, the configuration device drives its `OE` pin low. This low signal delays configuration because the `OE` pin is connected to the target device's `nSTATUS` pin. When the target and configuration devices complete POR, they release `nSTATUS`, which is then pulled high by a pull-up resistor.

When configuring multiple devices, configuration does not begin until all devices release their `OE` or `nSTATUS` pins. When all devices are ready, the configuration device clocks data out serially to the target devices using an internal oscillator.

After successful configuration, the Stratix FPGA starts initialization using the 10-MHz internal oscillator as the reference clock. After initialization, this internal oscillator is turned off. The `CONF_DONE` pin is released by the target device and then pulled high by a pull-up resistor. When initialization is complete, the FPGA enters user mode.

If an error occurs during configuration, the target device drives its `nSTATUS` pin low, resetting itself internally and resetting the configuration device. If the **Auto-Restart Configuration on Frame Error** option—available in the Quartus II **Global Device Options** dialog box (Assign menu)—is turned on, the device reconfigures automatically if an error occurs. To find this option, choose **Compiler Settings** (Processing menu), then click on the **Chips & Devices** tab.

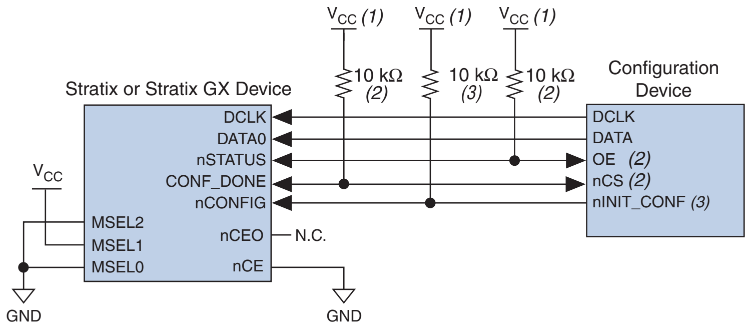
If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}. When configuration is complete, the target device releases CONF_DONE, which disables the configuration device by driving nCS high. The configuration device drives DCLK low before and after configuration.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the target device has not configured successfully. In this case, the configuration device pulses its OE pin low for a few microseconds, driving the target device's nSTATUS pin low. If the **Auto-Restart Configuration on Frame Error** option is set in the software, the target device resets and then pulses its nSTATUS pin low. When nSTATUS returns high, the configuration device reconfigures the target device. When configuration is complete, the configuration device drives DCLK low.

Do not pull CONF_DONE low to delay initialization. Instead, use the Quartus II software's **Enable User-Supplied Start-Up Clock (CLKUSR)** option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain initialize together. When CONF_DONE is driven low after device configuration, the configuration device recognizes that the target device has not configured successfully.

Figure 3–2 shows how to configure one Stratix or Stratix GX device with one configuration device.

Figure 3–2. Single Device Configuration Circuit



Notes to Figure 3–2:

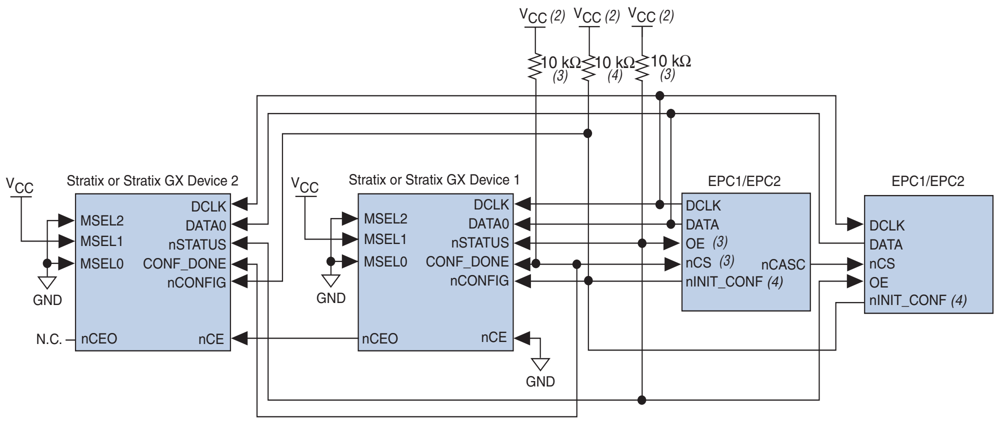
(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The enhanced configuration devices and EPC2 devices have internal programmable pull-ups on OE and nCS. You should only use the internal pull-ups of the configuration device if the nSTATUS and CONF_DONE signals are pulled up to 3.3 V or 2.5 V (not 1.8 V or 1.5 V). If external pull-ups are used, they should be 10 kΩ.

(3) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor. he nINIT_CONF pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the nINIT_CONF pin.

Figure 3–3 shows how to configure multiple Stratix and Stratix GX devices with multiple EPC2 or EPC1 configuration devices.

Figure 3–3. Multi-Device Configuration Circuit *Note (1)*



Notes to Figure 3–3:

(1) When performing multi-device active serial configuration, you must generate the configuration device programmer object file (**.pof**) from each project's SOF. You can combine multiple SOFs using the Quartus II software through the **Device & Pin Option** dialog box. For more information on how to create configuration and programming files, see the *Software Settings* section in the *Configuration Handbook, Volume 2*.

(2) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(3) The enhanced configuration devices and EPC2 devices have internal programmable pull-ups on OE and nCS. You should only use the internal pull-ups of the configuration device if the nSTATUS and CONF_DONE signals are pulled up to 3.3 V or 2.5 V (not 1.8 V or 1.5 V). If external pull-ups are used, they should be 10 kΩ

(4) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor. The nINIT_CONF pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the nINIT_CONF pin.

After the first Stratix or Stratix GX device completes configuration during multi-device configuration, its nCEO pin activates the second device's nCE pin, prompting the second device to begin configuration. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

In addition, all nSTATUS pins are tied together; thus, if any device (including the configuration devices) detects an error, configuration stops for the entire chain. Also, if the first configuration device does not detect CONF_DONE going high at the end of configuration, it resets the chain by pulsing its OE pin low for a few microseconds. This low pulse drives the OE pin low on the second configuration device and drives nSTATUS low on all Stratix and Stratix GX devices, causing them to enter an error state.

If the **Auto-Restart Configuration on Frame Error** option is turned on in the software, the Stratix or Stratix GX device releases its nSTATUS pins after a reset time-out period. When the nSTATUS pins are released and pulled high, the configuration devices reconfigure the chain. If the **Auto-Restart Configuration on Frame Error** option is not turned on, the Stratix or Stratix GX devices drive nSTATUS low until they are reset with a low pulse on nCONFIG.

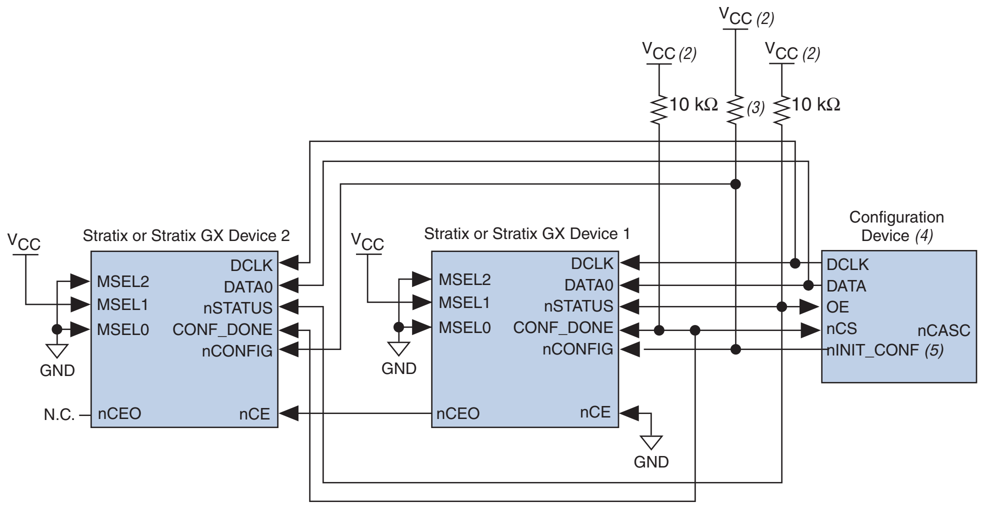
You can also cascade several EPC2/EPC1 configuration devices to configure multiple Stratix and Stratix GX devices. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the subsequent configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

☞ You cannot cascade enhanced (EPC16, EPC8, and EPC4) configuration devices.

You can use a single configuration chain to configure multiple Stratix and Stratix GX devices. In this scheme, the nCEO pin of the first device is connected to the nCE pin of the second device in the chain. If there are additional devices, connect the nCE pin of the next device to the nCEO pin of the previous device. To configure properly, all of the device CONF_DONE and nSTATUS pins must be tied together.

Figure 3–4 shows an example of configuring multiple Stratix and Stratix GX devices using a configuration device.

***Figure 3–4. Configuring Multiple Stratix & Stratix GX Devices with A Single Configuration Device** Note (1)*



Notes to Figure 3–4:

(1) When performing multi-device active serial configuration, you must generate the configuration device programmer object file (**.pof)** from each project's SOF. You can combine multiple SOFs using the Quartus II software through the **Device & Pin Option** dialog box. For more information on how to create configuration and programming files, see the *Software Settings section* in the *Configuration Handbook, Volume 2*.

(2) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(3) The enhanced configuration devices and EPC2 devices have internal programmable pull-ups on OE and nCS. You should only use the internal pull-ups of the configuration device if the nSTATUS and CONF_DONE signals are pulled up to 3.3 V or 2.5 V (not 1.8 V or 1.5 V). If external pull-ups are used, they should be 10 kΩ.

(4) EPC16, EPC8, and EPC4 configuration devices cannot be cascaded.

(5) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor. The nINIT_CONF pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the nINIT_CONF pin.

Table 3–7 shows the status of the device DATA pins during and after configuration.

Table 3–7. DATA Pin Status Before & After Configuration

Pins	Stratix or Stratix GX Device	
	During	After
`DATA0` *(1)*	Used for configuration	User defined
`DATA[7..1]` *(2)*	Used in some configuration modes	User defined
I/O Pins	Tri-state	User defined

Notes to Table 3–7:

(1) The status shown is for configuration with a configuration device.

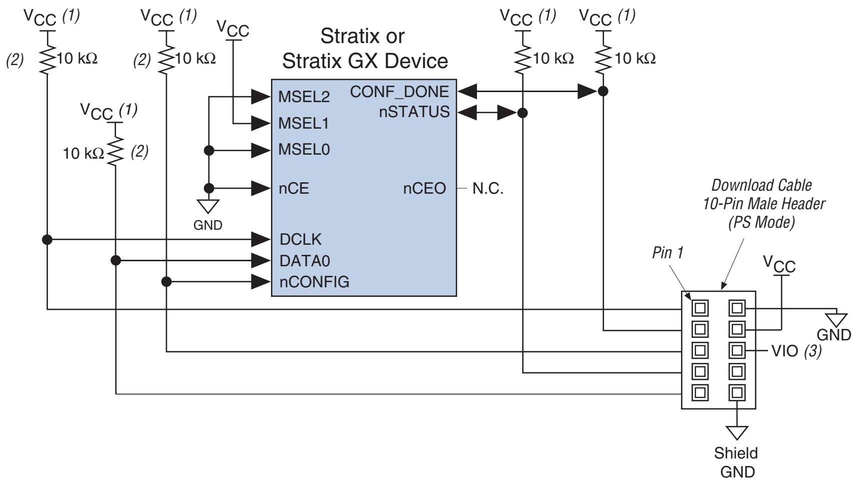
(2) The function of these pins depends upon the settings specified in the Quartus II software using the **Device & Pin Option** dialog box (see the *Software Settings section* in the *Configuration Handbook, Volume 2,* and the Quartus II Help software for more information).

PS Configuration with a Download Cable

In PS configuration with a download cable, an intelligent host transfers data from a storage device to the Stratix or Stratix GX device through the MasterBlaster, USB-Blaster, ByteBlaster II or ByteBlasterMV cable. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin. The programming hardware then places the configuration data one bit at a time on the device's `DATA0` pin. The data is clocked into the target device until `CONF_DONE` goes high.

When using programming hardware for the Stratix or Stratix GX device, turning on the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle because the Quartus II software must restart configuration when an error occurs. Additionally, the **Enable User-Supplied Start-Up Clock (CLKUSR)** option has no affect on the device initialization since this option is disabled in the SOF when programming the FPGA using the Quartus II software programmer and a download cable. Therefore, if you turn on the **CLKUSR** option, you do not need to provide a clock on `CLKUSR` when you are configuring the FPGA with the Quartus II programmer and a download cable. Figure 3–5 shows PS configuration for the Stratix or Stratix GX device using a MasterBlaster, USB-Blaster, ByteBLaster II or ByteBlasterMV cable.

Figure 3–5. PS Configuration Circuit with a Download Cable



Notes to Figure 3–5:

(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (`VIO` pin) or ByteBlasterMV cable.

(2) The pull-up resistors on the `DATA0` and `DCLK` pins are only needed if the download cable is the only configuration scheme used on the board. This is to ensure that the `DATA0` and `DCLK` pins are not left floating after configuration. For example, if the design also uses a configuration device, the pull-up resistors on the `DATA0` and `DCLK` pins are not necessary.

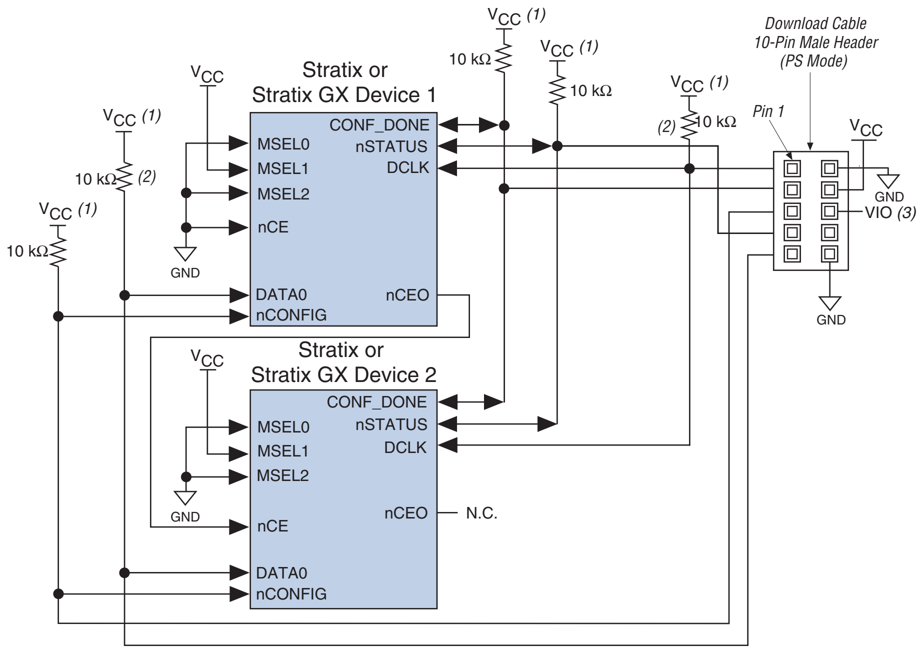
(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. This pin is a no-connect pin for the ByteBlasterMV header.

You can use programming hardware to configure multiple Stratix and Stratix GX devices by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. All other configuration pins are connected to each device in the chain.

Because all `CONF_DONE` pins are tied together, all devices in the chain initialize and enter user mode at the same time. In addition, because the `nSTATUS` pins are tied together, the entire chain halts configuration if any device detects an error. In this situation, the Quartus II software must restart configuration; the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle.

Figure 3–6 shows how to configure multiple Stratix and Stratix GX devices with a MasterBlaster or ByteBlasterMV cable.

Figure 3–6. Multi-Device PS Configuration with a Download Cable



Notes to Figure 3–6:

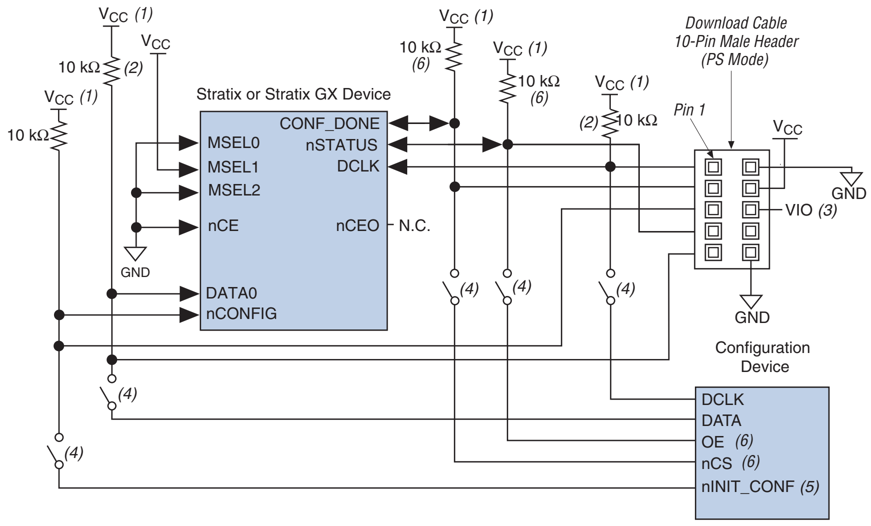
(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (`VIO` pin) or ByteBlasterMV cable.

(2) The pull-up resistors on the `DATA0` and `DCLK` pins are only needed if the download cable is the only configuration scheme used on the board. This is to ensure that the `DATA0` and `DCLK` pins are not left floating after configuration. For example, if the design also uses a configuration device, the pull-up resistors on the `DATA0` and `DCLK` pins are not necessary.

(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. See the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

If you are using a download cable to configure device(s) on a board that also has configuration devices, you should electrically isolate the configuration devices from the target device(s) and cable. One way to isolate the configuration devices is to add logic, such as a multiplexer, that can select between the configuration devices and the cable. The multiplexer device should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`CONF_DONE`, `nSTATUS`, `DCLK`, `nCONFIG`, and `DATA0`) between the cable and the configuration devices. The last option is to remove the configuration devices from the board when configuring with the cable. Figure 3–7 shows a combination of a configuration device and a download cable to configure a Stratix or Stratix GX device.

Figure 3–7. Configuring with a Combined PS & Configuration Device Scheme



Notes to Figure 3–7:

(1) You should connect the pull-up resistor to the same supply voltage as the configuration device.

(2) The pull-up resistors on the `DATA0` and `DCLK` pins are only needed if the download cable is the only configuration scheme used on the board. This is to ensure that the `DATA0` and `DCLK` pins are not left floating after configuration. For example, if the design also uses a configuration device, the pull-up resistors on the `DATA0` and `DCLK` pins are not necessary.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the target device's V_{CCIO}. This is a no-connect pin for the ByteBlasterMV header.

(4) You should not attempt configuration with a download cable while a configuration device is connected to a Stratix or Stratix GX device. Instead, 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. Remove the download cable when configuring with a configuration device.

(5) If `nINIT_CONF` is not used, `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(6) If external pull-ups are used on `CONF_DONE` and `nSTATUS` pins, they should always be 10 kΩ resistors. You can use the internal pull-ups of the configuration device only if the `CONF_DONE` and `nSTATUS` signals are pulled-up to 3.3 V or 2.5 V (not 1.8 V or 1.5 V).

For more information on how to use the MasterBlaster or ByteBlasterMV cables, see the following documents:

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

PS Configuration with a Microprocessor

In PS configuration with a microprocessor, a microprocessor transfers data from a storage device to the target Stratix or Stratix GX device. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin and the target device must release `nSTATUS`. The microprocessor or programming hardware then places the configuration data one bit at a time on the `DATA0` pin of the Stratix or Stratix GX device. The least significant bit (LSB) of each data byte must be presented first. Data is clocked continuously into the target device until `CONF_DONE` goes high.

After all configuration data is sent to the Stratix and Stratix GX device, the `CONF_DONE` pin will go high to show successful configuration and the start of initialization. Initialization, by default, uses an internal oscillator, which runs at 10 MHz. After initialization, this internal oscillator is turned off. If you are using the **clkusr** option, after all data is transferred `clkusr` must be clocked an additional 136 times for the Stratix or Stratix GX device to initialize properly. Driving `DCLK` to the device after configuration is complete does not affect device operation.

Handshaking signals are not used in PS configuration modes. Therefore, the configuration clock speed must be below the specified frequency to ensure correct configuration. No maximum `DCLK` period exists. You can pause configuration by halting `DCLK` for an indefinite amount of time.

If the target device detects an error during configuration, it drives its `nSTATUS` pin low to alert the microprocessor. The microprocessor can then pulse `nCONFIG` low to restart the configuration process. Alternatively, if the **Auto-Restart Configuration on Frame Error** option is turned on in the Quartus II software, the target device releases `nSTATUS` after a reset time-out period. After `nSTATUS` is released, the microprocessor can reconfigure the target device without needing to pulse `nCONFIG` low.

The microprocessor can also monitor the `CONF_DONE` and `INIT_DONE` pins to ensure successful configuration. If the microprocessor sends all data and the initialization clock starts but `CONF_DONE` and `INIT_DONE` have not gone high, it must reconfigure the target device. By default the `INIT_DONE` output is disabled. You can enable the `INIT_DONE` output by turning on **Enable INIT_DONE output** option in the Quartus II software.

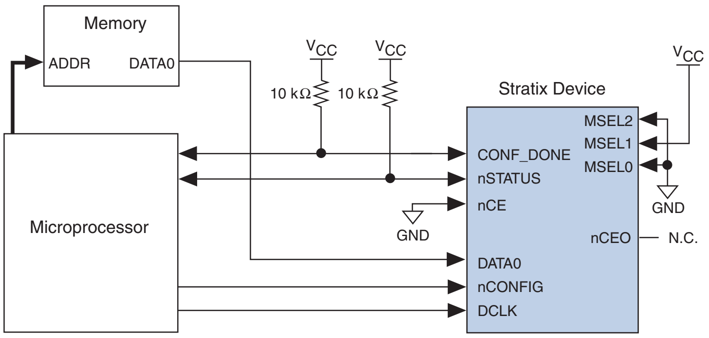
If you do not turn on the **Enable INIT_DONE output** option in the Quartus II software, you are advised to wait for the maximum value of t_{CD2UM} (see Table 3–8) after the `CONF_DONE` signal goes high to ensure the device has been initialized properly and that it has entered user mode.

During configuration and initialization, and before the device enters user mode, the microprocessor must not drive the CONF_DONE signal low.

If the optional CLKUSR pin is used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

Figure 3–8 shows the circuit for PS configuration with a microprocessor.

Figure 3–8. PS Configuration Circuit with Microprocessor



PS Configuration Timing

Figure 3–9 shows the PS configuration timing waveform for Stratix and Stratix GX devices. Table 3–8 shows the PS timing parameters for Stratix and Stratix GX devices.

Table 3–8. PS Timing Parameters for Stratix & Stratix GX Devices

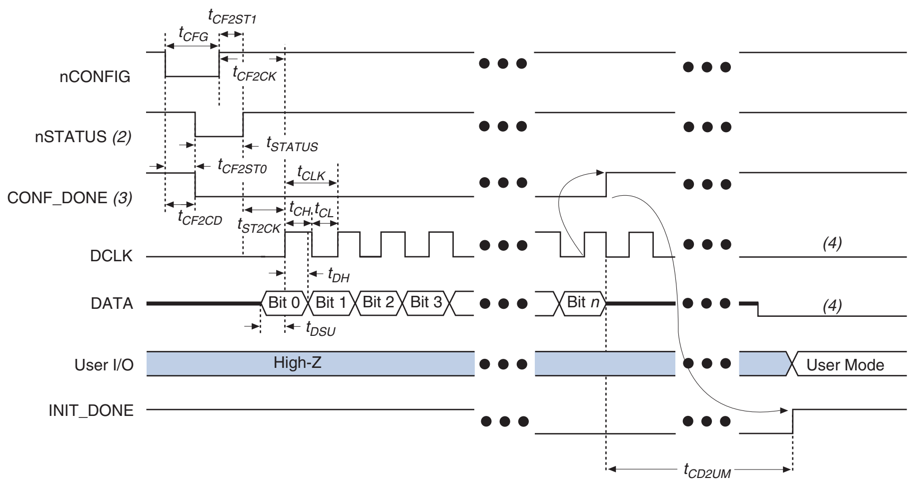
Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		800	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		800	ns
t_{CF2ST1}	nCONFIG high to nSTATUS high		40 *(2)*	µs
t_{CFG}	nCONFIG low pulse width	40		µs
t_{STATUS}	nSTATUS low pulse width	10	40 *(2)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	40		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	7		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK maximum frequency		100	MHz
t_{CD2UM}	CONF_DONE high to user mode *(1)*	6	20	µs

Notes to Table 3–8:

(1) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 136 to obtain this value.

(2) This value is obtainable if users do not delay configuration by extending the nSTATUS low pulse width.

***Figure 3–9. PS Timing Waveform for Stratix & Stratix GX Devices** Note (1)*



***Notes to** Figure 3–9:*

(1) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS`, and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.
(2) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.
(3) Upon power-up, before and during configuration, `CONF_DONE` is low.
(4) `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is convenient. `DATA[]` is available as user I/Os after configuration and the state of these pins depends on the dual-purpose pin settings.

FPP Configuration

Parallel configuration of Stratix and Stratix GX devices meets the continuously increasing demand for faster configuration times. Stratix and Stratix GX devices can receive byte-wide configuration data per clock cycle, and guarantee a configuration time of less than 100 ms with a 100-MHz configuration clock. Stratix and Stratix GX devices support programming data bandwidth up to 800 megabits per second (Mbps) in this mode. You can use parallel configuration with an EPC16, EPC8, or EPC4 device, or a microprocessor.

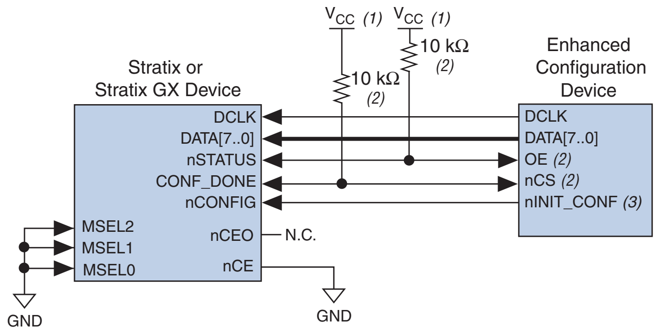
This section discusses the following schemes for FPP configuration in Stratix and Stratix GX devices:

- FPP Configuration Using an Enhanced Configuration Device
- FPP Configuration Using a Microprocessor

FPP Configuration Using an Enhanced Configuration Device

When using FPP with an enhanced configuration device, it supplies data in a byte-wide fashion to the Stratix or Stratix GX device every DCLK cycle. See Figure 3–10.

Figure 3–10. FPP Configuration Using Enhanced Configuration Devices



Notes to Figure 3–10:

(1) The pull-up resistors should be connected to the same supply voltage as the configuration device.

(2) The enhanced configuration devices and EPC2 devices have internal programmable pull-ups on OE and nCS. You should only use the internal pull-ups of the configuration device if the nSTATUS and CONF_DONE signals are pulled up to 3.3 V or 2.5 V (not 1.8 V or 1.5 V). If external pull-ups are used, they should be 10 kΩ.

(3) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor. The nINIT_CONF pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the nINIT_CONF pin.

In the enhanced configuration device scheme, nCONFIG is tied to nINIT_CONF. On power up, the target Stratix or Stratix GX device senses the low-to-high transition on nCONFIG and initiates configuration. The target Stratix or Stratix GX device then drives the open-drain CONF_DONE pin low, which in-turn drives the enhanced configuration device's nCS pin low.

Before configuration starts, there is a 2-ms POR delay if the PORSEL pin is connected to V_{CC} in the enhanced configuration device. If the PORSEL pin is connected to ground, the POR delay is 100 ms. When each device determines that its power is stable, it releases its nSTATUS or OE pin. Because the enhanced configuration device's OE pin is connected to the target Stratix or Stratix GX device's nSTATUS pin, configuration is delayed until both the nSTATUS and OE pins are released by each device. The nSTATUS and OE pins are pulled up by a resistor on their respective devices once they are released. When configuring multiple devices, connect the nSTATUS pins together to ensure configuration only happens when all devices release their OE or nSTATUS pins. The enhanced configuration device then clocks data out in parallel to the Stratix or Stratix GX device using a 66-MHz internal oscillator, or drives it to the Stratix or Stratix GX device through the EXTCLK pin.

If there is an error during configuration, the Stratix or Stratix GX device drives the nSTATUS pin low, resetting itself internally and resetting the enhanced configuration device. The Quartus II software provides an **Auto-restart configuration after error** option that automatically initiates the reconfiguration whenever an error occurs. See the *Software Settings* chapter in Volume 2 of the *Configuration Handbook* for information on how to turn this option on or off.

If this option is turned off, you must set monitor nSTATUS to check for errors. To initiate reconfiguration, pulse nCONFIG low. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}. Therefore, nCONFIG must be connected to nINIT_CONF if you want to reprogram the Stratix or Stratix GX device on the fly.

When configuration is complete, the Stratix or Stratix GX device releases the CONF_DONE pin, which is then pulled up by a resistor. This action disables the EPC16, EPC8, or EPC4 enhanced configuration device as nCS is driven high. Initialization, by default, uses an internal oscillator, which runs at 10 MHz. After initialization, this internal oscillator is turned off. When initialization is complete, the Stratix or Stratix GX device enters user mode. The enhanced configuration device drives DCLK low before and after configuration.

If, after sending out all of its data, the enhanced configuration device does not detect CONF_DONE going high, it recognizes that the Stratix or Stratix GX device has not configured successfully. The enhanced configuration device pulses its OE pin low for a few microseconds, driving the nSTATUS pin on the Stratix or Stratix GX device low. If the **Auto-restart configuration after error** option is on, the Stratix or Stratix GX device resets and then pulses its nSTATUS low. When nSTATUS returns high, reconfiguration is restarted (see Figure 3–11 on page 3–24).

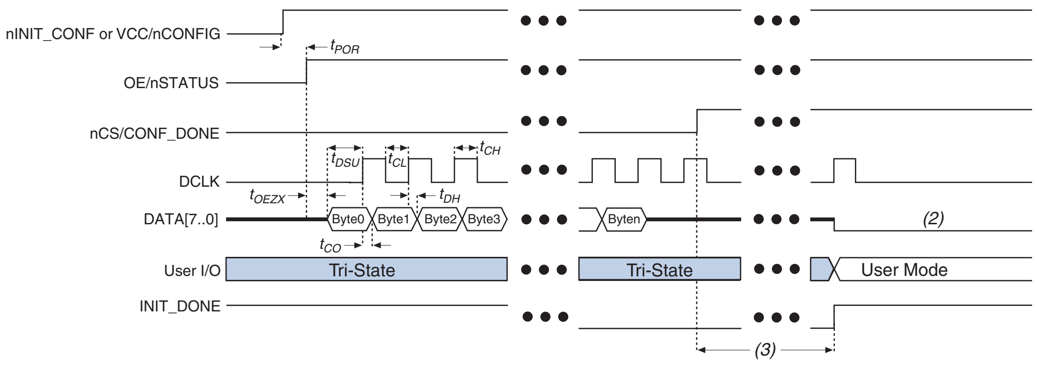
Do not drive CONF_DONE low after device configuration to delay initialization. Instead, use the **Enable User-Supplied Start-Up Clock (CLKUSR)** option in the **Device & Pin Options** dialog box. You can use this option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain initialize together.

After the first Stratix or Stratix GX device completes configuration during multi-device configuration, its `nCEO` pin activates the second Stratix or Stratix GX device's `nCE` pin, prompting the second device to begin configuration. Because `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time. Because `nSTATUS` pins are tied together, configuration stops for the whole chain if any device (including enhanced configuration devices) detects an error. Also, if the enhanced configuration device does not detect a high on `CONF_DONE` at the end of configuration, it pulses its `OE` low for a few microseconds to reset the chain. The low `OE` pulse drives `nSTATUS` low on all Stratix and Stratix GX devices, causing them to enter an error state. This state is similar to a Stratix or Stratix GX device detecting an error.

If the **Auto-restart configuration after error** option is on, the Stratix and Stratix GX devices release their `nSTATUS` pins after a reset time-out period. When the `nSTATUS` pins are released and pulled high, the configuration device reconfigures the chain. If the **Auto-restart configuration after error** option is off, `nSTATUS` stays low until the Stratix and Stratix GX devices are reset with a low pulse on `nCONFIG`.

Figure 3–11 shows the FPP configuration with a configuration device timing waveform for Stratix and Stratix GX devices.

***Figure 3–11. FPP Configuration with a Configuration Device Timing Waveform** Note (1)*



Notes to Figure 3–11:

(1) For timing information, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*.

(2) The configuration device drives `DATA` high after configuration.

(3) Stratix and Stratix GX devices enter user mode 136 clock cycles after `CONF_DONE` goes high.

FPP Configuration Using a Microprocessor

When using a microprocessor for parallel configuration, the microprocessor transfers data from a storage device to the Stratix or Stratix GX device through configuration hardware. To initiate configuration, the microprocessor needs to generate a low-to-high transition on the `nCONFIG` pin and the Stratix or Stratix GX device must release `nSTATUS`. The microprocessor then places the configuration data to the `DATA[7..0]` pins of the Stratix or Stratix GX device. Data is clocked continuously into the Stratix or Stratix GX device until `CONF_DONE` goes high.

The configuration clock (`DCLK`) speed must be below the specified frequency to ensure correct configuration. No maximum `DCLK` period exists. You can pause configuration by halting `DCLK` for an indefinite amount of time.

After all configuration data is sent to the Stratix or Stratix GX device, the `CONF_DONE` pin will go high to show successful configuration and the start of initialization. Initialization, by default, uses an internal oscillator, which runs at 10 MHz. After initialization, this internal oscillator is turned off. If you are using the **clkusr** option, after all data is transferred `clkusr` must be clocked an additional 136 times for the Stratix or Stratix GX device to initialize properly. Driving `DCLK` to the device after configuration is complete does not affect device operation. By default, the `INIT_DONE` output is disabled. You can enable the `INIT_DONE` output by turning on the **Enable INIT_DONE output** option in the Quartus II software.

If you do not turn on the **Enable INIT_DONE output** option in the Quartus II software, you are advised to wait for maximum value of t_{CD2UM} (see Table 3–9) after the `CONF_DONE` signal goes high to ensure the device has been initialized properly and that it has entered user mode.

During configuration and initialization and before the device enters user mode, the microprocessor must not drive the `CONF_DONE` signal low.

☞ If the optional `CLKUSR` pin is used and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 µs).

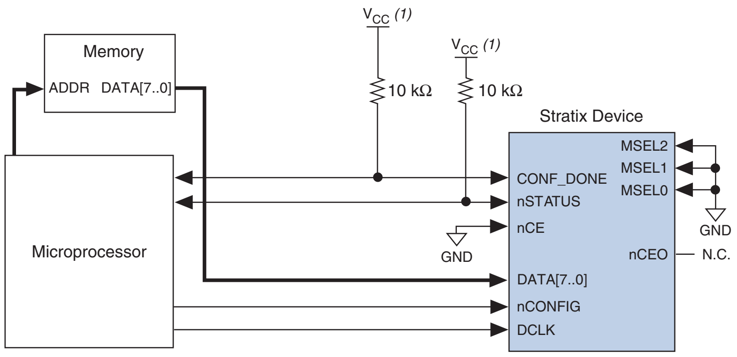
If the Stratix or Stratix GX device detects an error during configuration, it drives `nSTATUS` low to alert the microprocessor. The pin on the microprocessor connected to `nSTATUS` must be an input. The microprocessor can then pulse `nCONFIG` low to restart the configuration error. With the **Auto-restart configuration after error** option on, the

Stratix or Stratix GX device releases nSTATUS after a reset time-out period. After nSTATUS is released, the microprocessor can reconfigure the Stratix or Stratix GX device without pulsing nCONFIG low.

The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. If the microprocessor sends all the data and the initialization clock starts but CONF_DONE and INIT_DONE have not gone high, it must reconfigure the Stratix or Stratix GX device. After waiting the specified 136 DCLK cycles, the microprocessor should restart configuration by pulsing nCONFIG low.

Figure 3–12 shows the circuit for Stratix and Stratix GX parallel configuration using a microprocessor.

Figure 3–12. Parallel Configuration Using a Microprocessor



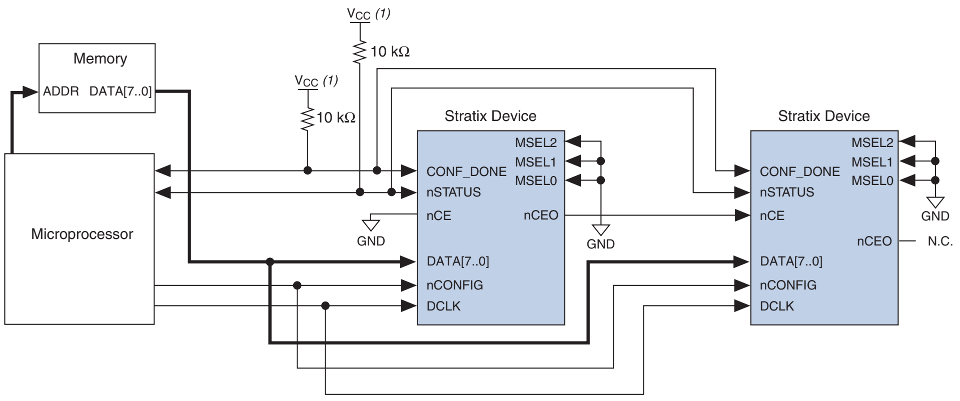
Note to Figure 3–12:

(1) The pull-up resistors should be connected to any V_{CC} that meets the Stratix high-level input voltage (V_{IH}) specification.

For multi-device parallel configuration with a microprocessor, the nCEO pin of the first Stratix or Stratix GX device is cascaded to the second device's nCE pin. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. Because the CONF_DONE pins of the devices are connected together, all devices initialize and enter user mode at the same time.

Because the nSTATUS pins are also tied together, if any of the devices detects an error, the entire chain halts configuration and drives nSTATUS low. The microprocessor can then pulse nCONFIG low to restart configuration. If the **Auto-restart configuration after error** option is on, the Stratix and Stratix GX devices release nSTATUS after a reset time-out period. The microprocessor can then reconfigure the devices once nSTATUS is released. Figure 3–13 shows multi-device configuration using a microprocessor. Figure 3–14 shows multi-device configuration when both Stratix and Stratix GX devices are receiving the same data. In this case, the microprocessor sends the data to both devices simultaneously, and the devices configure simultaneously.

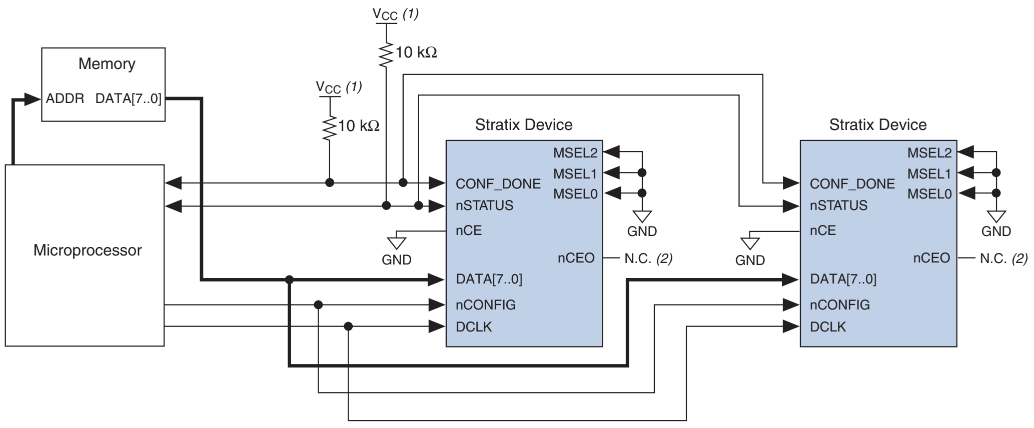
Figure 3–13. Parallel Data Transfer in Serial Configuration with a Microprocessor



Note to Figure 3–13:

(1) You should connect the pull-up resistors to any V_{CC} that meets the Stratix high-level input voltage (V_{IH}) specification.

Figure 3–14. Multiple Device Parallel Configuration with the Same Data Using a Microprocessor



Notes to Figure 3–14:

(1) You should connect the pull-up resistors to any V_{CC} that meets the Stratix high-level input voltage (V_{IH}) specification.

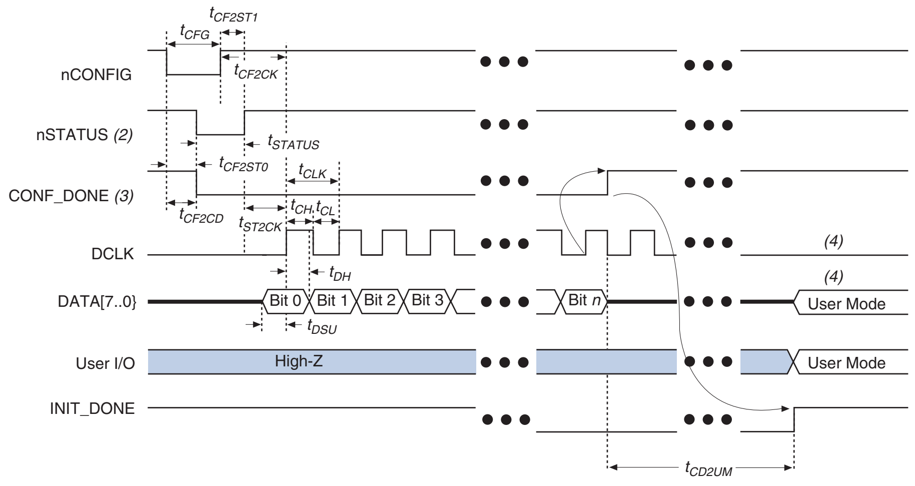
(2) The nCEO pins are left unconnected when configuring the same data into multiple Stratix or Stratix GX devices.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera FPGA Chains* chapter in the *Configuration Handbook, Volume 2*.

FPP Configuration Timing

Figure 3–15 shows FPP timing waveforms for configuring a Stratix or Stratix GX device in FPP mode. Table 3–9 shows the FPP timing parameters for Stratix or Stratix GX devices.

***Figure 3–15. Timing Waveform for Configuring Devices in FPP Mode** Note (1)*



Notes to Figure 3–15:

(1) The beginning of this waveform shows the device in user-mode. In user-mode, `nCONFIG`, `nSTATUS`, and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.
(2) Upon power-up, the Stratix II device holds `nSTATUS` low for the time of the POR delay.
(3) Upon power-up, before and during configuration, `CONF_DONE` is low.
(4) `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is convenient. `DATA[]` is available as user I/Os after configuration and the state of these pins depends on the dual-purpose pin settings.

Table 3–9. FPP Timing Parameters for Stratix & Stratix GX Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CK}	`nCONFIG` high to first rising edge on DCLK	40		µs
t_{DSU}	Data setup time before rising edge on DCLK	7		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CFG}	`nCONFIG` low pulse width	40		µs
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK frequency		100	MHz
t_{CD2UM}	CONF_DONE high to user mode (1)	6	20	µs
t_{CF2CD}	`nCONFIG` low to CONF_DONE low		800	ns
t_{CF2ST0}	`nCONFIG` low to nSTATUS low		800	ns

Table 3–9. FPP Timing Parameters for Stratix & Stratix GX Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2ST1}	nCONFIG high to nSTATUS high		40 *(2)*	µs
t_{STATUS}	nSTATUS low pulse width	10	40 *(2)*	µs
t_{ST2CK}	nSTATUS high to firstrising edge of DCLK	1		µs

Notes to Table 3–9:
(1) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 136 to obtain this value.
(2) This value is obtainable if users do not delay configuration by extending the nSTATUS low pulse width.

PPA Configuration

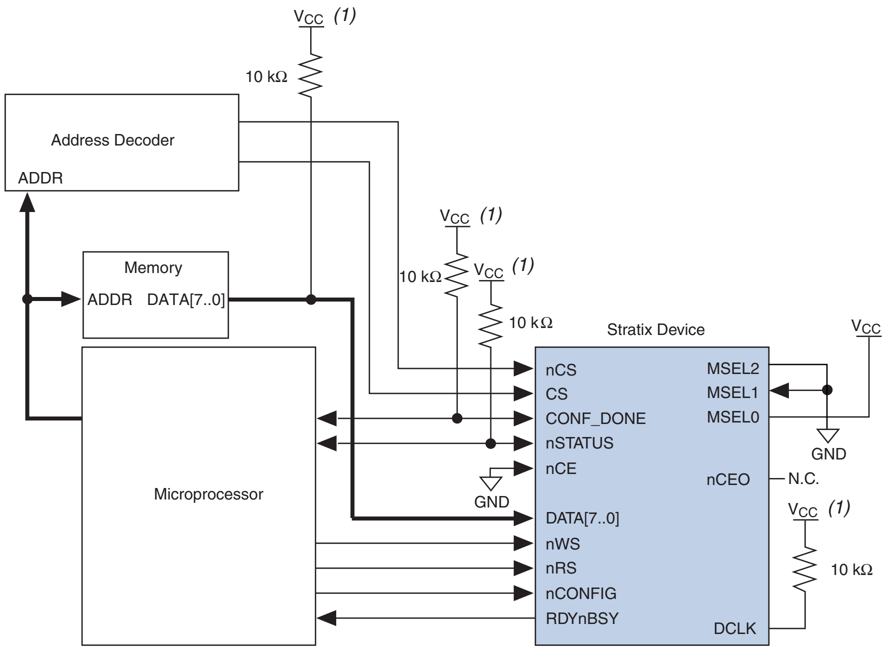
In PPA schemes, a microprocessor drives data to the Stratix or Stratix GX device through a download cable. When using a PPA scheme, use a 1-kΩ pull-up resistor to pull the DCLK pin high to prevent unused configuration pins from floating.

To begin configuration, the microprocessor drives nCONFIG high and then asserts the target device's nCS pin low and CS pin high. Next, the microprocessor places an 8-bit configuration word on the target device's data inputs and pulses nWS low. On the rising edge of nWS, the target device latches a byte of configuration data and then drives its RDYnBSY signal low, indicating that it is processing the byte of configuration data. The microprocessor then performs other system functions while the Stratix or Stratix GX device is processing the byte of configuration data.

Next, the microprocessor checks nSTATUS and CONF_DONE. If nSTATUS is high and CONF_DONE is low, the microprocessor sends the next data byte. If nSTATUS is low, the device is signaling an error and the microprocessor should restart configuration. However, if nSTATUS is high and all the configuration data is received, the device is ready for initialization. At the beginning of initialization, CONF_DONE goes high to indicate that configuration is complete. Initialization, by default, uses an internal oscillator, which runs at 10 MHz. After initialization, this internal oscillator is turned off. When initialization is complete, the Stratix or Stratix GX device enters user mode.

Figure 3–16 shows the PPA configuration circuit. An optional address decoder controls the device's nCS and CS pins. This decoder allows the microprocessor to select the Stratix or Stratix GX device by accessing a particular address, simplifying the configuration process.

Figure 3–16. PPA Configuration Circuit



Note to Figure 3–16:

(1) The pull-up resistor should be connected to the same supply voltage as the Stratix or Stratix GX device.

The device's nCS or CS pins can be toggled during PPA configuration if the design meets the specifications for t_{CSSU}, t_{WSP}, and t_{CSH} given in Table 3–10 on page 3–36. The microprocessor can also directly control the nCS and CS signals. You can tie one of the nCS or CS signals to its active state (i.e., nCS may be tied low) and toggle the other signal to control configuration.

Stratix and Stratix GX devices can serialize data internally without the microprocessor. When the Stratix or Stratix GX device is ready for the next byte of configuration data, it drives RDYnBSY high. If the microprocessor senses a high signal when it polls RDYnBSY, the microprocessor strobes the next byte of configuration data into the device. Alternatively, the nRS signal can be strobed, causing the RDYnBSY signal to appear on DATA7. Because RDYnBSY does not need to be monitored, reading the state of the configuration data by strobing nRS low saves a system I/O port. Do not drive data onto the data bus while nRS is low because it causes contention on DATA7. If the nRS pin is not used to monitor configuration, you should tie it high. To simplify configuration, the microprocessor can wait for the total time of t_{BUSY} (max) + t_{RDY2WS} + t_{W2SB} before sending the next data bit.

After configuration, the nCS, CS, nRS, nWS, and RDYnBSY pins act as user I/O pins. However, if the PPA scheme is chosen in the Quartus II software, these I/O pins are tri-stated by default in user mode and should be driven by the microprocessor. To change the default settings in the Quartus II software, select **Device & Pin Option** (Compiler Setting menu).

If the Stratix or Stratix GX device detects an error during configuration, it drives nSTATUS low to alert the microprocessor. The microprocessor can then pulse nCONFIG low to restart the configuration process. Alternatively, if the **Auto-Restart Configuration on Frame Error** option is turned on, the Stratix or Stratix GX device releases nSTATUS after a reset time-out period. After nSTATUS is released, the microprocessor can reconfigure the Stratix or Stratix GX device. At this point, the microprocessor does not need to pulse nCONFIG low.

The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The microprocessor must monitor the CONF_DONE pin to detect errors and determine when programming completes. If the microprocessor sends all configuration data and starts initialization but CONF_DONE is not asserted, the microprocessor must reconfigure the Stratix or Stratix GX device.

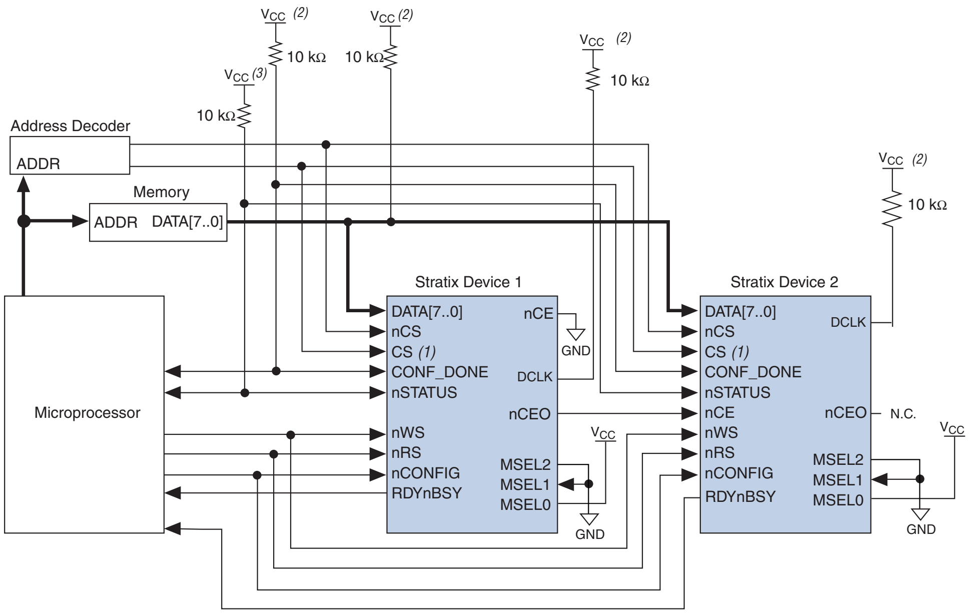
By default, the INIT_DONE is disabled. You can enable the INIT_DONE output by turning on the **Enable INIT_DONE output** option in the Quartus II software. If you do not turn on the **Enable INIT_DONE output** option in the Quartus II software, you are advised to wait for the maximum value of t_{CD2UM} (see Table 3–10) after the CONF_DONE signal goes high to ensure the device has been initialized properly and that it has entered user mode.

During configuration and initialization, and before the device enters user mode, the microprocessor must not drive the CONF_DONE signal low.

If the optional CLKUSR pin is used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

You can also use PPA mode to configure multiple Stratix and Stratix GX devices. Multi-device PPA configuration is similar to single-device PPA configuration, except that the Stratix and Stratix GX devices are cascaded. After you configure the first Stratix or Stratix GX device, nCEO is asserted, which asserts the nCE pin on the second device, initiating configuration. Because the second Stratix or Stratix GX device begins configuration within one write cycle of the first device, the transfer of data destinations is transparent to the microprocessor. All Stratix and Stratix GX device CONF_DONE pins are tied together; therefore, all devices initialize and enter user mode at the same time. See Figure 3–17.

Figure 3–17. PPA Multi-Device Configuration Circuit



Notes to Figure 3–17:

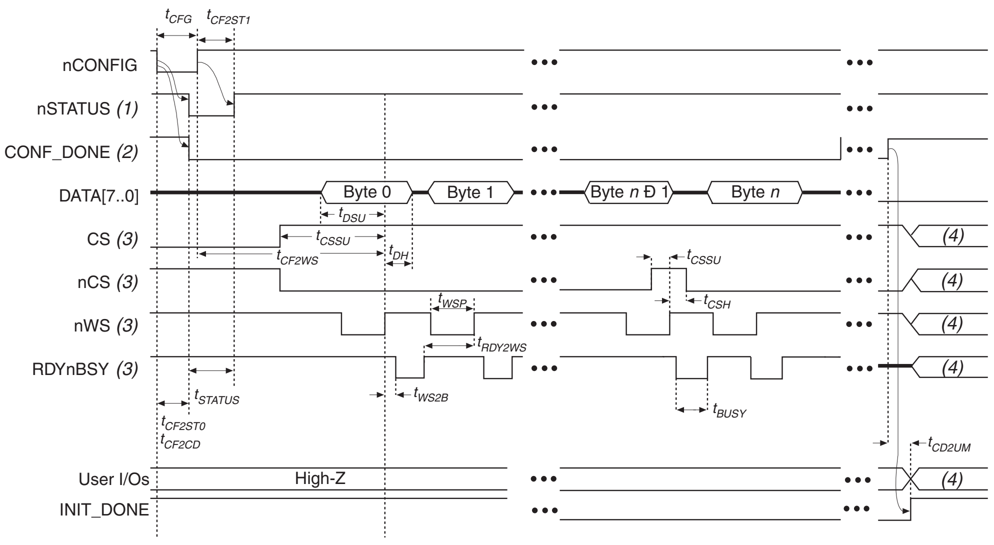
(1) If not used, you can connect the CS pin to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) Connect the pull-up resistor to the same supply voltage as the Stratix or Stratix GX device.

PPA Configuration Timing

Figure 3–18 shows the Stratix and Stratix GX device timing waveforms for PPA configuration.

Figure 3–18. PPA Timing Waveforms for Stratix & Stratix GX Devices



Notes to Figure 3–18:

(1) Upon power-up, `nSTATUS` is held low for the time of the POR delay.

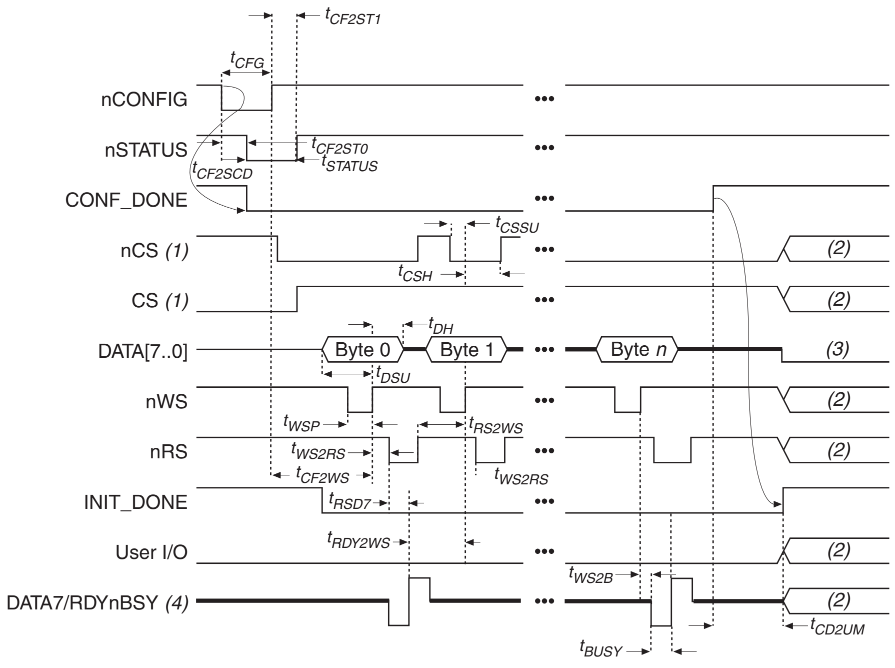
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) After configuration, the state of `CS`, `nCS`, `nWS`, and `RDYnBSY` depends on the design programmed into the Stratix or Stratix GX device.

(4) Device I/O pins are in user mode.

Figure 3–19 shows the Stratix and Stratix GX timing waveforms when using strobed nRS and nWS signals.

Figure 3–19. PPA Timing Waveforms Using Strobed nRS & nWS Signals



Notes to Figure 3–19:

(1) The user can toggle nCS or CS during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(2) Device I/O pins are in user mode.

(3) The DATA[7..0] pins are available as user I/Os after configuration and the state of theses pins depends on the dual-purpose pin settings. Do not leave DATA[7..0] floating. If these pins are not used in user-mode, you should drive them high or low, whichever is more convenient.

(4) DATA7 is a bidirectional pin. It represents an input for data input, but represents an output to show the status of RDYnBSY.

Table 3–10 defines the Stratix and Stratix GX timing parameters for PPA configuration

Table 3–10. PPA Timing Parameters for Stratix & Stratix GX Devices

Symbol	Parameter	Min	Max	Units
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	15		ns
t_{CFG}	`nCONFIG` low pulse width	40		µs
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		20	ns
t_{BUSY}	`RDYnBSY` low pulse width	7	45	ns
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	15		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	15		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	15		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		20	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(1)*	6	20	µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(2)*	µs
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		800	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		800	ns
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		40 *(2)*	µs

Notes to Table 3–10:

(1) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

(2) This value is obtained if you do not delay configuration by extending the `nstatus` to low pulse width.

For information on how to create configuration and programming files for this configuration scheme, see the *Software Settings* section in the *Configuration Handbook, Volume 2*.

JTAG Programming & Configuration

The JTAG has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on printed circuit boards (PCBs) with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. You can also use the JTAG circuitry to shift configuration data into the device.

For more information on JTAG boundary-scan testing, see *AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

To use the SignalTap® II embedded logic analyzer, you need to connect the JTAG pins of your Stratix device to a download cable header on your PCB.

For more information on SignalTap II, see the *Design Debugging Using SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook, Volume 2*.

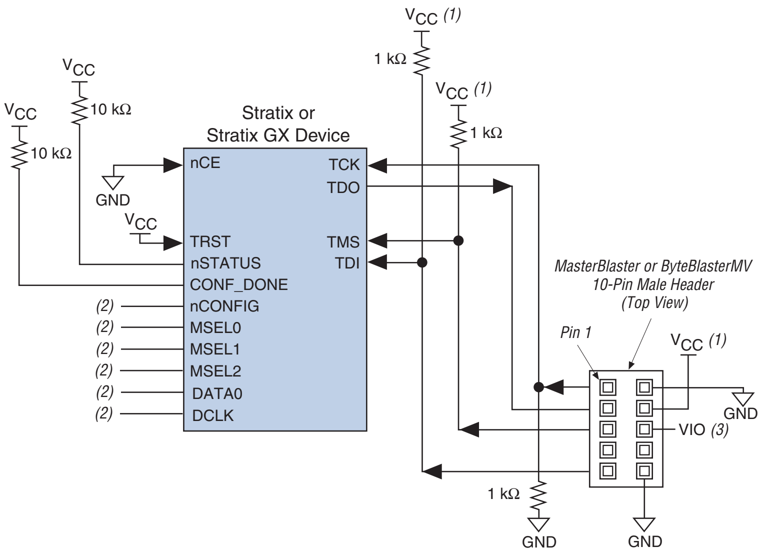
A device operating in JTAG mode uses four required pins, `TDI`, `TDO`, `TMS`, and `TCK`, and one optional pin, `TRST`. The four JTAG input pins (`TDI`, `TMS`, `TCK` and `TRST`) have weak, internal pull-up resistors, whose values range from 20 to 40 kΩ. All other pins are tri-stated during JTAG configuration. Do not begin JTAG configuration until all other configuration is complete. Table 3–11 shows each JTAG pin's function.

Table 3–11. JTAG Pin Descriptions

Pin	Description	Function
`TDI`	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. The `VCCSEL` pin controls the input buffer selection.
`TDO`	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. The high level output voltage is determined by VCCIO.
`TMS`	Test mode select	Input pin that provides the control signal to determine the transitions of the Test Access Port (TAP) controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. The `VCCSEL` pin controls the input buffer selection.
`TCK`	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. The `VCCSEL` pin controls the input buffer selection.
`TRST`	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. The `VCCSEL` pin controls the input buffer selection.

During JTAG configuration, data is downloaded to the device on the PCB through the MasterBlaster or ByteBlasterMV header. Configuring devices through a cable is similar to programming devices in-system. One difference is to connect the TRST pin to V_{CC} to ensure that the TAP controller is not reset. See Figure 3–20.

Figure 3–20. JTAG Configuration of a Single Device



Notes to Figure 3–20:

(1) You should connect the pull-up resistor to the same supply voltage as the download cable.

(2) You should connect the `nCONFIG`, `MSEL0`, and `MSEL1` pins to support a non-JTAG configuration scheme. If you only use JTAG configuration, connect `nCONFIG` to V_{CC}, and `MSEL0`, `MSEL1`, and `MSEL2` to ground. Pull `DATA0` and `DCLK` to high or low.

(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. See the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

To configure a single device in a JTAG chain, the programming software places all other devices in BYPASS mode. In BYPASS mode, devices pass programming data from the `TDI` pin to the `TDO` pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the `TDO` pin one clock cycle later.

Stratix and Stratix GX devices have dedicated JTAG pins. You can perform JTAG testing on Stratix and Stratix GX devices before and after, but not during configuration. The chip-wide reset and output enable pins on Stratix and Stratix GX devices do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of Stratix and Stratix GX devices, you should consider the regular configuration pins. Table 3–12 shows how you should connect these pins during JTAG configuration.

Table 3–12. Dedicated Configuration Pin Connections During JTAG Configuration

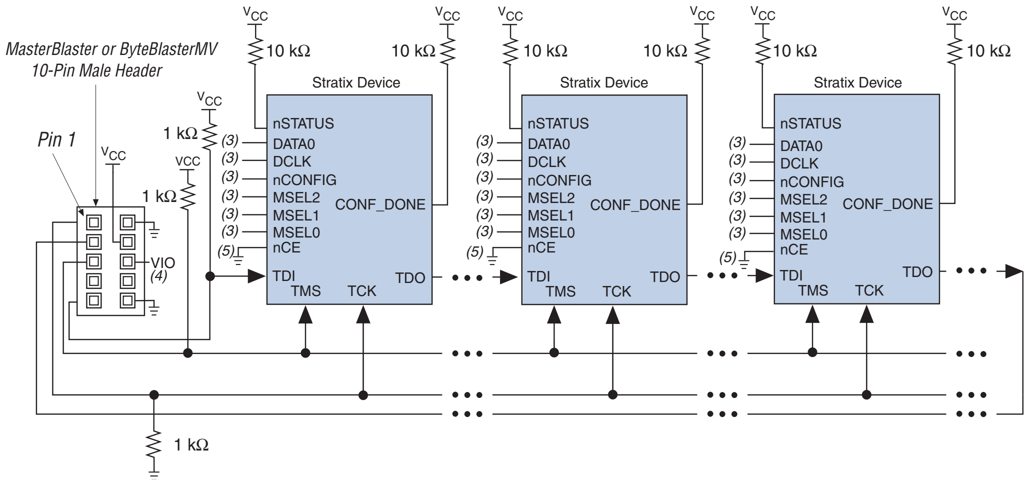
Signal	Description
nCE	On all Stratix and Stratix GX devices in the chain, nCE should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, FPP or PPA configuration chains, the nCE pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all Stratix and Stratix GX devices in the chain, nCEO can be left floating or connected to the nCE of the next device. See nCE pin description above.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie both pins to ground.
nCONFIG	nCONFIG must be driven high through the JTAG programming process. Driven high by connecting to V_{CC}, pulling high via a resistor, or driven by some control circuitry.
nSTATUS	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each nSTATUS pin should be pulled up to V_{CC} individually. nSTATUS pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each CONF_DONE pin should be pulled up to V_{CC} individually. CONF_DONE going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.
DATA0	Should not be left floating. Drive low or high, whichever is more convenient on your board.

JTAG Programming & Configuration of Multiple Devices

When programming a JTAG device chain, one JTAG-compatible header, such as the ByteBlasterMV header, is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capacity of the download cable. However, when more than five devices are connected in a JTAG chain, Altera recommends buffering the TCK, TDI, and TMS pins with an on-board buffer.

JTAG-chain device programming is ideal when the PCB contains multiple devices, or when testing the PCB using JTAG BST circuitry. Figure 3–21 shows multi-device JTAG configuration.

Figure 3–21. Multi-Device JTAG Configuration *Notes (1), (2)*



Notes to Figure 3–21:

(1) Stratix, Stratix GX, APEX™ II, APEX 20K, Mercury™, ACEX® 1K, and FLEX® 10K devices can be placed within the same JTAG chain for device programming and configuration.

(2) For more information on all configuration pins connected in this mode, see Table 3–11 on page 3–37.

(3) Connect the `nCONFIG`, `MSEL0`, `MSEL1`, and `MSEL2` pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0`, `MSEL1`, and `MSEL2` to ground. Pull `DATA0` and `DCLK` to either high or low.

(4) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. See the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

(5) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to `GND` or driven low during JTAG configuration. In multi-device PS, FPP and PPA configuration chains, the first device's `nCE` pin is connected to `GND` while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to `GND` during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device drives `nCE` of the next device low when it has successfully been JTAG configured.

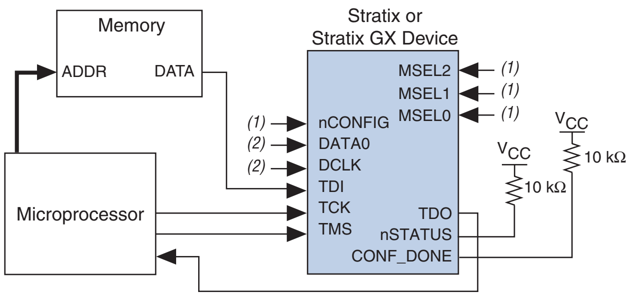
The Quartus II software verifies successful JTAG configuration upon completion. The software checks the state of `CONF_DONE` through the JTAG port. If `CONF_DONE` is not in the correct state, the Quartus II software indicates that configuration has failed. If `CONF_DONE` is in the correct state, the software indicates that configuration was successful.

If `VCCIO` is tied to 3.3 V, both the I/O pins and JTAG `TDO` port drive at 3.3-V levels.

Do not attempt JTAG and non-JTAG configuration simultaneously. When configuring through JTAG, allow any non-JTAG configuration to complete first.

Figure 3–22 shows the JTAG configuration of a Stratix or Stratix GX device with a microprocessor.

Figure 3–22. JTAG Configuration of Stratix & Stratix GX Devices with a Microprocessor



Notes to Figure 3–22:

(1) Connect the `nCONFIG`, `MSEL2`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL2`, `MSEL1`, and `MSEL0` pins to ground.

(2) Pull `DATA0` and `DCLK` to either high or low.

Configuration with JRunner Software Driver

JRunner is a software driver that allows you to configure Altera FPGAs through the ByteBlasterMV download cable in JTAG mode. The programming input file supported is in Raw Binary File (**.rbf**) format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system. You can customize the code to make it run on other platforms.

For more information on the JRunner software driver, see the *JRunner Software Driver: An Embedded Solution to the JTAG Configuration White Paper* and zip file.

Jam STAPL Programming & Test Language

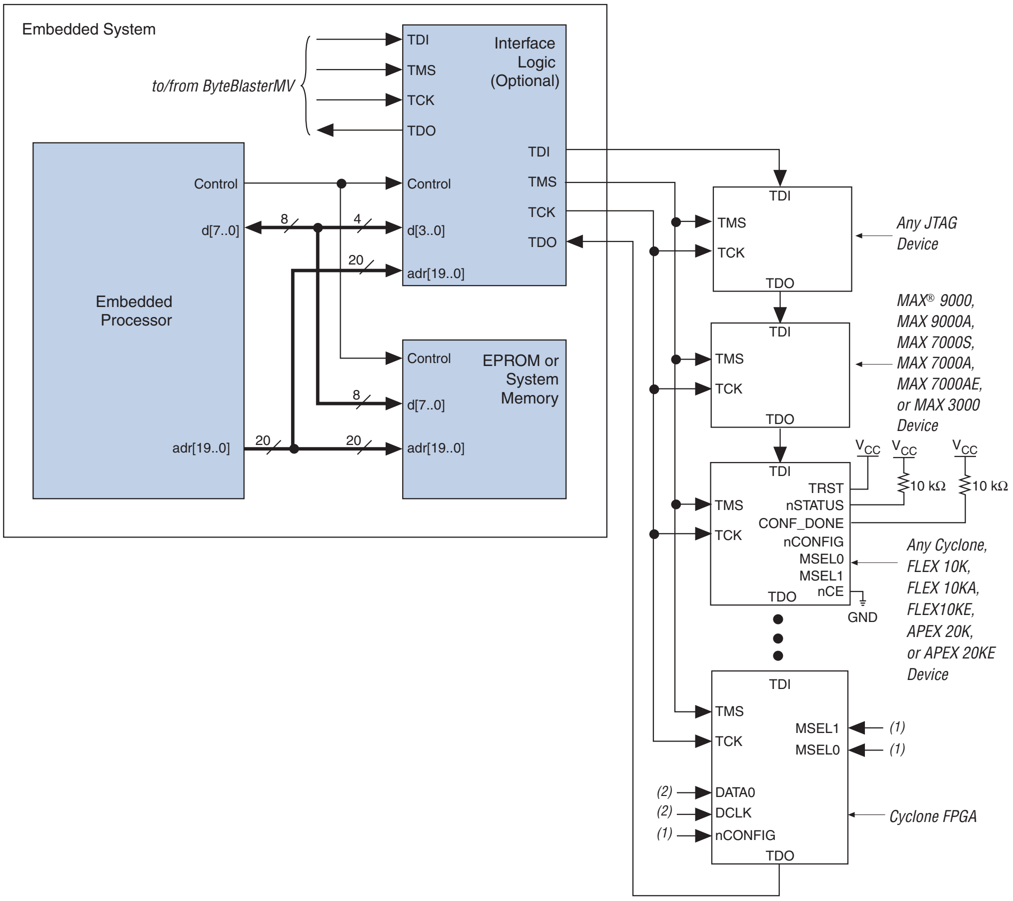
The Jam™ Standard Test and Programming Language (STAPL), JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

Connecting the JTAG Chain to the Embedded Processor

There are two ways to connect the JTAG chain to the embedded processor. The most straightforward method is to connect the embedded processor directly to the JTAG chain. In this method, four of the processor pins are dedicated to the JTAG interface, saving board space but reducing the number of available embedded processor pins.

Figure 3–23 illustrates the second method, which is to connect the JTAG chain to an existing bus through an interface PLD. In this method, the JTAG chain becomes an address on the existing bus. The processor then reads from or writes to the address representing the JTAG chain.

Figure 3–23. Embedded System Block Diagram



Notes to Figure 3–23:

(1) Connect the `nCONFIG`, `MSEL2`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL2`, `MSEL1`, and `MSEL0` pins to ground.

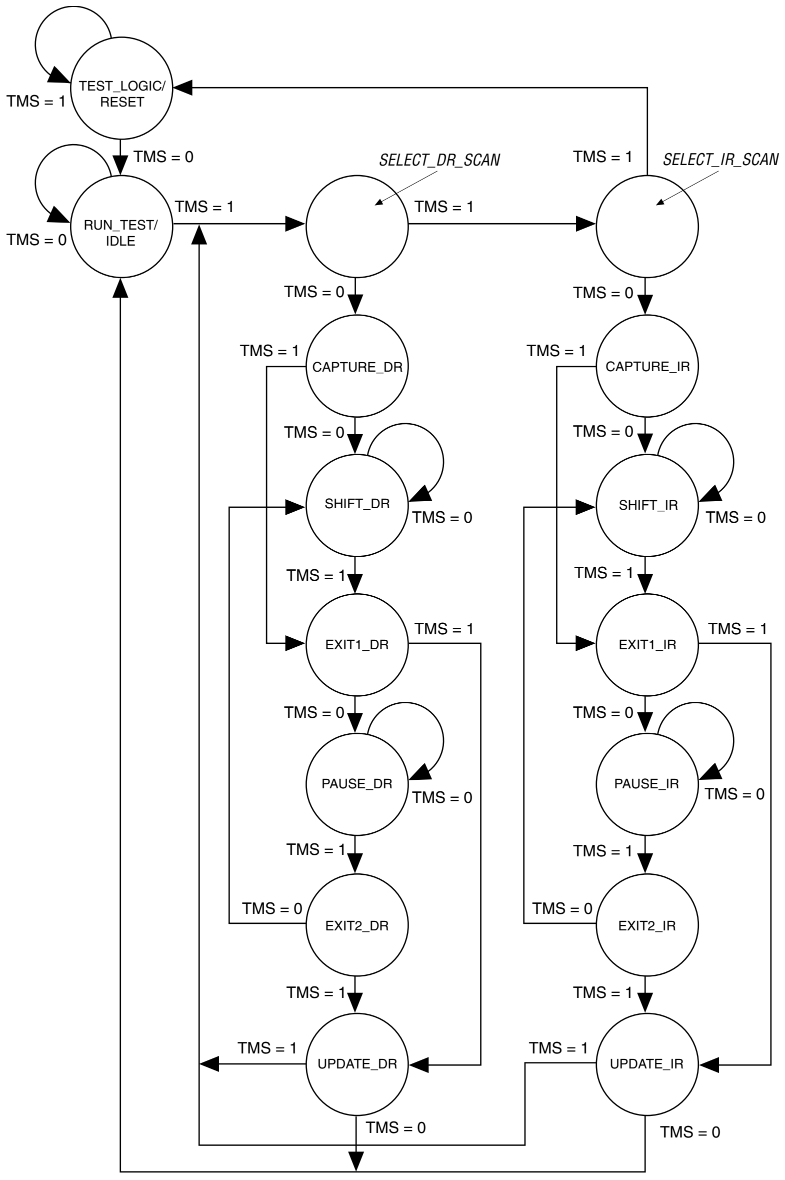
(2) Pull `DATA0` and `DCLK` to either high or low.

Both JTAG connection methods should include space for the MasterBlaster or ByteBlasterMV header connection. The header is useful during prototyping because it allows you to verify or modify the Stratix or Stratix GX device's contents. During production, you can remove the header to save cost.

Program Flow

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine. The TAP controller is a 16-state state machine that is clocked on the rising edge of `TCK`, and uses the `TMS` pin to control JTAG operation in a device. Figure 3–24 shows the flow of an IEEE Std. 1149.1 TAP controller state machine.

Figure 3–24. JTAG TAP Controller State Machine



While the Jam Player provides a driver that manipulates the TAP controller, the Jam Byte-Code File (**.jbc**) provides the high-level intelligence needed to program a given device. All Jam instructions that send JTAG data to the device involve moving the TAP controller through either the data register leg or the instruction register leg of the state machine. For example, loading a JTAG instruction involves moving the TAP controller to the `SHIFT_IR` state and shifting the instruction into the instruction register through the `TDI` pin. Next, the TAP controller is moved to the `RUN_TEST/IDLE` state where a delay is implemented to allow the instruction time to be latched. This process is identical for data register scans, except that the data register leg of the state machine is traversed.

The high-level Jam instructions are the `DRSCAN` instruction for scanning the JTAG data register, the `IRSCAN` instruction for scanning the instruction register, and the `WAIT` command that causes the state machine to sit idle for a specified period of time. Each leg of the TAP controller is scanned repeatedly, according to instructions in the JBC file, until all of the target devices are programmed.

Figure 3–25 illustrates the functional behavior of the Jam Player when it parses the JBC file. When the Jam Player encounters a `DRSCAN`, `IRSCAN`, or `WAIT` instruction, it generates the proper data on `TCK`, `TMS`, and `TDI` to complete the instruction. The flow diagram shows branches for the `DRSCAN`, `IRSCAN`, and `WAIT` instructions. Although the Jam Player supports other instructions, they are omitted from the flow diagram for simplicity.

Figure 3–25. Jam Player Flow Diagram (Part 1 of 2)

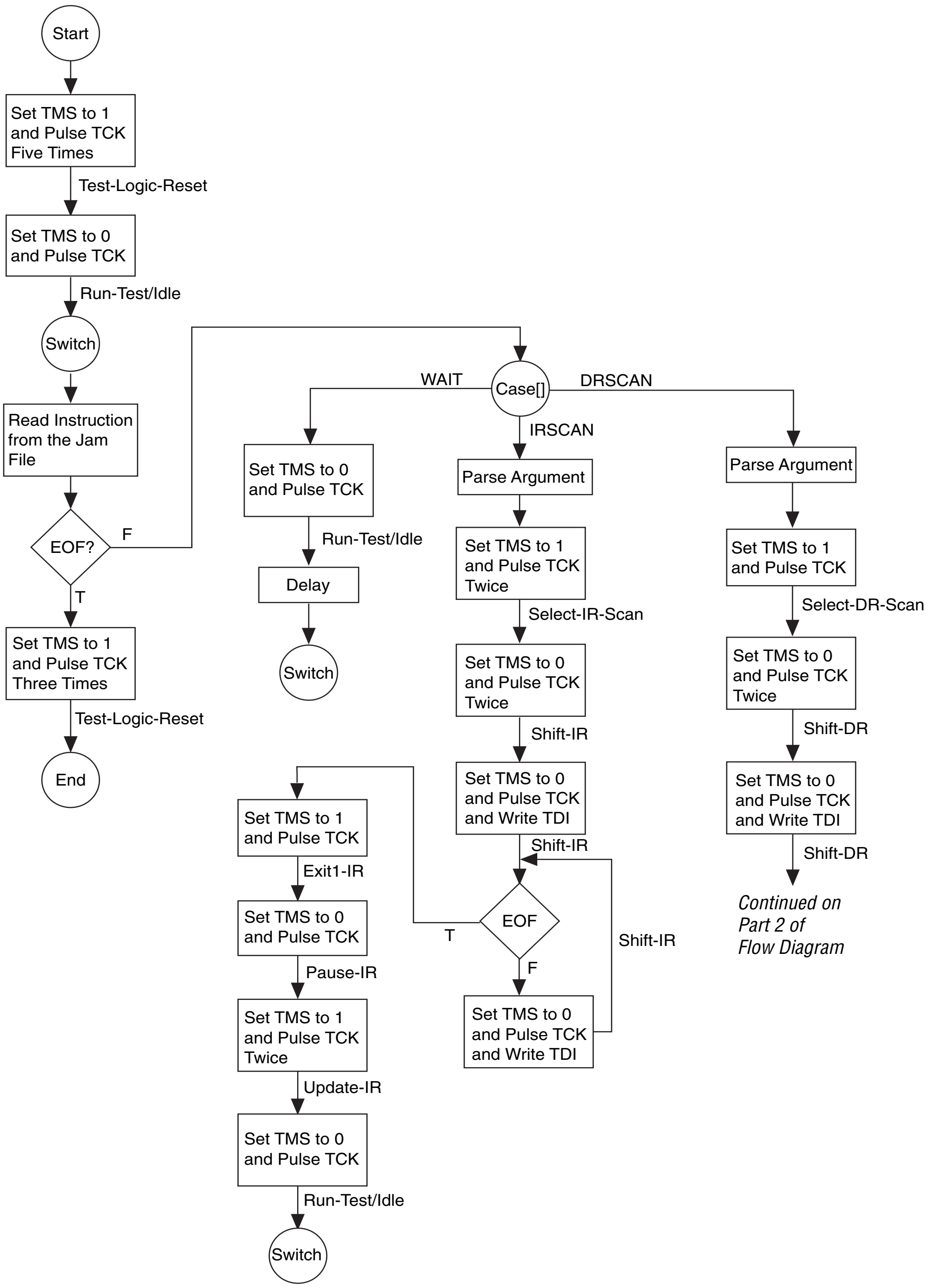
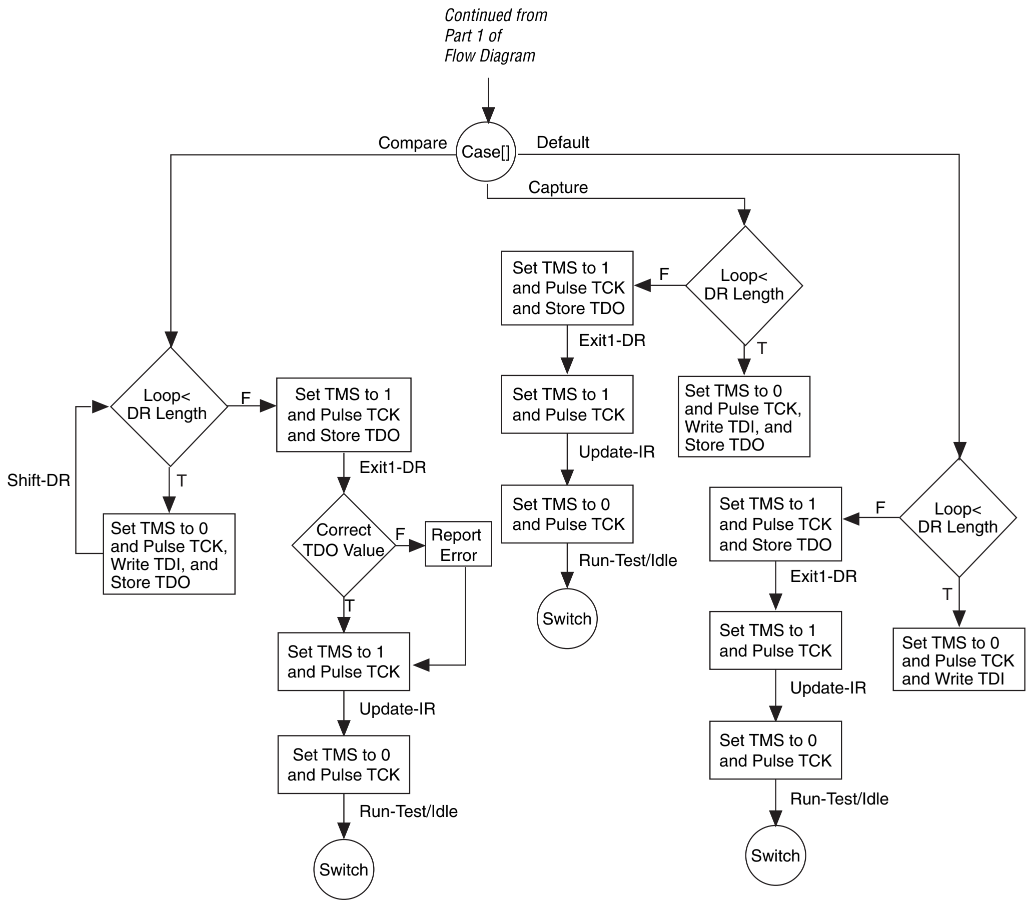

Start
Set TMS to 1 and Pulse TCK Five Times
Test-Logic-Reset
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Read Instruction from the Jam File
EOF?
F
T
Set TMS to 1 and Pulse TCK Three Times
Test-Logic-Reset
End
Case[]
WAIT
DRSCAN
IRSCAN
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Delay
Switch
Parse Argument
Set TMS to 1 and Pulse TCK Twice
Select-IR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-IR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-IR
EOF
T
F
Set TMS to 0 and Pulse TCK and Write TDI
Shift-IR
Set TMS to 1 and Pulse TCK
Exit1-IR
Set TMS to 0 and Pulse TCK
Pause-IR
Set TMS to 1 and Pulse TCK Twice
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Parse Argument
Set TMS to 1 and Pulse TCK
Select-DR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-DR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-DR
Continued on Part 2 of Flow Diagram

Figure 3–26. Jam Player Flow Diagram (Part 2 of 2)


Continued from Part 1 of Flow Diagram
Case[]
Compare
Default
Capture
Loop< DR Length
F
T
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK, Write TDI, and Store TDO
Shift-DR
Loop< DR Length
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Correct TDO Value
Report Error
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK, Write TDI, and Store TDO
Loop< DR Length
Set TMS to 1 and Pulse TCK and Store TDO
Exit1-DR
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK and Write TDI

Execution of a Jam program starts at the beginning of the program. The program flow is controlled using `GOTO`, `CALL/RETURN`, and `FOR/NEXT` structures. The `GOTO` and `CALL` statements see labels that are symbolic names for program statements located elsewhere in the Jam program. The language itself enforces almost no constraints on the organizational structure or control flow of a program.

The Jam language does not support linking multiple Jam programs together or including the contents of another file into a Jam program.

Jam Instructions

Each Jam statement begins with one of the instruction names listed in Table 3–13. The instruction names, including the names of the optional instructions, are reserved keywords that you cannot use as variable or label identifiers in a Jam program.

Table 3–13. Instruction Names

BOOLEAN	INTEGER	PREIR
CALL	IRSCAN	PRINT
CRC	IRSTOP	PUSH
DRSCAN	LET	RETURN
DRSTOP	NEXT	STATE
EXIT	NOTE	WAIT
EXPORT	POP	VECTOR *(1)*
FOR	POSTDR	VMAP *(1)*
GOTO	POSTIR	–
IF	PREDR	–

Note to Table 3–13:
(1) This instruction name is an optional language extension.

Table 3–14 shows the state names that are reserved keywords in the Jam language. These keywords correspond to the state names specified in the IEEE Std. 1149.1 JTAG specification.

Table 3–14. Reserved Keywords (Part 1 of 2)

IEEE Std. 1149.1 JTAG State Names	Jam Reserved State Names
Test-Logic-Reset	RESET
Run-Test-Idle	IDLE
Select-DR-Scan	DRSELECT
Capture-DR	DRCAPTURE
Shift-DR	DRSHIFT
Exit1-DR	DREXIT1
Pause-DR	DRPAUSE
Exit2-DR	DREXIT2
Update-DR	DRUPDATE
Select-IR-Scan	IRSELECT

Table 3–14. Reserved Keywords (Part 2 of 2)

IEEE Std. 1149.1 JTAG State Names	Jam Reserved State Names
`Capture-IR`	`IRCAPTURE`
`Shift-IR`	`IRSHIFT`
`Exit1-IR`	`IREXIT1`
`Pause-IR`	`IRPAUSE`
`Exit2-IR`	`IREXIT2`
`Update-IR`	`IRUPDATE`

Example Jam File that Reads the IDCODE

Figure 3–27 illustrates the flexibility and utility of the Jam STAPL. The example reads the IDCODE out of a single device in a JTAG chain.

☞ The array variable, `I_IDCODE`, is initialized with the IDCODE instruction bits ordered the LSB first (on the left) to most significant bit (MSB) (on the right). This order is important because the array field in the IRSCAN instruction is always interpreted, and sent, MSB to LSB.

Figure 3–27. Example Jam File Reading IDCODE

```
BOOLEAN read_data[32];
BOOLEAN I_IDCODE[10] = BIN 1001101000; ‘assumed
BOOLEAN ONES_DATA[32] = HEX FFFFFFFF;
INTEGER i;
‘Set up stop state for IRSCAN
IRSTOP IRPAUSE;
‘Initialize device
STATE RESET;
IRSCAN 10, I_IDCODE[0..9]; ‘LOAD IDCODE INSTRUCTION
STATE IDLE;
WAIT 5 USEC, 3 CYCLES;
DRSCAN 32, ONES_DATA[0..31], CAPTURE
read_data[0..31];
‘CAPTURE IDCODE
PRINT “IDCODE:”;
FOR i=0 to 31;
PRINT read_data[i];
NEXT i;
EXIT 0;
```

Configuring Using the MicroBlaster Driver

The MicroBlaster™ software driver allows you to configure Altera devices in an embedded environment using PS or FPP mode. The MicroBlaster software driver supports a Raw Binary File (**.rbf**) programming input file. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, go to the Altera web site (**www.altera.com**).

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the Stratix or Stratix GX device. Table 3–15 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 1 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
VCCSEL	N/A	All	Input	Dedicated input that selects which input buffer is used on the configuration input pins; nCONFIG, DCLK, RUnLU, nCE, nWS, nRS, CS, nCS and CLKUSR. The VCCSEL input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) selects the 1.8-V/1.5-V input buffer, and a logic low selects the 3.3-V/2.5-V input buffer. See the "VCCSEL Pins" section for more details.
PORSEL	N/A	All	Input	Dedicated input which selects between a POR time of 2 ms or 100 ms. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) selects a POR time of about 2 ms and a logic low selects POR time of about 100 ms. The PORSEL input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 2 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nIO_PULLUP	N/A	All	Input	Dedicated input that chooses whether the internal pull-ups on the user I/Os and dual-purpose I/Os (DATA[7..0], nWS, nRS, RDYnBSY, nCS, CS, RUnLU, PGM[], CLKUSR, INIT_DONE, DEV_OE, DEV_CLR) are on or off before and during configuration. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) turns off the weak internal pull-ups, while a logic low turns them on. The nIO_PULLUP input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active.
MSEL[2..0]	N/A	All	Input	3-bit configuration input that sets the Stratix or Stratix GX device configuration scheme. See Table 3–2 for the appropriate connections. These pins can be connected to V_{CCIO} of the I/O bank they reside in or ground. This pin uses Schmitt trigger input buffers.
nCONFIG	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode causes the FPGA to lose its configuration data, enter a reset state, tri-state all I/O pins. Returning this pin to a logic high level initiates a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, nCONFIG can be tied directly to V_{CC} or to the configuration device's nINIT_CONF pin. This pin uses Schmitt trigger input buffers.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 3 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nSTATUS	N/A	All	Bidirectional open-drain	The device drives nSTATUS low immediately after power-up and releases it after the POR time. Status output. If an error occurs during configuration, nSTATUS is pulled low by the target device. Status input. If an external source drives the nSTATUS pin low during configuration or initialization, the target device enters an error state. Driving nSTATUS low after configuration and initialization does not affect the configured device. If a configuration device is used, driving nSTATUS low causes the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on nSTATUS in user-mode, the FPGA does not reconfigure. To initiate a reconfiguration, nCONFIG must be pulled low. The enhanced configuration devices' and EPC2 devices' OE and nCS pins have optional internal programmable pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, external 10-kΩ pull-up resistors should not be used on these pins. When using EPC2 devices, only external 10-kΩ pull-up resistors should be used. This pin uses Schmitt trigger input buffers.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 4 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
CONF_DONE	N/A	All	Bidirectional open-drain	Status output. The target FPGA drives the CONF_DONE pin low before and during configuration. Once all configuration data is received without error and the initialization cycle starts, the target device releases CONF_DONE. Status input. After all data is received and CONF_DONE goes high, the target device initializes and enters user mode. Driving CONF_DONE low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' OE and nCS pins have optional internal programmable pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, external 10-kΩ pull-up resistors should not be used on these pins. When using EPC2 devices, only external 10-kΩ pull-up resistors should be used. This pin uses Schmitt trigger input buffers.
nCE	N/A	All	Input	Active-low chip enable. The nCE pin activates the device with a low signal to allow configuration. The nCE pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, nCE of the first device is tied low while its nCEO pin is connected to nCE of the next device in the chain. The nCE pin must also be held low for successful JTAG programming of the FPGA. This pin uses Schmitt trigger input buffers.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 5 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nCEO`	N/A	All Multi-Device Schemes	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's `nCE` pin. The `nCEO` of the last device in the chain is left floating. The voltage levels driven out by this pin are dependent on the V_{CCIO} of the I/O bank it resides in.
`DCLK`	N/A	Synchronous configuration schemes (PS, FPP)	Input (PS, FPP)	In PS and FPP configuration, `DCLK` is the clock input used to clock data from an external source into the target device. Data is latched into the FPGA on the rising edge of `DCLK`. In PPA mode, `DCLK` should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, `DCLK` is driven low after configuration is done. In schemes that use a control host, `DCLK` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device. This pin uses Schmitt trigger input buffers.
`DATA0`	I/O	PS, FPP, PPA	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the `DATA0` pin. The V_{IH} and V_{IL} levels for this pin are dependent on the V_{CCIO} of the I/O bank that it resides in. After configuration, `DATA0` is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings. After configuration, EPC1 and EPC1441 devices tri-state this pin, while enhanced configuration and EPC2 devices drive this pin high.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 6 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`DATA[7..1]`	I/O	Parallel configuration schemes (FPP and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on `DATA[7..0]`. The V_{IH} and V_{IL} levels for these pins are dependent on the V_{CCIO} of the I/O banks that they reside in. In serial configuration schemes, they function as user I/Os during configuration, which means they are tri-stated. After PPA or FPP configuration, `DATA[7..1]` are available as a user I/Os and the state of these pin depends on the **Dual-Purpose Pin** settings.
`DATA7`	I/O	PPA	Bidirectional	In the PPA configuration scheme, the `DATA7` pin presents the `RDYnBSY` signal after the `nRS` signal has been strobed low. The V_{IL} and V_{IL} levels for this pin are dependent on the V_{CCIO} of the I/O bank that it resides in. In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `DATA7` is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings.
`nWS`	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the `DATA[7..0]` pins. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nWS` is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 7 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nRS	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the RDYnBSY signal on the DATA7 pin. If the nRS pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nRS is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings.
RDYnBSY	I/O	PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPA configuration schemes, this pin drives out high after power-up, before configuration and after configuration before entering user-mode. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, RDYnBSY is available as a user I/O and the state of this pin depends on the **Dual-Purpose Pin** settings.
nCS/CS	I/O	PPA	Input	Chip-select inputs. A low on nCS and a high on CS select the target device for configuration. The nCS and CS pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the nCS or CS pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, nCS can be tied to GND while CS is toggled to control configuration.In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nCS and CS are available as a user I/Os and the state of these pins depends on the **Dual-Purpose Pin** settings.

Table 3–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 8 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
RUnLU	N/A if using Remote Configuration; I/O if not	Remote Configuration in FPP, PS or PPA	Input	Input that selects between remote update and local update. A logic high (1.5-V, 1.8-V, 2.5-V, 3.3-V) selects remote update and a logic low selects local update. When not using remote update or local update configuration modes, this pins is available as general-purpose user I/O pin.
PGM[2..0]	N/A if using Remote Configuration; I/O if not using	Remote Configuration in FPP, PS or PPA	Input	These output pins select one of eight pages in the memory (either flash or enhanced configuration device) when using a remote configuration mode. When not using remote update or local update configuration modes, these pins are available as general-purpose user I/O pins.

Table 3–16 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration, these pins function as user I/O pins and are tri-stated with weak pull-ups.

Table 3–16. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
CLKUSR	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the **Enable user-supplied start-up clock (CLKUSR)** option in the Quartus II software.
INIT_DONE	N/A if option is on. I/O if option is off.	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. When nCONFIG is low and during the beginning of configuration, the INIT_DONE pin is tri-stated and pulled high due to an external 10-kΩ 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 goes low. When initialization is complete, the INIT_DONE pin is released and pulled high and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the **Enable INIT_DONE output** option in the Quartus II software.

Table 3–16. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
`DEV_OE`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/Os are tri-stated. When this pin is driven high, all I/Os behave as programmed. This pin is enabled by turning on the **Enable device-wide output enable (DEV_OE)** option in the Quartus II software.
`DEV_CLRn`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared. When this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the **Enable device-wide reset (DEV_CLRn)** option in the Quartus II software.

Table 3–17 describes the dedicated JTAG pins. JTAG pins must be kept stable before and during configuration to prevent accidental loading of JTAG instructions. If you plan to use the SignalTap II Embedded Logic Analyzer, you will need to connect the JTAG pins of your device to a JTAG header on your board.

Table 3–17. Dedicated JTAG pins

Pin Name	User Mode	Pin Type	Description
`TDI`	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}. This pin uses Schmitt trigger input buffers.
`TDO`	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, `TMS` must be set up before the rising edge of TCK. `TMS` is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}. This pin uses Schmitt trigger input buffers.

Table 3–17. Dedicated JTAG pins

Pin Name	User Mode	Pin Type	Description
TCK	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND. This pin uses Schmitt trigger input buffers.
TRST	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The TRST pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND. This pin uses Schmitt trigger input buffers.

4. Configuring Cyclone II Devices

CII51013-1.1

Introduction

Cyclone™ II devices use SRAM cells to store configuration data. Since SRAM memory is volatile, configuration data must be downloaded to Cyclone II devices each time the device powers up. You can use the active serial (AS) configuration scheme, which can operate at a DCLK frequency up to 40 MHz, to configure Cyclone II devices. You can also use the passive serial (PS) and Joint Test Action Group (JTAG)-based configuration schemes to configure Cyclone II devices. Additionally, Cyclone II devices can receive a compressed configuration bitstream and decompress this data on-the-fly, reducing storage requirements and configuration time.

This chapter explains the Cyclone II configuration features and describes how to configure Cyclone II devices using the supported configuration schemes. This chapter also includes configuration pin descriptions and the Cyclone II configuration file format.

For more information on setting device configuration options or creating configuration files, see the *Software Settings* chapter in the *Configuration Handbook*.

Cyclone II Configuration Overview

You can use the AS, PS, and JTAG configuration schemes to configure Cyclone II devices. You can select which configuration scheme to use by driving the Cyclone II device MSEL pins either high or low as shown in Table 4–1. The MSEL pins are powered by the V_{CCIO} power supply of the bank they reside in. During power-on reset (POR) and reconfiguration, the MSEL pins have to be at LVTTL V_{IL} or V_{IH} levels to be considered a logic low or logic high, respectively. Therefore, to avoid any problems with detecting an incorrect configuration scheme, you should connect the MSEL[] pins to the V_{CCIO} of the I/O bank they reside in and GND without any pull-up or pull-down resistors. The MSEL[] pins should not be driven by a microprocessor or another device.

Table 4–1. Cyclone II Configuration Schemes

Configuration Scheme	MSEL1	MSEL0
AS (20 MHz) *(1)*	0	0
PS	0	1
Fast AS (40 MHz) *(1)*	1	0
JTAG-based Configuration *(2)*	*(3)*	*(3)*

Notes to Table 4–1:

(1) Only the EPCS16 and EPCS64 devices support a DCLK up to 40 MHz. Other EPCS devices support a DCLK up to 20 MHz. See the *Serial Configuration Devices Data Sheet* for more information.

(2) JTAG-based configuration takes precedence over other configuration schemes, which means MSEL pin settings are ignored.

(3) Do not leave the MSEL pins floating; connect them to V_{CCIO} or ground. These pins support the non-JTAG configuration scheme used in production. If you are only using JTAG configuration, you should connect the MSEL pins to ground.

You can download configuration data to Cyclone II FPGAs with the AS, PS, or JTAG interfaces using the options in Table 4–2.

Table 4–2. Cyclone II Device Configuration Schemes

Configuration Scheme	Description
AS configuration	Configuration using serial configuration devices (EPCS1, EPCS4, EPCS16 or EPCS64 devices)
PS configuration	Configuration using enhanced configuration devices (EPC4, EPC8, and EPC16 devices), EPC2 and EPC1 configuration devices, an intelligent host (microprocessor), or a download cable
JTAG-based configuration	Configuration via JTAG pins using a download cable, an intelligent host (microprocessor), or the Jam™ Standard Test and Programming Language (STAPL)

Configuration File Format

Table 4–3 shows the approximate uncompressed configuration file sizes for Cyclone II devices. To calculate the amount of storage space required for multiple device configurations, add the file size of each device together.

Table 4–3. Cyclone II Raw Binary File (.rbf) Sizes *Note (1)*

Device	Data Size (Bits)	Data Size (Bytes)
EP2C5	1,265,792	152,998
EP2C8	1,983,536	247,974
EP2C20	3,892,496	486,562
EP2C35	6,858,656	857,332
EP2C50	9,963,392	1,245,424
EP2C70	14,319,216	1,789,902

Note to Table 4–3:
(1) These values are preliminary.

Use the data in Table 4–3 only to estimate the file size before design compilation. Different configuration file formats, such as a Hexadecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II software, any design targeted for the same device will have the same uncompressed configuration file size. If compression is used, the file size can vary after each compilation since the compression ratio is dependent on the design.

Configuration Data Compression

Cyclone II devices support configuration data decompression, which saves configuration memory space and time. This feature allows you to store compressed configuration data in configuration devices or other memory and transmit this compressed bitstream to Cyclone II devices. During configuration, the Cyclone II device decompresses the bitstream in real time and programs its SRAM cells.

☞ Preliminary data indicates that compression reduces configuration bitstream size by 35 to 55%.

Cyclone II devices support decompression in the AS and PS configuration schemes. Decompression is not supported in JTAG-based configuration.

Although they both use the same compression algorithm, the decompression feature supported by Cyclone II devices is different from the decompression feature in enhanced configuration devices (EPC16, EPC8, and EPC4 devices). The data decompression feature in the enhanced configuration devices allows them to store compressed data and decompress the bitstream before transmitting it to the target devices.

In PS mode, you should use the Cyclone II decompression feature since sending compressed configuration data reduces configuration time. You should not use both the Cyclone II device and the enhanced configuration device decompression features simultaneously. The compression algorithm is not intended to be recursive and could expand the configuration file instead of compressing it further.

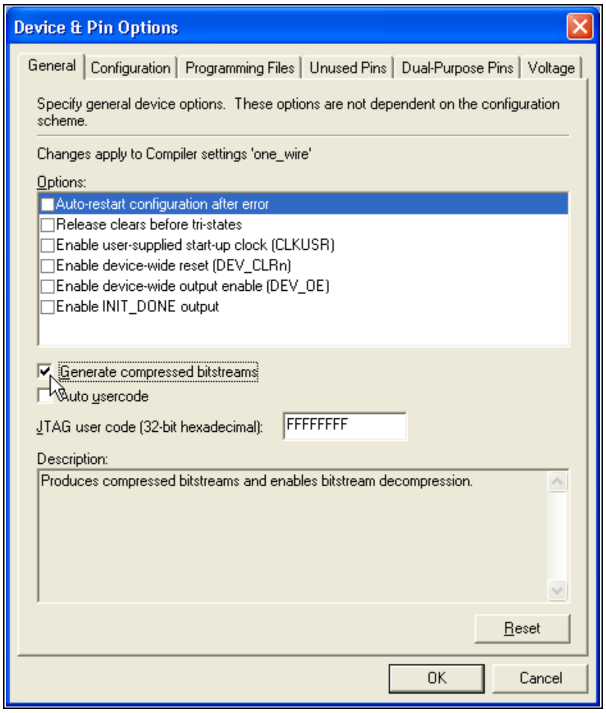
You should use the Cyclone II decompression feature during AS configuration if you need to save configuration memory space in the serial configuration device.

When you enable compression, the Quartus II software generates configuration files with compressed configuration data. This compressed file reduces the storage requirements in the configuration device or flash, and decreases the time needed to transmit the bitstream to the Cyclone II device. The time required by a Cyclone II device to decompress a configuration file is less than the time needed to transmit the configuration data to the FPGA.

There are two methods to enable compression for Cyclone II bitstreams: before design compilation (in the Compiler Settings menu) and after design compilation (in the **Convert Programming Files** window).

To enable compression in the project's compiler settings, select **Device** under the Assignments menu to bring up the settings window. After selecting your Cyclone II device open the **Device & Pin Options** window, and in the **General settings** tab enable the check box for **Generate compressed bitstreams** (see Figure 4–1).

Figure 4–1. Enabling Compression for Cyclone II Bitstreams in Compiler Settings


Device & Pin Options

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

Changes apply to Compiler settings 'one_wire'

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:

Produces compressed bitstreams and enables bitstream decompression.

You can also use the following steps to enable compression when creating programming files from the Convert Programming Files window.

1. Click **Convert Programming Files** (File menu).

2. Select the Programming File type. Only Programmer Object Files (**.pof**), SRAM HEXOUT, RBF, or TTF files support compression.

3. For POFs, select a configuration device.

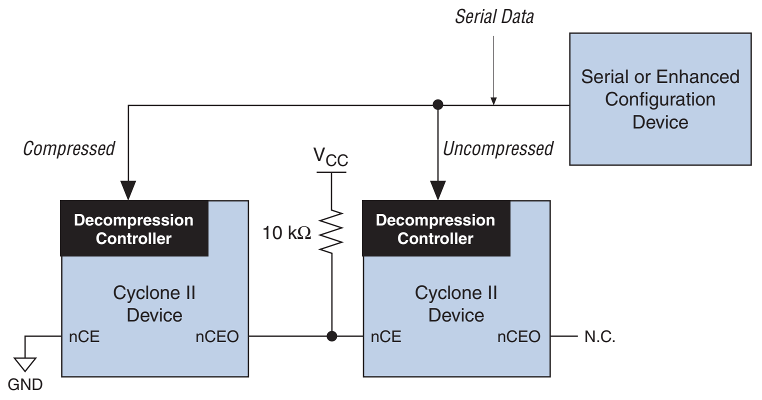
4. Select **Add File** and add a Cyclone II SRAM Object File(s) (**.sof**).

5. Select the name of the file you added to the SOF Data area and click on **Properties**.

6. Check the **Compression** check box.

When multiple Cyclone II devices are cascaded, the compression feature can be selectively enabled for each device in the chain. Figure 4–2 depicts a chain of two Cyclone II devices. The first Cyclone II device has compression enabled and therefore receives a compressed bitstream from the configuration device. The second Cyclone II device has the compression feature disabled and receives uncompressed data.

Figure 4–2. Compressed & Uncompressed Configuration Data in a Programming File



You can generate programming files (for example, POF files) for this setup in the Quartus II software.

Active Serial Configuration (Serial Configuration Devices)

In the AS configuration scheme, Cyclone II devices are configured using a serial configuration device. These configuration devices are low-cost devices with non-volatile memory that feature a simple, four-pin interface and a small form factor. These features make serial configuration devices an ideal low-cost configuration solution.

For more information on serial configuration devices, see the *Serial Configuration Devices Data Sheet* in the Configuration Handbook.

Serial configuration devices provide a serial interface to access configuration data. During device configuration, Cyclone II devices read configuration data via the serial interface, decompress data if necessary, and configure their SRAM cells. The FPGA controls the configuration interface in the AS configuration scheme, while the external host (e.g., the configuration device or microprocessor) controls the interface in the PS configuration scheme.

☞ The Cyclone II decompression feature is available when configuring your Cyclone II device using AS mode.

Table 4–4 shows the MSEL pin settings when using the AS configuration scheme.

Table 4–4. Cyclone II Configuration Schemes

Configuration Scheme	MSEL1	MSEL0
AS (20 MHz) *(1)*	0	0
Fast AS (40 MHz) *(1)*	1	0

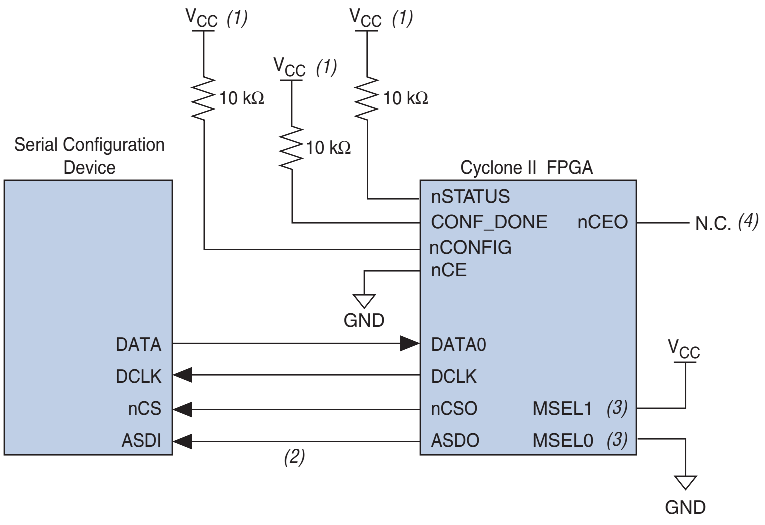
Note to Table 4–4:

(1) The EPCS16 and EPCS64 supported a DCLK up to 40 MHz. Other EPCS devices support a DCLK up to 20 MHz. See the *Serial Configuration Devices Data Sheet* for more information.

Single Device AS Configuration

Serial configuration devices have a four-pin interface: serial clock input (DCLK), serial data output (DATA), AS data input (ASDI), and an active-low chip select (nCS). This four-pin interface connects to Cyclone II device pins, as shown in Figure 4–3.

Figure 4–3. Single Device AS Configuration



Notes to Figure 4–3:

(1) Connect the pull-up resistors to a 3.3-V supply.
(2) Cyclone II devices use the `ASDO` to `ASDI` path to control the configuration device.
(3) If your design uses an EPCS4 or an EPCS1 device, set the `MSEL[1..0]` pins to 00. See Table 4–4 for more details.
(4) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed another device's `nCE` pin.

Upon power-up, the Cyclone II device goes through a POR. During POR, the device will reset, hold `nSTATUS` and `CONF_DONE` low, and tri-state all user I/O pins. After POR, which typically lasts 100 ms, the Cyclone II device releases `nSTATUS` and enters configuration mode when the external 10-kΩ resistor pulls the `nSTATUS` pin high. Once the FPGA successfully exits POR, all user I/O pins continue to be tri-stated. Cyclone II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration are available in the *DC Characteristics & Timing Specifications* chapter of the *Cyclone II Device Handbook*.

The configuration cycle consists of the reset, configuration, and initialization stages.

Reset Stage

When nCONFIG or nSTATUS are low, the device is in reset. After POR, the Cyclone II device releases nSTATUS. An external 10-kΩ pull-up resistor pulls the nSTATUS signal high, and the Cyclone II device enters configuration mode.

☞ V_{CCINT} and V_{CCIO} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

Configuration Stage

The serial clock (DCLK) generated by the Cyclone II device controls the entire configuration cycle and provides the timing for the serial interface. Cyclone II devices use an internal oscillator to generate DCLK. Using the MSEL[] pins, you can select either a 20- or 40-MHz oscillator. Although you can select either 20- or 40-MHz oscillator when designing with EPCS16 and EPCS64 serial configuration devices, the 40-MHz oscillator provides faster configuration times. There is some variation in the internal oscillator frequency because of the process, temperature, and voltage conditions in Cyclone II devices. The internal oscillator is designed such that its maximum frequency is guaranteed to meet EPCS device specifications.

☞ Only the EPCS16 and EPCS64 configuration devices support a DCLK up to 40 MHz. Other serial configuration devices support a DCLK up to 20 MHz. See the *Serial Configuration Devices Data Sheet* for more information.

Table 4–5 shows the AS DCLK output frequencies.

Table 4–5. AS DCLK Output Frequency *Note (1)*

Oscillator Selected	Minimum	Typical	Maximum	Units
40 MHz *(2)*	20	26	40	MHz
20 MHz	10	13	20	MHz

Notes to Table 4–5:
(1) These values are preliminary.
(2) The EPCS16 and EPCS64 devices support a DCLK clock up to 40 MHz. Other serial configuration devices support a DCLK clock up to 20 MHz. See the *Serial Configuration Devices Data Sheet* for more information.

The serial configuration device latches input/control signals on the rising edge of DCLK and drives out configuration data on the falling edge. Cyclone II devices drive out control signals on the falling edge of DCLK and latch configuration data on the rising edge of DCLK.

In configuration mode, the Cyclone II device enables the serial configuration device by driving its nCSO output pin low, which connects to the chip select (nCS) pin of the configuration device. The Cyclone II device uses the serial clock (DCLK) and serial data output (ASDO) pins to send operation commands and/or read address signals to the serial configuration device. The configuration device then provides data on its serial data output (DATA) pin, which connects to the DATA0 input of the Cyclone II device.

After the Cyclone II device receives all the configuration bits, it releases the open-drain CONF_DONE pin, which is then pulled high by an external 10-kΩ resistor. Also, the Cyclone II device stops driving the DCLK signal. Initialization begins only after the CONF_DONE signal reaches a logic high level. All AS configuration pins (DATA0, DCLK, nCSO, and ASDO) have weak internal pull-up resistors which are always active. Therefore, after configuration, these pins will be driven high.

Initialization Stage

In Cyclone II devices, the initialization clock source is either the Cyclone II 10-MHz (typical) internal oscillator (separate from the AS internal oscillator) or the optional CLKUSR pin. The internal oscillator is the default clock source for initialization. If the internal oscillator is used, the Cyclone II device will provide itself with enough clock cycles for proper initialization. The advantage of using the internal oscillator is you do not need to send additional clock cycles from an external source to the CLKUSR pin during the initialization stage. Additionally, you can use the CLKUSR pin as a user I/O pin.

If you want to delay the initialization of the device, you can use the CLKUSR pin option. Using the CLKUSR pin allows you to control when your device enters user mode. The device can be delayed from entering user mode for an indefinite amount of time. When you enable the **User Supplied Start-Up Clock** option, the CLKUSR pin is the initialization clock source. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Cyclone II devices require 299 clock cycles to initialize properly and support a CLKUSR f_{MAX} of 100 MHz.

Cyclone II devices offer an optional INIT_DONE pin which signals the end of initialization and the start of user mode with a low-to-high transition. The **Enable INIT_DONE output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** window. If you use the INIT_DONE pin, an external 10-kΩ pull-up resistor is required to pull the signal high when nCONFIG is low and during the beginning of configuration. Once the optional 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 that the FPGA has entered user mode. If you do not use the INIT_DONE pin, the initialization period will be complete after CONF_DONE goes high and 299 clock cycles are sent to the CLKUSR pin or after the time t_{CF2UM} (see Table 4–8) if the Cyclone II device uses the internal oscillator.

User Mode

When initialization is complete, the FPGA enters user mode. In user mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

When the Cyclone II device is in user mode, you can initiate reconfiguration by pulling the nCONFIG signal low. The nCONFIG signal should be low for at least 40 µs. When nCONFIG is pulled low, the Cyclone II device is reset and enters the reset stage. The Cyclone II device also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the Cyclone II device, reconfiguration begins.

Error During Configuration

If an error occurs during configuration, the Cyclone II device drives the nSTATUS signal low to indicate a data frame error, and the CONF_DONE signal will stay low. If you enable the **Auto-restart configuration after error** option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box, the Cyclone II device resets the serial configuration device by pulsing nCSO, releases nSTATUS after a reset time-out period (about 40 µs), and retries configuration. If the **Auto-restart configuration after error** option is turned off, the external system must monitor nSTATUS for errors and then pull nCONFIG low for at least 40 µs to restart configuration.

If you use the optional CLKUSR pin and the nCONFIG pin is pulled low to restart configuration during device initialization, ensure CLKUSR continues to toggle during the time nSTATUS is low (a maximum of 40 µs).

For more information on configuration issues, see the *Debugging Configuration Problems* chapter of the *Configuration Handbook* and the FPGA Configuration Troubleshooter on the Altera web site (**www.altera.com**).

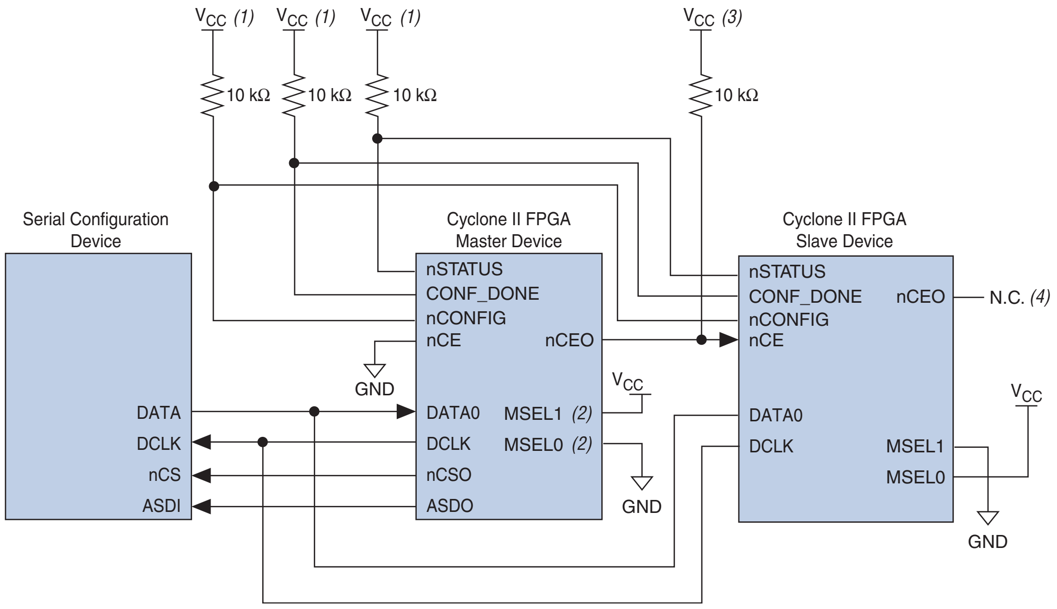
Multiple Device AS Configuration

You can configure multiple Cyclone II devices using a single serial configuration device. You can cascade multiple Cyclone II devices using the chip-enable (`nCE`) and chip-enable-out (`nCEO`) pins. Connect the `nCE` pin of the first device in the chain to ground and connect the `nCEO` pin to the `nCE` pin of the next device in the chain. Use an external 10-kΩ pull-up resistor to pull the `nCEO` signal high to its V_{CCIO} level to help the internal weak pull-up resistor. When the first device captures all of its configuration data from the bitstream, it transitions its `nCEO` pin low, initiating the configuration of the next device in the chain. You can leave the `nCEO` pin of the last device unconnected or use it as a user I/O pin after configuration if the last device in chain is a Cyclone II device.

The Quartus II software sets the Cyclone II device `nCEO` pin as an output pin driving to ground by default. If the device is in a chain, and the `nCEO` pin is connected to the next device's `nCE` pin, you must make sure that the `nCEO` pin is not used as a user I/O pin after configuration. The software setting is in the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box in Quartus II software.

The first Cyclone II device in the chain is the configuration master and controls the configuration of the entire chain. Select the AS configuration scheme for the first Cyclone II device and the PS configuration scheme for the remaining Cyclone II devices (configuration slaves). Any other Altera® device that supports PS configuration can also be part of the chain as a configuration slave. In a multiple device chain, the `nCONFIG`, `nSTATUS`, `CONF_DONE`, `DCLK`, and `DATA0` pins of each device in the chain are connected (see Figure 4–4). Figure 4–4 shows the pin connections for this setup.

Figure 4–4. Multiple Device AS Configuration



Notes to Figure 4–4:

(1) Connect the pull-up resistors to a 3.3-V supply.

(2) If using an EPCS4 or an EPCS1 device, set `MSEL[1..0]` to `00`. See Table 4–4 for more details.

(3) Connect the pull-up resistor to the V_{CCIO} supply voltage of I/O bank that the `nCEO` pin resides in.

(4) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed another device's `nCE` pin.

As shown in Figure 4–4, the `nSTATUS` and `CONF_DONE` pins on all target FPGAs are connected together with external pull-up resistors. These pins are open-drain bidirectional pins on the FPGAs. When the first device asserts `nCEO` (after receiving all of its configuration data), it releases its `CONF_DONE` pin. However, the subsequent devices in the chain keep the `CONF_DONE` signal low until they receive their configuration data. When all the target FPGAs in the chain have received their configuration data and have released `CONF_DONE`, the pull-up resistor pulls this signal high, and all devices simultaneously enter initialization mode.

During initialization, the initialization clock source is either the Cyclone II 10 MHz (typical) internal oscillator (separate from the AS internal oscillator) or the optional `CLKUSR` pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Cyclone II device will provide itself with enough clock cycles for proper initialization. The advantage of using the internal oscillator is you do not need to send additional clock cycles from an external source to the `CLKUSR` pin during the initialization stage. You can also make use of the `CLKUSR` pin as a user I/O pin, which means you have an additional user I/O pin.

If you want to delay the initialization of the devices in the chain, you can use the `CLKUSR` pin option. The `CLKUSR` pin allows you to control when your device enters user mode. This feature also allows you to control the order of when each device enters user mode by feeding a separate clock to each device's `CLKUSR` pin. By using the `CLKUSR` pins, you can choose any device in the multiple device chain to enter user mode first and have the other devices enter user mode at a later time.

Different device families may require a different number of initialization clock cycles. Therefore, if your multiple device chain consists of devices from different families, the devices may enter user mode at a slightly different time due to the different number of initialization clock cycles required. However, if the number of initialization clock cycles is similar across different device families or if the devices are from the same family, then the devices enter user mode at the same time. See the respective device family handbook for more information about the number of initialization clock cycles required.

If an error occurs at any point during configuration, the FPGA with the error drives the `nSTATUS` signal low. If you enable the **Auto-restart configuration after error** option, the entire chain begins reconfiguration after a reset time-out period (a maximum of 40 µs). If the **Auto-restart configuration after error** option is turned off, a microprocessor or controller must monitor `nSTATUS` for errors and then pulse `nCONFIG` low to restart configuration. The microprocessor or controller can pulse `nCONFIG` if it is under system control rather than tied to V_{CC}.

☞ While you can cascade Cyclone II devices, serial configuration devices cannot be cascaded or chained together.

☞ If you use the optional `CLKUSR` pin and the `nCONFIG` is pulled low to restart configuration during device initialization, make sure the `CLKUSR` pin continues to toggle while `nSTATUS` is low (a maximum of 40 µs).

If the configuration bitstream size exceeds the capacity of a serial configuration device, you must select a larger configuration device and/or enable the compression feature. When configuring multiple devices, the size of the bitstream is the sum of the individual devices' configuration bitstreams.

Configuring Multiple Cyclone II Devices with the Same Design

Certain designs require you to configure multiple Cyclone II devices with the same design through a configuration bitstream or SOF. You can do this through one of two methods, as described in this section. For both methods, the serial configuration devices cannot be cascaded or chained together.

Multiple SOFs

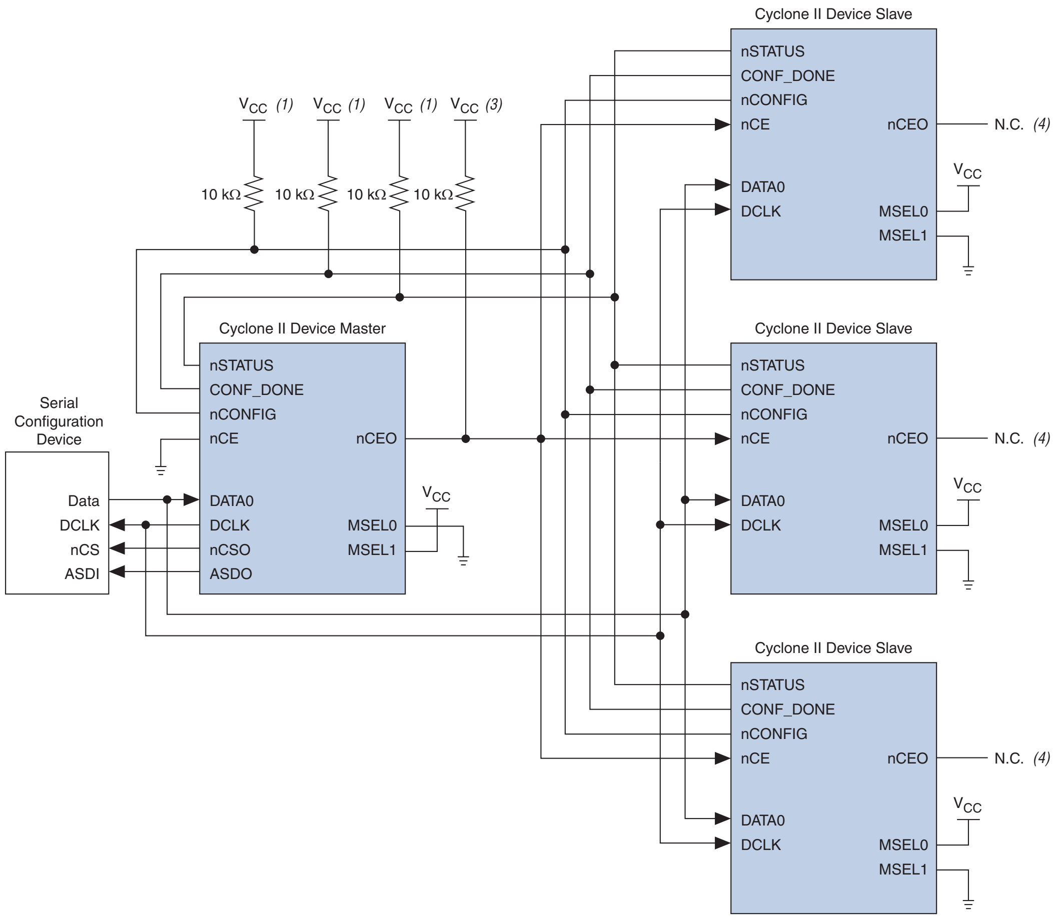
In the first method, two copies of the SOF file are stored in the serial configuration device. Use the first copy to configure the master Cyclone II device and the second copy to configure all remaining slave devices concurrently. In this setup, the master Cyclone II device is in AS mode, and the slave Cyclone II devices are in PS mode (`MSEL=01`). See Figure 4–5.

To configure four identical Cyclone II devices with the same SOF file, connect the three slave devices for concurrent configuration as shown in Figure 4–5. The `nCEO` pin from the master device drives the `nCE` input pins on all three slave devices. Connect the configuration device's `DATA` and `DCLK` pins to the Cyclone II device's `DATA` and `DCLK` pins in parallel. During the first configuration cycle, the master device reads its configuration data from the serial configuration device while holding `nCEO` high. After completing its configuration cycle, the master drives `nCE` low and transmits the second copy of the configuration data to all three slave devices, configuring them simultaneously.

The advantage of using the setup in Figure 4–5 is that you can have a different SOF file for the Cyclone II master device. However, all the Cyclone II slave devices must be configured with the same SOF file. The SOF files in this configuration method can be either compressed or uncompressed.

☞ You can still use this method if the master and slave Cyclone II devices use the same SOF.

Figure 4–5. Multiple Device AS Configuration When FPGAs Receive the Same Data with Multiple SOFs



Notes to Figure 4–5:

(1) Connect the pull-up resistors to a 3.3-V supply.

(2) If your design uses an EPCS4 or EPCS1 device, set the `MSEL[1..0]` pins to `00`. See Table 4–4 for more details.

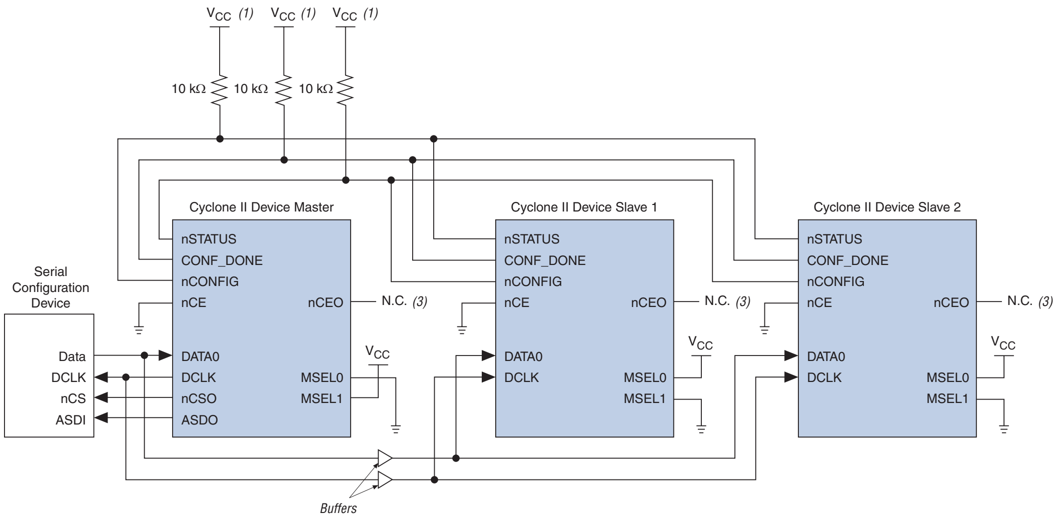
(3) Connect the pull-up resistor to the V_{CCIO} supply voltage of I/O bank that the `nCEO` pin resides in.

(4) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed another device's `nCE` pin.

Single SOF

The second method configures both the master and slave Cyclone II devices with the same SOF. The serial configuration device stores one copy of the SOF file. This setup is shown in Figure 4–6 where the master is setup in AS mode, and the slave devices are setup in PS mode (`MSEL=01`). You could setup one or more slave devices in the chain and all the slave devices are setup in the same way as shown in Figure 4–6.

Figure 4–6. Multiple Device AS Configuration When FPGAs Receive the Same Data with a Single SOF



Notes to Figure 4–6:

(1) Connect the pull-up resistors to a 3.3-V supply.

(2) If your design uses an EPCS4 or EPCS1 device, set the `MSEL[1..0]` pins to `00`. See Table 4–4 for more details.

(3) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed another device's `nCE` pin.

In this setup, all the Cyclone II devices in the chain are connected for concurrent configuration. This can reduce the AS configuration time because all the Cyclone II devices are configured in one configuration cycle. Connect the `nCE` input pins of all the Cyclone II devices to ground. You can either leave the `nCEO` output pins on all the Cyclone II devices unconnected or use the `nCEO` output pins as normal user I/O pins. The `DATA` and `DCLK` pins are connected in parallel to all the Cyclone II devices.

You should put a buffer before the DATA and DCLK output from the master Cyclone II device to avoid signal strength and signal integrity issues. The buffer should not significantly change the DATA-to-DCLK relationships or delay them with respect to other AS signals (ASDI and nCS). Also, the buffer should only drive the slave Cyclone II devices, so that the timing between the master Cyclone II device and serial configuration device is unaffected.

This configuration method supports both compressed and uncompressed SOFs. Therefore, if the configuration bitstream size exceeds the capacity of a serial configuration device, you can enable the compression feature in the SOF file used or you can select a larger serial configuration device.

Estimating AS Configuration Time

The AS configuration time is the time it takes to transfer data from the serial configuration device to the Cyclone II device. The Cyclone II DCLK output (generated from an internal oscillator) clocks this serial interface. As listed in Table 4–5, if you are using the 40-MHz oscillator, the DCLK minimum frequency is 20 MHz (50 ns). Therefore, the maximum configuration time estimate for an EP2C5 device (1,223,980 bits of uncompressed data) is:

RBF size × (maximum DCLK period / 1 bit per DCLK cycle) = estimated maximum configuration time

1,223,980 bits × (50 ns / 1 bit) = 61.2 ms

To estimate the typical configuration time, use the typical DCLK period listed in Table 4–5. With a typical DCLK period of 38.46 ns, the typical configuration time is 47.1 ms. Enabling compression reduces the amount of configuration data that is transmitted to the Cyclone II device, which also reduces configuration time. On average, compression reduces configuration time by 50%.

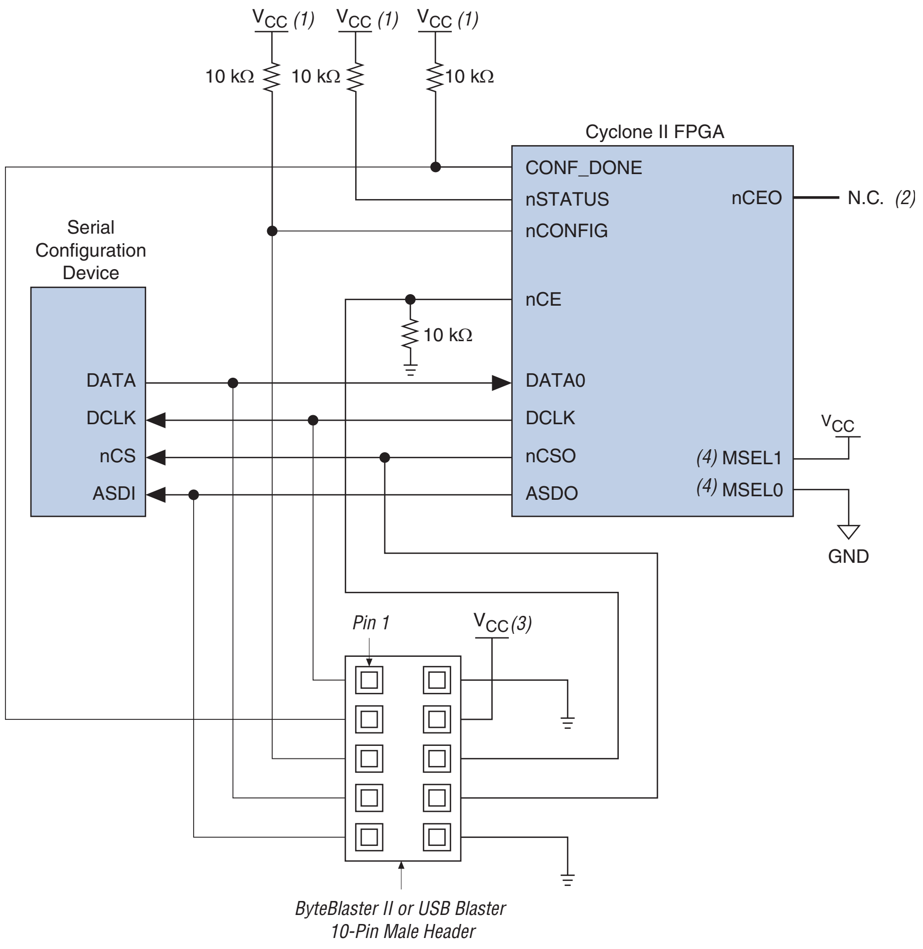
Programming Serial Configuration Devices

Serial configuration devices are non-volatile, flash-memory-based devices. You can program these devices in-system using the USB-Blaster™ or ByteBlaster™ II download cable. Alternatively, you can program them using the Altera Programming Unit (APU), supported third-party programmers, or a microprocessor with the SRunner software driver.

You can use the AS programming interface to program serial configuration devices in-system. During in-system programming, the download cable disables FPGA access to the AS interface by driving the `nCE` pin high. Cyclone II devices are also held in reset by pulling the `nCONFIG` signal low. After programming is complete, the download cable releases the `nCE` and `nCONFIG` signals, allowing the pull-down and pull-up resistor to drive GND and V_{CC}, respectively. Figure 4–7 shows the download cable connections to the serial configuration device.

For more information on the USB-Blaster download cable, see the *USB-Blaster USB Port Download Cable Data Sheet*. For more information on the ByteBlaster II cable, see the *ByteBlaster II Download Cable Data Sheet*.

Figure 4–7. In-System Programming of Serial Configuration Devices



Notes to Figure 4–7:

(1) Connect these pull-up resistors to 3.3-V supply.

(2) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

(3) Power up the ByteBlaster II or USB Blaster cable's V_{CC} with a 3.3-V supply.

(4) If using an EPCS4 or an EPCS1 device, set the `MSEL[1..0]` pins to 00. See Table 4–4 for more details.

You can use the Quartus II software with the APU and the appropriate configuration device programming adapter to program serial configuration devices. All serial configuration devices are offered in an 8-pin or 16-pin small outline integrated circuit (SOIC) package and can be programmed using the PLMSEPC-8 adapter.

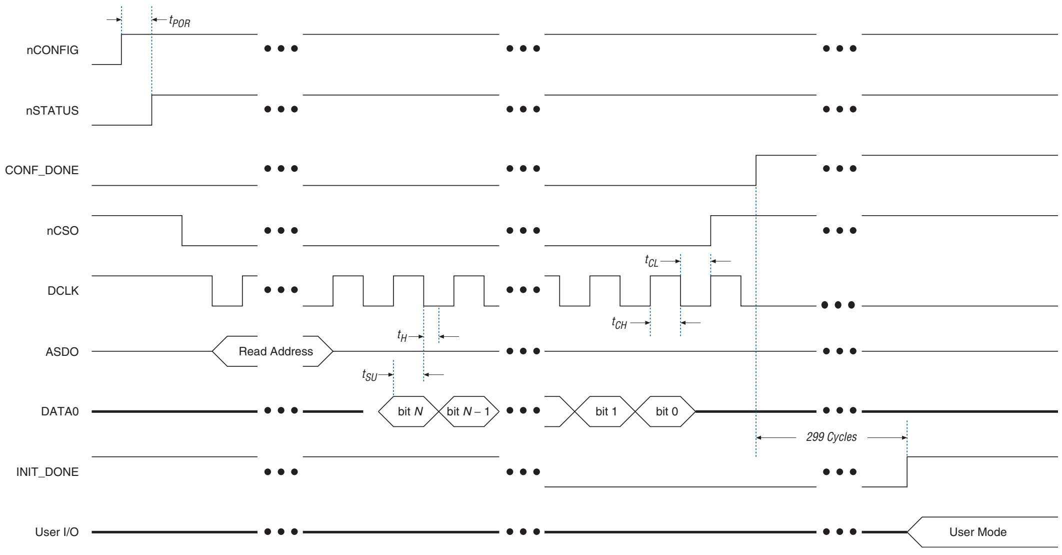
Altera programming hardware (APU) or other third-party programming hardware can be used to program blank serial configuration devices before they are mounted onto PCBs. Alternatively, you can use an on-board microprocessor to program the serial configuration device on the PCB using C-based software drivers provided by Altera (i.e., the SRunner software driver).

A 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, which can be easily customized to fit in different embedded systems. SRunner can read a Raw Programming Data File (**.rpd**) and write to the serial configuration devices. The serial configuration device programming time using SRunner is comparable to the programming time when using the Quartus II Programmer.

For more information about SRunner, see the *SRunner: An Embedded Solution for EPCS Programming White Paper* and the source code on the Altera web site at www.altera.com. For more information on programming serial configuration devices, see the *Serial Configuration Devices Data Sheet* in the *Configuration Handbook*.

Figure 4–8 shows the timing waveform for the AS configuration scheme using a serial configuration device.

Figure 4–8. AS Configuration Timing



PS Configuration

You can use an Altera configuration device, a download cable, or an intelligent host, such as a MAX® II device or microprocessor to configure a Cyclone II device with the PS scheme. In the PS scheme, an external host (configuration device, MAX II device, embedded processor, or host PC) controls configuration. Configuration data is input to the target Cyclone II devices via the `DATA0` pin at each rising edge of `DCLK`.

☞ The Cyclone II decompression feature is fully available when configuring your Cyclone II device using PS mode.

Table 4–6 shows the `MSEL` pin settings when using the PS configuration scheme.

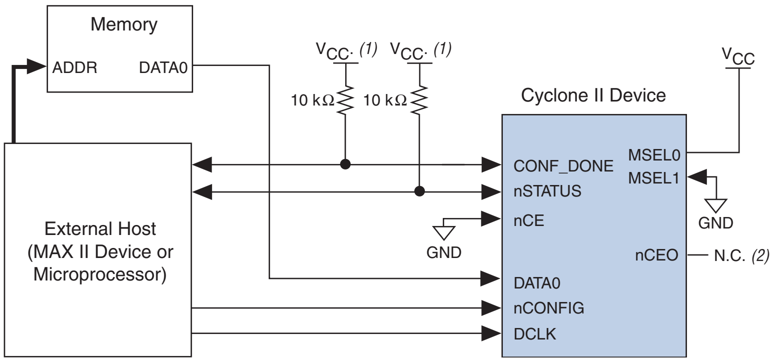
Table 4–6. Cyclone II MSEL Pin Settings for PS Configuration Schemes

Configuration Scheme	MSEL1	MSEL0
PS	0	1

Single Device PS Configuration Using a MAX II Device as an External Host

In the PS configuration scheme, you can use a MAX II device as an intelligent host that controls the transfer of configuration data from a storage device, such as flash memory, to the target Cyclone II device. Configuration data can be stored in RBF, HEX, or TTF format. Figure 4–9 shows the configuration interface connections between the Cyclone II device and a MAX II device for single device configuration.

Figure 4–9. Single Device PS Configuration Using an External Host



Notes to Figure 4–9:

(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device. V_{CC} should be high enough to meet the VIH specification of the I/O on the device and the external host.

(2) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

Upon power-up, the Cyclone II device goes through a POR, which lasts approximately 100 ms. During POR, the device will reset, hold `nSTATUS` low, and tri-state all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins continue to be tri-stated.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *Cyclone II Device Handbook*.

The configuration cycle consists of three stages: reset, configuration, and initialization.

Reset Stage

While the Cyclone II device's `nCONFIG` or `nSTATUS` pins are low, the device is in reset. To initiate configuration, the MAX II device must transition the Cyclone II `nCONFIG` pin from low to high.

V_{CCINT} and V_{CCIO} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When the Cyclone II `nCONFIG` pin transitions high, the Cyclone II device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` is released, the FPGA is ready to receive configuration data and the MAX II device can start the configuration at any time.

Configuration Stage

After the Cyclone II device's `nSTATUS` pin transitions high, the MAX II device should send the configuration data on the `DATA0` pin one bit at a time. If you are using configuration data in RBF, HEX, or TTF format, send the least significant bit (LSB) of each data byte first. For example, if the RBF contains the byte sequence 02 1B EE 01 FA, you should transmit the serial bitstream `0100-0000 1101-1000 0111-0111 1000-0000 0101-1111` to the device first.

The Cyclone II device receives configuration data on its `DATA0` pin and the clock on the `DCLK` pin. Data is latched into the FPGA on the rising edge of `DCLK`. Data is continuously clocked into the target device until the `CONF_DONE` pin transitions high. After the Cyclone II device receives all the configuration data successfully, it releases the open-drain `CONF_DONE` pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on `CONF_DONE` indicates configuration is complete and initialization of the device can begin.

The configuration clock (`DCLK`) speed must be below the specified system frequency (see Table 4–7) to ensure correct configuration. No maximum `DCLK` period exists, which means you can pause configuration by halting `DCLK` for an indefinite amount of time.

Initialization Stage

In Cyclone II devices, the initialization clock source is either the Cyclone II internal oscillator (typically 10 MHz) or the optional `CLKUSR` pin. The internal oscillator is the default clock source for initialization. If you use the internal oscillator, the Cyclone II device will make sure to provide enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. You do not need to provide additional clock cycles externally during the initialization stage. Driving `DCLK` back to the device after configuration is complete does not affect device operation. Additionally, if you use the internal oscillator as the clock source, you can use the `CLKUSR` pin as a user I/O pin.

If you want to delay the initialization of the device, you can use the `CLKUSR` pin. Using the `CLKUSR` pin allows you to control when your device enters user mode. You can delay the device from entering user mode for an indefinite amount of time.

The **Enable user-supplied start-up clock (CLKUSR)** option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on `CLKUSR` will not affect the configuration process. After all configuration data has been accepted and `CONF_DONE` goes high, Cyclone II devices require 299 clock cycles to initialize properly and support a `CLKUSR` f_{MAX} of 100 MHz.

If the optional `CLKUSR` pin is being used and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure that `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 µs).

An optional `INIT_DONE` pin signals the end of initialization and the start of user mode with a low-to-high transition. By default, the `INIT_DONE` output is disabled. You can enable the `INIT_DONE` output by turning on the **Enable INIT_DONE output** option in the Quartus II software. If you use the `INIT_DONE` pin, an external 10-kΩ pull-up resistor pulls the pin high when `nCONFIG` is low and during the beginning of configuration. Once the optional bit to enable `INIT_DONE` is programmed into the device (during the first frame of configuration data), the `INIT_DONE` pin transitions low. When initialization is complete, the `INIT_DONE` pin will be released and pulled high. The MAX II device must be able to detect this low-to-high transition, which signals the FPGA has entered user mode.

If you want to use the `INIT_DONE` pin as a user I/O pin, you should wait for the maximum value of t_{CD2UM} (see Table 4–7) after the `CONF_DONE` signal transitions high so to ensure the Cyclone II device has been initialized properly and is in user mode.

Make sure the MAX II device does not drive the `CONF_DONE` signal low during configuration, initialization, and before the device enters user mode.

User Mode

When initialization is complete, the Cyclone II device enters user mode. In user mode, the user I/O pins no longer have pull-up resistors and will function as assigned in your design.

To ensure `DCLK` and `DATA0` are not left floating at the end of configuration, the MAX II device must drive them either high or low, which ever is convenient on your PCB. The Cyclone II device `DATA0` pin is not available as a user I/O pin after configuration.

When the FPGA is in user mode, you can initiate a reconfiguration by transitioning the `nCONFIG` pin low-to-high. The `nCONFIG` pin must be low for at least 40 µs. When the `nCONFIG` transitions low, the Cyclone II device also pulls `nSTATUS` and `CONF_DONE` low and tri-states all I/O pins. Once the `nCONFIG` pin returns to a logic high level and the Cyclone II device releases the `nSTATUS` pin, the MAX II device can begin reconfiguration.

Error During Configuration

If an error occurs during configuration, the Cyclone II device transitions its `nSTATUS` pin low, resetting itself internally. The low signal on the `nSTATUS` pin tells the MAX II device that there is an error. If you turn on the **Auto-restart configuration after error** option in the Quartus II software, the Cyclone II device releases `nSTATUS` after a reset time-out period (maximum of 40 µs). After `nSTATUS` is released and pulled high by a pull-up resistor, the MAX II device can try to reconfigure the target device without needing to pulse `nCONFIG` low. If this option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on `nCONFIG` to restart the configuration process.

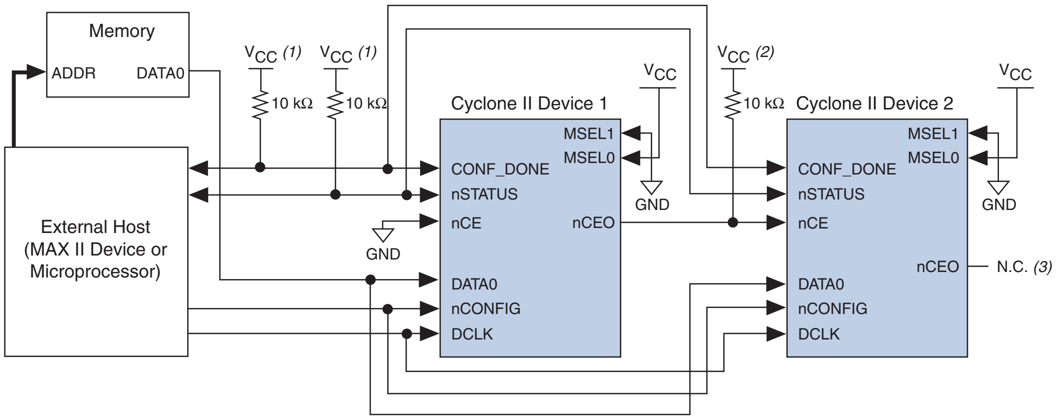
The MAX II device can also monitor the `CONF_DONE` and `INIT_DONE` pins to ensure successful configuration. The MAX II device must monitor the Cyclone II device's `CONF_DONE` pin to detect errors and determine when programming completes. If all configuration data is sent, but `CONF_DONE` or `INIT_DONE` do not transition high, the MAX II device must reconfigure the target device.

For more information on configuration issues, see the *Debugging Configuration Problems* chapter of the *Configuration Handbook* and the FPGA Configuration Troubleshooter on the Altera web site (**www.altera.com**).

Multiple Device PS Configuration Using a MAX II Device as an External Host

Figure 4–10 shows how to configure multiple devices using a MAX II device. This circuit is similar to the PS configuration circuit for a single device, except Cyclone II devices are cascaded for multiple device configuration.

Figure 4–10. Multiple Device PS Configuration Using an External Host



Notes to Figure 4–10:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the devices and the external host.

(2) Connect the pull-up resistor to the V_{CCIO} supply voltage of I/O bank that the `nCEO` pin resides in.

(3) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed another device's `nCE` pin.

In multiple device PS configuration, connect the first Cyclone II device's `nCE` pin to GND and connect the `nCEO` pin to the `nCE` pin of the next Cyclone II device in the chain. Use an external 10-kΩ pull-up resistor to pull the Cyclone II device's `nCEO` pin high to its V_{CCIO} level to help the internal weak pull-up resistor when the `nCEO` pin feeds next Cyclone II device's `nCE` pin. The input to the `nCE` pin of the last Cyclone II device in the chain comes from the previous Cyclone II device. After the first device completes configuration in a multiple device configuration chain, its `nCEO` pin transitions low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle. Therefore, the MAX II device begins to transfer data to the next Cyclone II device without interruption. The `nCEO` pin is a dual-purpose pin in Cyclone II devices. You can leave the `nCEO` pin of the last device unconnected or use it as a user I/O pin after configuration if the last device in chain is a Cyclone II device.

The Quartus II software sets the Cyclone II device `nCEO` pin as a dedicated output by default. If the `nCEO` pin feeds the next device's `nCE` pin, you must make sure that the `nCEO` pin is not used as a user I/O after configuration. This software setting is in the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box in Quartus II software.

You must connect all other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) to every Cyclone II device in the chain. The configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. You should buffer the `DCLK` and `DATA` lines for every fourth device. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

Since all `nSTATUS` and `CONF_DONE` pins are connected, if any Cyclone II device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first Cyclone II detects an error, it resets the chain by pulling its `nSTATUS` pin low. This behavior is similar to a single Cyclone II device detecting an error.

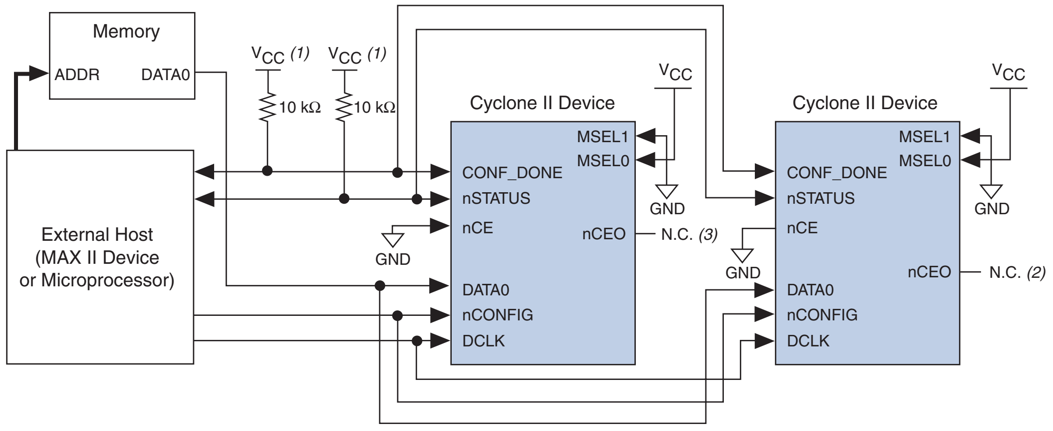
If the **Auto-restart configuration after error** option is turned on, the Cyclone II devices release their `nSTATUS` pins after a reset time-out period (maximum of 40 µs). After all `nSTATUS` pins are released and pulled high, the MAX II device reconfigures the chain without pulsing `nCONFIG` low. If the **Auto-restart configuration after error** option is turned off, the MAX II device must generate a low-to-high transition (with a low pulse of at least 40 µs) on `nCONFIG` to restart the configuration process.

If you want to delay the initialization of the devices in the chain, you can use the `CLKUSR` pin option. The `CLKUSR` pin allows you to control when your device enters user mode. This feature also allows you to control the order of when each device enters user mode by feeding a separate clock to each device's `CLKUSR` pin. By using the `CLKUSR` pins, you can choose any device in the multiple device chain to enter user mode first and have the other devices enter user mode at a later time.

Different device families may require a different number of initialization clock cycles. Therefore, if your multiple device chain consists of devices from different families, the devices may enter user mode at a slightly different time due to the different number of initialization clock cycles required. However, if the number of initialization clock cycles is similar across different device families or if the devices are from the same family, then the devices enter user mode at the same time. See the respective device family handbook for more information about the number of initialization clock cycles required.

If your system has multiple Cyclone II devices (in the same density and package) with the same configuration data, you can configure them in one configuration cycle by connecting all device's `nCE` pins to ground and connecting all the Cyclone II device's configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) together. You can also use the `nCEO` pin as a user I/O pin after configuration. The configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Make sure the `DCLK` and `DATA` lines are buffered for every fourth device. All devices will start and complete configuration at the same time. Figure 4–11 shows multiple device PS configuration when both Cyclone II devices are receiving the same configuration data.

Figure 4–11. Multiple Device PS Configuration When Both FPGAs Receive the Same Data



Notes to Figure 4–11:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain. V_{CC} should be high enough to meet the V_{IH} specification of the I/O on the devices and the external host.

(2) The `nCEO` pins of both devices can be left unconnected or used as user I/O pins when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Cyclone II devices with other Altera devices. Connect all the Cyclone II device's and all other Altera device's `CONF_DONE` and `nSTATUS` pins together so all devices in the chain complete configuration at the same time or that an error reported by one device initiates reconfiguration in all devices.

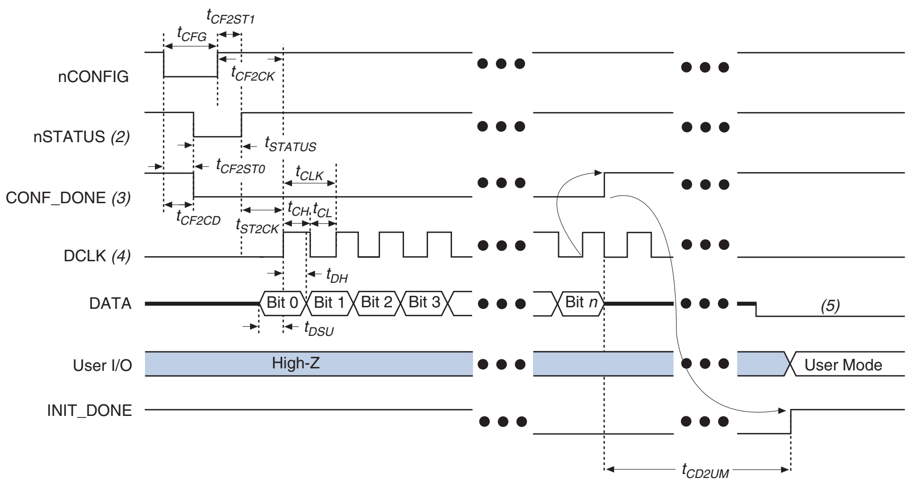
For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the *Configuration Handbook*.

PS Configuration Timing

A PS configuration must meet the setup and hold timing parameters and the maximum clock frequency. When using a microprocessor or another intelligent host to control the PS interface, ensure that you meet these timing requirements.

Figure 4–12 shows the timing waveform for PS configuration for Cyclone II devices.

Figure 4–12. PS Configuration Timing Waveform *Note (1)*



Notes to Figure 4–12:

(1) The beginning of this waveform shows the device in user mode. In user mode, `nCONFIG`, `nSTATUS` and `CONF_DONE` are at logic high levels. When `nCONFIG` is pulled low, a reconfiguration cycle begins.
(2) Upon power-up, the Cyclone II device holds `nSTATUS` low for the time of the POR delay.
(3) Upon power-up, before and during configuration, `CONF_DONE` is low.
(4) In user mode, drive `DCLK` either high or low when using the PS configuration scheme, whichever is more convenient. When using the AS configuration scheme, `DCLK` is a Cyclone II output pin and should not be driven externally.
(5) Do not leave the `DATA` pin floating after configuration. Drive it high or low, whichever is more convenient.

Table 4–7 defines the timing parameters for Cyclone II devices for PS configuration.

Table 4–7. PS Timing Parameters for Cyclone II Devices *Note (1)*

Symbol	Parameter	Minimum	Maximum	Units
t_{POR}	POR delay		100	ms
t_{CF2CD}	nCONFIG low to CONF_DONE low		800	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		800	ns
t_{CFG}	nCONFIG low pulse width	40		µs
t_{STATUS}	nSTATUS low pulse width	10	40 *(2)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		40 *(2)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	40		µs
t_{ST2CK}	nSTATUS high to first rising edge of DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	7		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK frequency		100	MHz
t_{CD2UM}	CONF_DONE high to user mode *(3)*	18	40	µs
t_{CD2CU}	CONF_DONE high to CLKUSR enabled	4 × maximum DCLK period		
t_{CD2UMC}	CONF_DONE high to user mode with CLKUSR option on	t_{CD2CU} + (299 × CLKUSR period)		

Notes to Table 4–7:

(1) This information is preliminary.

(2) This value is applicable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.

(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting the device.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* section in Volume 2 of the *Configuration Handbook*.

PS Configuration Using a Microprocessor

In the PS configuration scheme, a microprocessor can control the transfer of configuration data from a storage device, such as flash memory, to the target Cyclone II device.

All information in the "Single Device PS Configuration Using a MAX II Device as an External Host" on page 4–22 section is also applicable when using a microprocessor as an external host. Refer to that section for all configuration information.

The MicroBlaster™ software driver allows you to configure Altera FPGAs, including Cyclone II devices, through the ByteBlaster II or ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a RBF programming input file and is targeted for embedded PS configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems.

Since the Cyclone II device can decompress the compressed configuration data on-the-fly during PS configuration, the MicroBlaster software can accept a compressed RBF file as its input file.

For more information on the MicroBlaster software driver, see the *Configuring the MicroBlaster Passive Serial Software Driver White Paper* and source files on the Altera web site at **www.altera.com**.

If you turn on the **Enable user-supplied start-up clock (CLKUSR)** option in the Quartus II software, the Cyclone II devices will not enter user mode after the MicroBlaster has transmitted all the configuration data in the RBF file. You need to supply enough initialization clock cycles to `CLKUSR` pin to enter user mode.

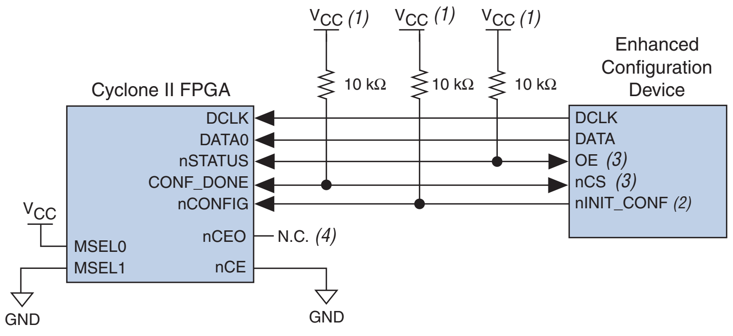
Single Device PS Configuration Using a Configuration Device

You can use an Altera configuration device (for example, an EPC2, EPC1, or enhanced configuration device) to configure Cyclone II devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 4–13 shows the configuration interface connections between the Cyclone II device and a configuration device.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on enhanced configuration devices and flash interface pins (e.g., `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`), see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*.

Figure 4–13. Single Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 4–13:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. This pull-up resistor is 10 kΩ.

(2) The nINIT_CONF pin is available on enhanced configuration devices and has an internal pull-up resistor that is always active, meaning an external pull-up resistor should not be used on the nINIT_CONF to 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) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet*.

When using enhanced configuration devices or EPC2 devices, you can connect the Cyclone II nCONFIG pin to the configuration device nINIT_CONF pin, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. You do not need to connect the nINIT_CONF pin if you are not using it. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), pull the nCONFIG signal to V_{CC} either directly or through a resistor. An internal pull-up resistor on the nINIT_CONF pin is always active in enhanced configuration devices and EPC2 devices. Therefore, you do not need an external pull-up if nCONFIG is connected to nINIT_CONF.

Upon power-up, the Cyclone II device goes through a POR. During POR, the device will reset, hold nSTATUS and CONF_DONE low, and tri-state all user I/O pins. After POR, which typically lasts 100 ms, the Cyclone II FPGA releases nSTATUS and enters configuration mode when this signal is pulled high by the external 10-kΩ resistor. Once the FPGA successfully exits POR, all user I/O pins continue to be tri-stated. Cyclone II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The configuration device also goes through a POR delay to allow the power supply to stabilize. The maximum POR time for EPC2 or EPC1 devices is 200 ms. The POR time for enhanced configuration devices can be set to 100 ms or 2 ms, depending on the enhanced configuration device's PORSEL pin setting. If the PORSEL pin is connected to ground, the POR delay is 100 ms. If the PORSEL pin is connected to V_{CC}, the POR delay is 2 ms. You must power the Cyclone II device before or during the enhanced configuration device POR time. During POR, the configuration device transitions its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When the target and configuration devices complete POR, they both release the nSTATUS to OE line, which is then pulled high by a pull-up resistor.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization.

☞ The Cyclone II device does not have a PORSEL pin.

Reset Stage

While nCONFIG or nSTATUS is low, the device is in reset. You can delay configuration by holding the nCONFIG or nSTATUS pin low.

☞ V_{CCINT} and V_{CCIO} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When the nCONFIG signal goes high, the device comes out of reset and releases the nSTATUS pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the OE pin. You can turn on this option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, you need to connect an external 10-kΩ pull-up resistor to the OE and nSTATUS line. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins.

Configuration Stage

When the nSTATUS pin transitions high, the configuration device's OE pin also transitions high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The Cyclone II device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK.

After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since the Cyclone II device's CONF_DONE pin is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the nCS pin. You can turn this option on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If you do not use this internal pull-up resistor, you need to connect an external 10-kΩ pull-up resistor to the nCS and CONF_DONE line. A low-to-high transition on CONF_DONE indicates configuration is complete, and the device can begin initialization.

Initialization Stage

In Cyclone II devices, the default initialization clock source is the Cyclone II internal oscillator (typically 10 MHz). Cyclone II devices can also use the optional CLKUSR pin. If your design uses the internal oscillator, the Cyclone II device will supply itself with enough clock cycles for proper initialization. The advantage of using the internal oscillator is you do not need to use another device or source to send additional clock cycles to the CLKUSR pin during the initialization stage. Additionally, you can use of the CLKUSR pin as a user I/O pin, which means you have an additional user I/O pin.

If you want to delay the initialization of the device, you can use the CLKUSR pin. Using the CLKUSR pin allows you to control when the Cyclone II device enters user mode. You can delay the Cyclone II devices from entering user mode for an indefinite amount of time. You can turn on the **Enable user-supplied start-up clock (CLKUSR)** option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data is accepted and CONF_DONE goes high, Cyclone II devices require 299 clock cycles to properly initialize and support a CLKUSR f_{MAX} of 100 MHz.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user mode with a low-to-high transition. The **Enable INIT_DONE output** option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If you use the INIT_DONE pin, an external 10-kΩ pull-up resistor pulls it high when nCONFIG is low and during the beginning of configuration. Once the optional bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. If you do not use the INIT_DONE pin, the initialization period will be complete after the CONF_DONE signal transitions high and 299 clock cycles are sent to the CLKUSR pin or after the time t_{CF2UM} (see Table 4–7) if the Cyclone II device uses the internal oscillator.

After successful configuration, if you intend to synchronize the initialization of multiple devices that are not in the same configuration chain, your system must not pull the CONF_DONE signal low to delay initialization. Instead, use the optional CLKUSR pin to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

User Mode

When initialization is complete, the FPGA enters user mode. In user mode, the user I/O pins do not have weak pull-up resistors and will function as assigned in your design. Enhanced configuration devices and EPC2 devices drive DCLK low and DATA0 high (EPC1 devices drive the DCLK pin low and tri-state the DATA pin) at the end of configuration.

When the FPGA is in user mode, pull the nCONFIG pin low to begin reconfiguration. The nCONFIG pin should be low for at least 40 µs. When nCONFIG transitions low, the Cyclone II device also pulls the nSTATUS and CONF_DONE pins low and all I/O pins are tri-stated. Since CONF_DONE transitions low, this will activate the configuration device since it will see its nCS pin transition low. Once nCONFIG returns to a logic high level and nSTATUS is released by the FPGA, reconfiguration begins.

Error During Configuration

If an error occurs during configuration, the Cyclone II drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If you turn on the **Auto-restart configuration after error** option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box, the FPGA automatically initiates reconfiguration if an error occurs. The Cyclone II device will release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 µs to restart configuration. The external system can pulse the nCONFIG pin if the pin is under system control rather than tied to V_{CC}.

Additionally, if the configuration device sends all of its data and then detects that the CONF_DONE pin has not transitioned high, it recognizes that the FPGA has not configured successfully. Enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit was sent for the CONF_DONE pin to transition high. EPC2 devices wait for 16 DCLK cycles. After that, the configuration device pulls its OE pin low, which in turn drives the target device's nSTATUS pin low. If you turn on the **Auto-restart configuration after error** option in the Quartus II software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS transitions high again, the configuration device reconfigures the FPGA.

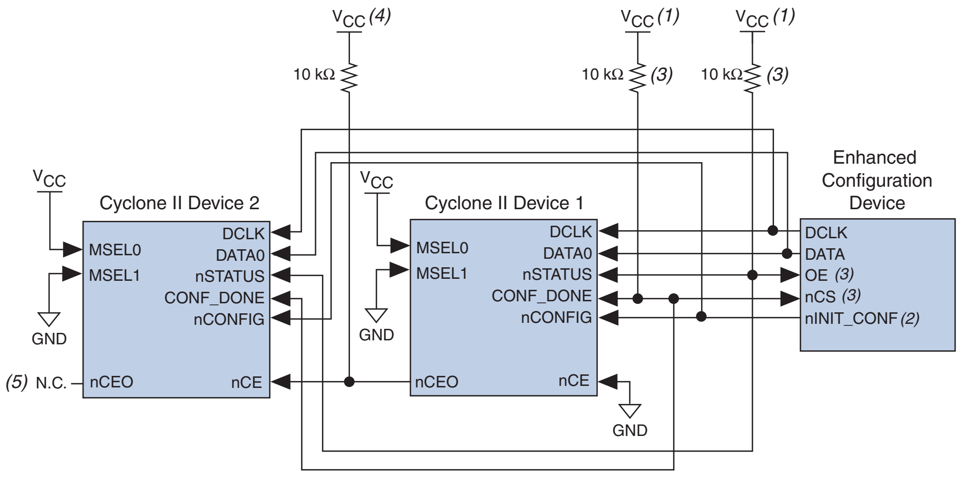
For more information on configuration issues, see the *Debugging Configuration Problems* chapter of the *Configuration Handbook* and the FPGA Configuration Troubleshooter on the Altera web site (**www.altera.com**).

Multiple Device PS Configuration Using a Configuration Device

You can use Altera enhanced configuration devices (EPC16, EPC8, and EPC4 devices) or EPC2 and EPC1 configuration devices to configure multiple Cyclone II devices in a PS configuration chain.

Figure 4–14 shows how to configure multiple devices with an enhanced configuration device. This circuit is similar to the configuration device circuit for a single device, except Cyclone II devices are cascaded for multiple device configuration.

Figure 4–14. Multiple Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 4–14:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used on the nINIT_CONF to 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) Connect the pull-up resistor to the V_{CCIO} supply voltage of I/O bank that the nCEO pin resides in.

(5) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.

You cannot cascade enhanced configuration devices (EPC16, EPC8, and EPC4 devices).

When configuring multiple devices, you must generate the configuration device's POF from each project's SOF. You can combine multiple SOFs using the **Convert Programming Files** window in the Quartus II software.

For more information on how to create configuration files for multiple device configuration chains, see the *Software Settings* section in Volume 2 of the *Configuration Handbook*.

When configuring multiple devices with the PS scheme, connect the first Cyclone II device's nCE pin to GND and connect its nCEO pin to the nCE pin of the Cyclone II device in the chain. Use an external 10-kΩ pull-up resistor to pull the Cyclone II device's nCEO pin to the V_{CCIO} level when it feeds the next device's nCE pin. After the first device in the chain completes configuration, its nCEO pin transitions low to activate the second device's nCE pin, which prompts the second device to begin configuration. You can leave the nCEO pin of the last device unconnected or use it as a user I/O pin after configuration. The nCEO pin is a dual-purpose pin in Cyclone II devices.

☞ The Quartus II software sets the Cyclone II device nCEO pin as an output pin driving to ground by default. If the device is in a chain, and the nCEO pin is connected to the next device's nCE pin, you must make sure that the nCEO pin is not used as a user I/O pin after configuration. This software setting is in the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box in Quartus II software.

Connect all other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) to every Cyclone II device in the chain. The configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Buffer the DCLK and DATA lines for every fourth device.

When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

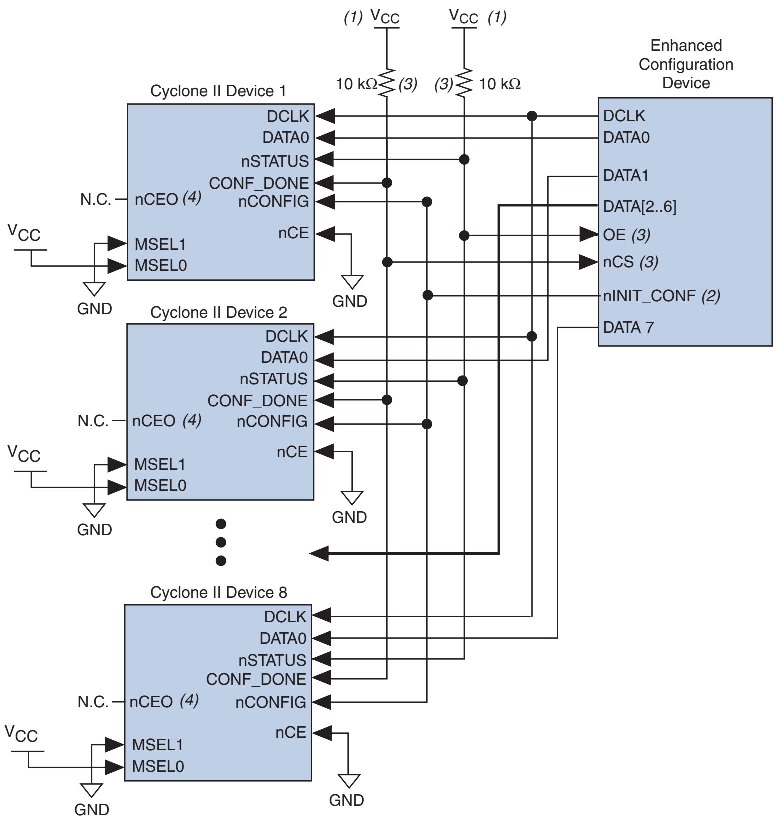
You should not pull CONF_DONE low to delay initialization. Instead, use the Quartus II software's **User-Supplied Start-Up Clock** option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain initialize together since their CONF_DONE pins are tied together.

Since all nSTATUS and CONF_DONE pins are connected, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if there is an error when configuring the first Cyclone II device, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the enhanced configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state.

If the **Auto-restart configuration after error** option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (40 µs maximum). When all the nSTATUS pins are released and pulled high, the configuration device reconfigures the chain. If the **Auto-restart configuration after error** option is turned off, a microprocessor or controller must monitor the nSTATUS pin for errors and then pulse nCONFIG low for at least 40 µs to restart configuration. The microprocessor or controller can only transition the nCONFIG pin low if the pin is under system control and not tied to V_{CC}.

The enhanced configuration devices support parallel configuration of up to eight devices. The *n*-bit (*n* = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure a chain of FPGAs. These devices do not have to be the same device family or density; they can be any combination of Altera FPGAs with different designs. An individual enhanced configuration device DATA pin is available for each targeted FPGA. Each DATA line can also feed a chain of FPGAs. Figure 4–15 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 4–15. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device



Notes to *Table 4–15:*

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used on the `nINIT_CONF` to `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) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

The Quartus II software only allows you to set *n* to 1, 2, 4, or 8. However, you can use these modes to configure any number of devices from 1 to 8. For example, if you configure three FPGAs, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding SOF data is transmitted from the configuration device to the FPGA. For

DATA3, you can leave the corresponding bit 3 line blank in the Quartus II software. On the printed circuit board (PCB), leave the DATA3 line from the enhanced configuration device unconnected. Use the Quartus II **Convert Programming Files** window (Tools menu) setup for this scheme.

You can also connect two FPGAs to one of the configuration device's DATA pins while the other DATA pins drive one device each. For example, you could use the 2-bit PS mode to drive two FPGAs with DATA bit 0 (two EP2C5 devices) and the third device (an EP2C8 device) with DATA bit 1. In this example, the memory space required for DATA bit 0 is the sum of the SOF file size for the two EP2C5 devices.

1,223,980 bits + 1,223,980 bits = 2,447,960 bits

The memory space required for DATA bit 1 is the SOF file size for on EP2C8 device (1,983,792 bits). Since the memory space required for DATA bit 0 is larger than the memory space required for DATA bit 1, the size of the POF file is 2 × 2,447,960 = 4,895,920.

For more information on using *n*-bit PS modes with enhanced configuration devices, see the *Using Altera Enhanced Configuration Devices* in the *Configuration Handbook*.

When configuring SRAM-based devices using *n*-bit PS modes, use Table 4–8 to select the appropriate configuration mode for the fastest configuration times.

Table 4–8. Recommended Configuration Using n-Bit PS Modes

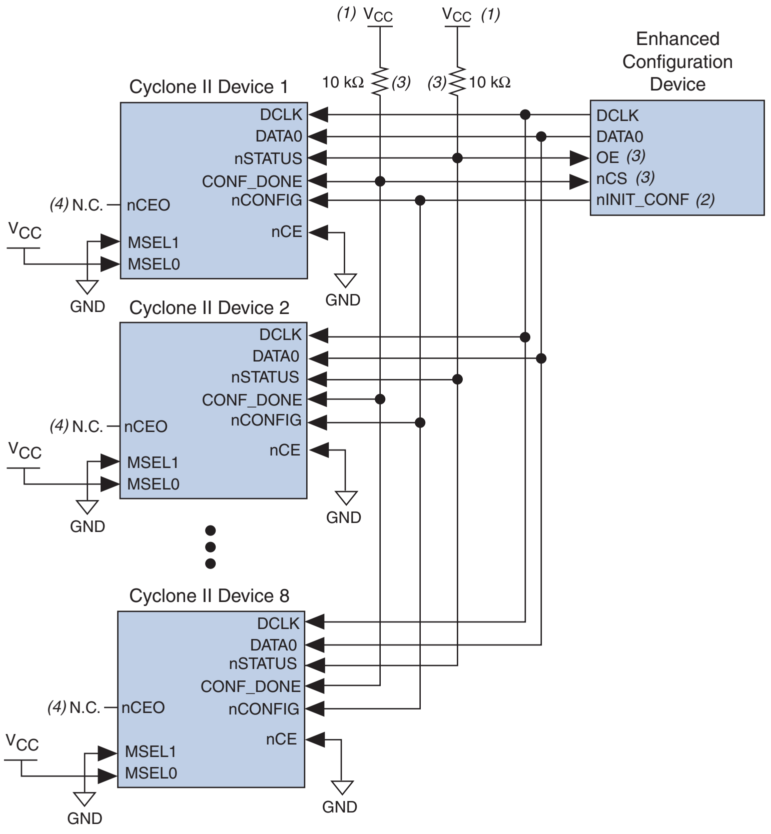
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 4–8:

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

If your design has multiple Cyclone II devices of the same density and package that contain the same configuration data, connect the `nCE` inputs to GND and leave the `nCEO` pins floating. You can also use the `nCEO` pin as a user I/O pin. Connect the configuration device `nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE` pins to each Cyclone II device in the chain. The configuration signals may require buffering to ensure signal integrity and prevent clock skew problems. Make sure that the `DCLK` and `DATA` lines are buffered for every fourth device. All devices will start and complete configuration at the same time. Figure 4–16 shows multiple device PS configuration when the Cyclone II devices are receiving the same configuration data.

Figure 4–16. Multiple Device PS Configuration Using an Enhanced Configuration Device When FPGAs Receive the Same Data



Notes to Figure 4–16:

(1) The pull-up resistor 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, meaning an external pull-up resistor should not be used on the `nINIT_CONF` to `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) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

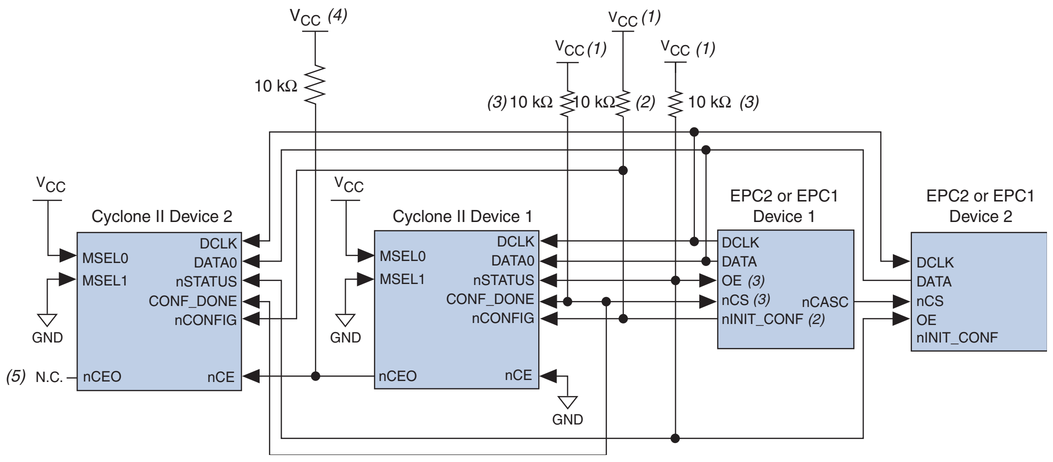
You can cascade several EPC2 or EPC1 devices to configure multiple Cyclone II devices. The first configuration device in the chain is the master configuration device, and the subsequent devices are the slave devices. The master configuration device sends `DCLK` to the Cyclone II devices and to the slave configuration devices. Connect the first configuration device's nCS pin to all the Cyclone II device's CONF_DONE pins, and connect the nCASC pin to the nCS pin of the next configuration device in the chain. Leave the nCASC pin of the last configuration device floating. When the master configuration device sends all the data to the Cyclone II device, the configuration device transitions the nCASC pin low, which drives nCS on the next configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

☞ Enhanced configuration devices (EPC16, EPC8, and EPC4 devices) cannot be cascaded.

Since all nSTATUS and CONF_DONE pins are connected, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect the Cyclone II device's CONF_DONE pin transitioning high at the end of configuration, it resets the entire chain by transitioning its OE pin low. This low signal drives the OE pin low on the slave configuration device(s) and drives nSTATUS low on all Cyclone II devices, causing them to enter a reset state. This behavior is similar to the FPGA detecting an error in the configuration data.

Figure 4–17 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 4–17. Multiple Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 4–17:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor should not be used on the `nINIT_CONF` to `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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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) Use an external 10-kΩ pull-up resistor to pull the `nCEO` pin high to the I/O bank V_{CCIO} level to help the internal weak pull-up when it feeds next device's `nCE` pin.

(5) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

When using enhanced configuration devices or EPC2 devices, you can connect the Cyclone II device's `nCONFIG` pin to the configuration device's `nINIT_CONF` pin, which allows the `INIT_CONF` JTAG instruction to initiate FPGA configuration. You do not need to connect the `nINIT_CONF` pin if it is not used. If the `nINIT_CONF` pin is not used or not available (for example, on EPC1 devices), pull the `nCONFIG` pin to V_{CC} levels either directly or through a resistor. An internal pull-up resistor on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices. Therefore, do not use an external pull-up resistor if you connect the `nCONFIG` pin to `nINIT_CONF`. If you use multiple EPC2 devices to configure a Cyclone II device(s), only connect the first EPC2 device's `nINIT_CONF` pin to the device's `nCONFIG` pin.

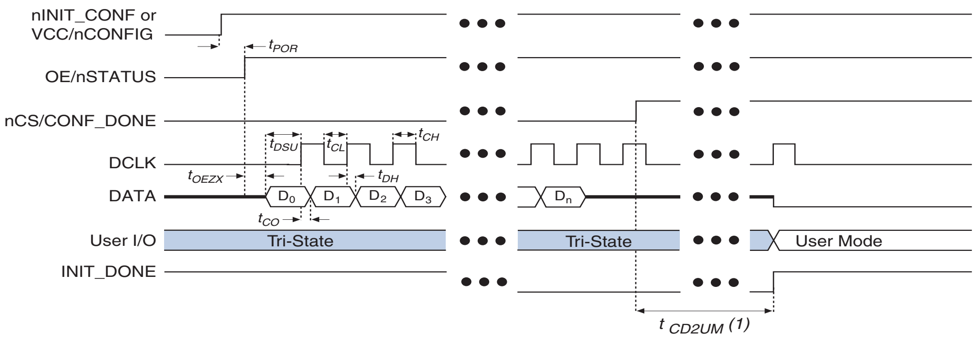
You can use a single configuration chain to configure Cyclone II devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, connect all the Cyclone II device `CONF_DONE` pins and connect all Cyclone II device `nSTATUS` pins together.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera FPGA Chains* chapter in the *Configuration Handbook*.

During PS configuration, the design must meet the setup and hold timing parameters and maximum DCLK frequency. The enhanced configuration and EPC2 devices are designed to meet these interface timing specifications.

Figure 4–18 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 4–18. Cyclone II PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 4–18:

(1) Cyclone II devices enter user mode 299 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the Cyclone II internal oscillator or the `CLKUSR` pin.

For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the *Configuration Handbook*.

For more information on device configuration options and how to create configuration files, see the *Software Settings* section in Volume 2 of the *Configuration Handbook*.

PS Configuration Using a Download Cable

In PS configuration, an intelligent host (e.g., a PC) can use a download cable to transfer data from a storage device to the Cyclone II device. You can use the Altera USB-Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster II parallel port download cable, or the ByteBlasterMV™ parallel port as a download cable.

Upon power up, the Cyclone II device goes through POR, which lasts approximately 100 ms. During POR, the device will reset, hold `nSTATUS` low, and tri-state all user I/O pins. Once the FPGA successfully exits POR, the `nSTATUS` pin is released and all user I/O pins continue to be tri-stated.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the *Cyclone II Device Handbook*.

The configuration cycle consists of three stages: reset, configuration, and initialization. While the `nCONFIG` or `nSTATUS` pins are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin.

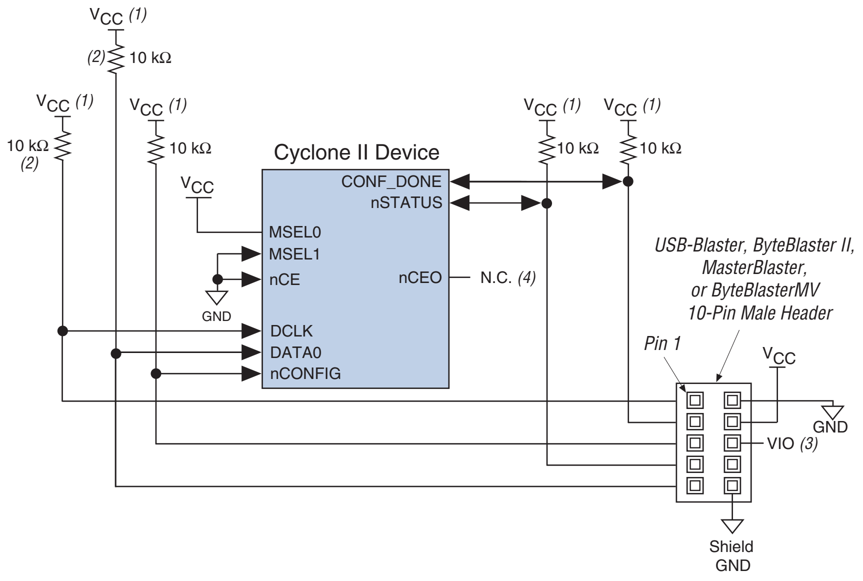
Make sure V_{CCINT} and V_{CCIO} for the banks where the configuration and JTAG pins reside are powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` transitions high, the Cyclone II device comes out of reset and begins configuration. The Cyclone II device releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` transitions high, the Cyclone II device is ready to receive configuration data. The programming hardware or download cable then transmits the configuration data one bit at a time to the device's `DATA0` pin. The configuration data is clocked into the target device until `CONF_DONE` goes high.

When using a download cable, you cannot use the **Auto-restart configuration after error** option. You must manually restart configuration in the Quartus II software when an error occurs. Additionally, you cannot use the **Enable user-supplied start-up clock (CLKUSR)** option when programming the FPGA using the Quartus II programmer and download cable. This option is disabled in the SOF. Therefore, if you turn on the `CLKUSR` option, you do not need to provide a clock on `CLKUSR` when you are configuring the FPGA with the

Quartus II programmer and a download cable. Figure 4–19 shows the PS configuration for Cyclone II devices using a USB-Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cable.

Figure 4–19. PS Configuration Using a USB-Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV Cable



Notes to Figure 4–19:

(1) The pull-up resistor should be connected to the same supply voltage as the USB-Blaster, MasterBlaster (VIO pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB-Blaster and ByteBlaster II, this pin is connected to nCE when it is used for AS programming, otherwise it is a no connect.

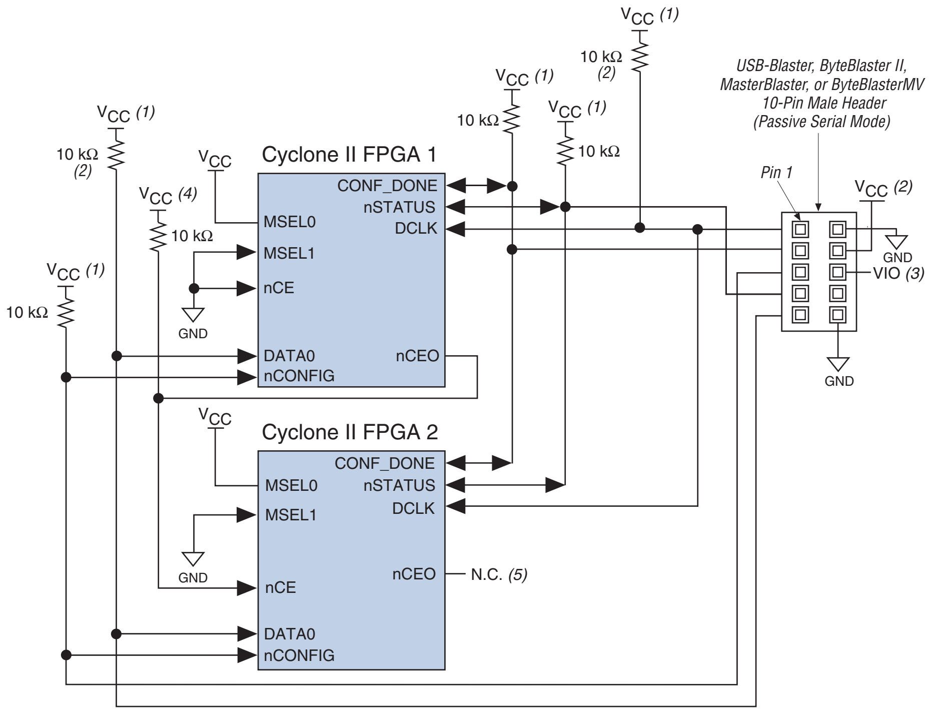
(4) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.

You can use a download cable to configure multiple Cyclone II devices by connecting each device's nCEO pin to the subsequent device's nCE pin. Connect the first Cyclone II device's nCE pin to GND and connect its nCEO pin to the nCE pin of the next device in the chain. Use an external 10-kΩ pull-up resistor to pull the nCEO pin high to V_{CCIO} when it feeds next device's nCE pin. Connect all other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) on every device in the chain together. Because all CONF_DONE pins are connected, all devices in the chain initialize and enter user mode at the same time.

In addition, because the nSTATUS pins are connected, all the Cyclone II devices in the chain stop configuration if any device detects an error. If this happens, you must manually restart configuration in the Quartus II software.

Figure 4–20 shows how to configure multiple Cyclone II devices with a download cable.

Figure 4–20. Multiple Device PS Configuration Using a USB-Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV Cable



Notes to Figure 4–20:

(1) The pull-up resistor should be connected to the same supply voltage as the USB-Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

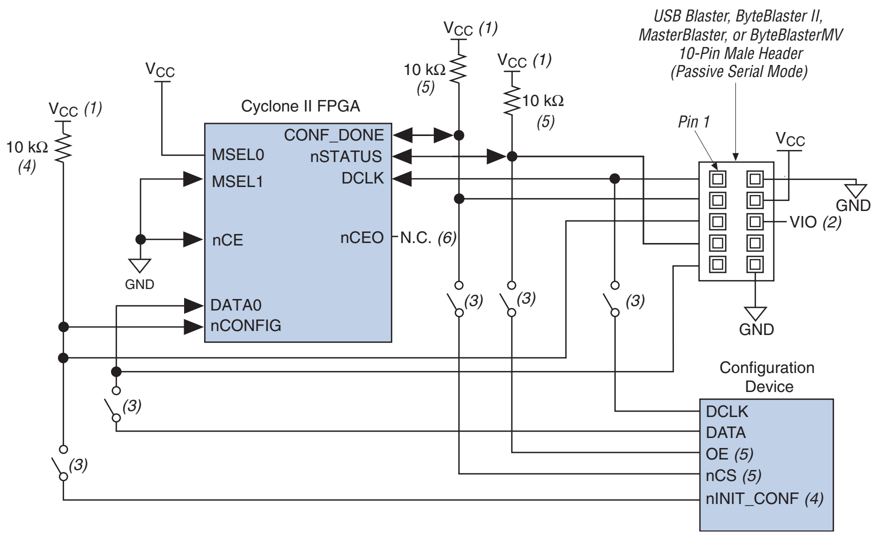
(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB-Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for AS programming, otherwise it is a no connect.

(4) Connect the pull-up resistor to the V_{CCIO} supply voltage of I/O bank that the `nCEO` pin resides in.

(5) The `nCEO` pin of the last device in chain can be left unconnected or used as a user I/O pin.

If you are using a download cable to configure Cyclone II devices on a PCB that also has configuration devices, you should electrically isolate the configuration devices from the target Cyclone II devices and cable. One way to isolate the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Additionally, you can add switches to the five common signals (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) between the cable and the configuration device. You can also remove the configuration device from the board when configuring the FPGA with the cable. Figure 4–21 shows a combination of a configuration device and a download cable to configure an FPGA.

Figure 4–21. PS Configuration with a Download Cable & Configuration Device Circuit



Notes to Figure 4–21:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB-Blaster and ByteBlaster II, this pin is connected to nCE when it is used for AS programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to a Cyclone II device. Instead, 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.

(4) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active. This means an external pull-up resistor should not be used on the nINIT_CONF to 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(5) 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.

(6) The nCEO pin can be left unconnected or used as a user I/O pin when it does not feed other device's nCE pin.

For more information on how to use the USB-Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cables, refer to the following documents:

- *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*

JTAG Configuration

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture allows you to test components on PCBs with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The JTAG circuitry can also be used to shift configuration data into the device. The Quartus II software automatically generates SOF files that can be used for JTAG configuration with a download cable in the Quartus II programmer.

For more information on JTAG boundary-scan testing, see the following documents:

- *IEEE 1149.1 (JTAG) Boundary-Scan Testing for Cyclone II Devices* chapter in Volume 2 of the *Cyclone II Device Handbook*
- Jam Programming & Testing Language Specification

Cyclone II devices are designed such that JTAG instructions have precedence over any device configuration modes. This means that JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of Cyclone II devices during PS configuration, PS configuration will terminate and JTAG configuration will begin. If the Cyclone II `MSEL` pins are set to AS or fast AS mode, the Cyclone II device will not output a `DCLK` signal when JTAG configuration takes place.

You cannot use the Cyclone II decompression feature if you are configuring your Cyclone II device when using JTAG-based configuration.

A device operating in JTAG mode uses the `TDI`, `TDO`, `TMS`, and `TCK` pins. The `TCK` pin has a weak internal pull-down resistor while the other JTAG input pins, `TDI` and `TMS`, have weak internal pull-up resistors. All user I/O pins are tri-stated during JTAG configuration. Table 4–9 explains each JTAG pin's function.

Table 4–9. Dedicated JTAG Pins

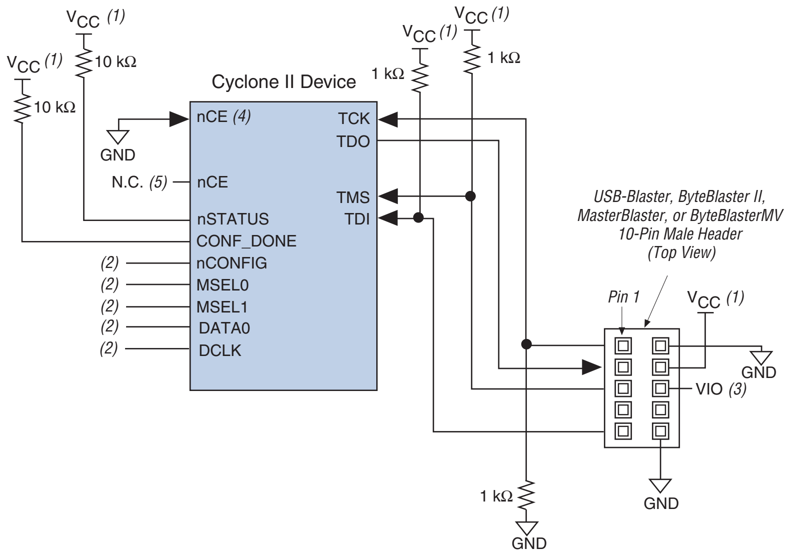
Pin Name	Pin Type	Description
`TDI`	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TDO`	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TCK`	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

☞ The `TDO` output is powered by the V_{CCIO} power supply. If V_{CCIO} is tied to 3.3-V, both the I/O pins and the JTAG `TDO` port drive at 3.3-V levels.

Single Device JTAG Configuration

During JTAG configuration, you can use the USB-Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV download cable to download data to the device. Configuring Cyclone II devices through a cable is similar to programming devices in system. Figure 4–22 shows JTAG configuration of a single Cyclone II device using a download cable.

Figure 4–22. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 4–22:

(1) The pull-up resistor should be connected to the same supply voltage as the USB-Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.
(2) Connect the `nCONFIG` and `MSEL[1..0]` pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect the `nCONFIG` pin to V_{CC}, and the `MSEL[1..0]` pins to ground. In addition, pull `DCLK` and `DATA0` to either high or low, whichever is convenient on your board.
(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB-Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for AS programming, otherwise it is a no connect.
(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.
(5) The `nCEO` pin can be left unconnected or used as a user I/O pin when it does not feed other device's `nCE` pin.

To configure a single device in a JTAG chain, the programming software places all other devices in BYPASS mode. In BYPASS mode, Cyclone II devices pass programming data from the TDI pin to the TDO pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the target device appears on the TDO pin one clock cycle later.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, the software checks the CONF_DONE pin through the JTAG port. When the Quartus II software generates a JAM file for a multiple device chain, it contains instructions so that all the devices in the chain will be initialized at the same time. If CONF_DONE is not high, the Quartus II software indicates that configuration has failed. If the CONF_DONE pin transitions high, the software indicates that configuration was successful. After the configuration bitstream is transmitted serially via the JTAG TDI port, the TCK port is clocked an additional 299 cycles to perform Cyclone II device initialization.

The **Enable user-supplied start-up clock (CLKUSR)** option has no affect on the device initialization since this option is disabled in the SOF when configuring the FPGA in JTAG using the Quartus II programmer and download cable. Therefore, if you turn on the CLKUSR option, you do not need to provide a clock on CLKUSR when you are configuring the FPGA with the Quartus II programmer and a download cable.

Cyclone II devices have dedicated JTAG pins that always function as JTAG pins. You can perform JTAG testing on Cyclone II devices before, after, and during configuration. Cyclone II devices support the BYPASS, IDCODE and SAMPLE instructions during configuration without interruption. All other JTAG instructions may only be issued by first interrupting configuration and reprogramming I/O pins using the CONFIG_IO instruction.

The CONFIG_IO instruction allows I/O buffers to be configured via the JTAG port. The CONFIG_IO instruction will interrupt configuration. This instruction allows you to perform board-level testing before configuring the Cyclone II device or waiting for a configuration device to complete configuration. If you interrupt configuration, the Cyclone II device must be reconfigured via JTAG (PULSE_CONFIG instruction) or by pulsing nCONFIG low after JTAG testing is complete.

For more information, see the *MorphIO: An I/O Reconfiguration Solution for Altera White Paper*.

The chip-wide reset (DEV_CLRn) and chip-wide output enable (DEV_OE) pins on Cyclone II devices do not affect JTAG boundary-scan or programming operations. Toggling these pins will not affect JTAG operations (other than the usual boundary-scan operation).

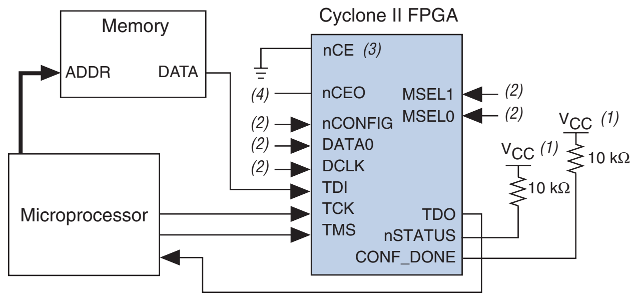
When designing a Cyclone II board for JTAG configuration, use the guidelines in Table 4–10 for the placement of the dedicated configuration pins.

Table 4–10. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all Cyclone II devices in the chain, nCE should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multiple device AS, or PS configuration chains, the nCE pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all Cyclone II devices in the chain, nCEO can be used as a user I/O or connected to the nCE of the next device. If nCEO is connected to the nCE of the next device, the nCEO pin must be pulled high to V_{CCIO} by an external 10-kΩ pull-up resistor to help the internal weak pull-up resistor. If the nCEO pin is not connected to the nCE pin of the next device, you can use it as a user I/O pin after configuration.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie these pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling up via a resistor, or driven high by some control circuitry.
nSTATUS	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each nSTATUS pin should be pulled up to V_{CC} individually. nSTATUS pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each CONF_DONE pin should be pulled up to V_{CC} individually. CONF_DONE going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.

Figure 4–23 shows JTAG configuration of a Cyclone II device with a microprocessor.

Figure 4–23. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 4–23:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

(2) Connect the `nCONFIG` and `MSEL[1..0]` pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect the `nCONFIG` pin to V_{CC}, and the `MSEL[1..0]` pins to ground. In addition, pull `DCLK` and `DATA0` to either high or low, whichever is convenient on your board.

(3) `nCE` must be connected to GND or driven low for successful JTAG configuration.

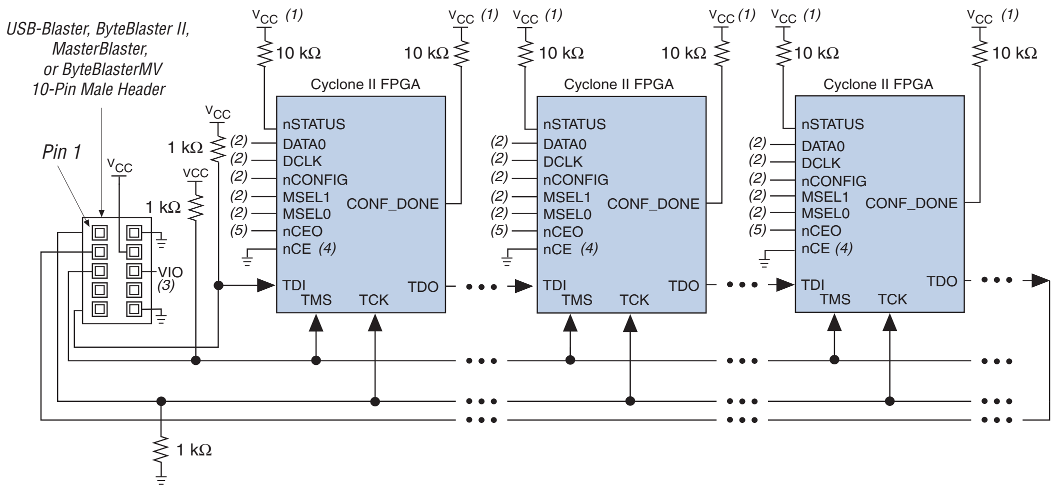
(4) If using an EPCS4 or EPCS1 device, set `MSEL[1..0]` to `00`. See Table 4–4 for more details.

JTAG Configuration of Multiple Devices

When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. When four or more devices are connected in a JTAG chain, Altera recommends buffering the TCK, TDI, and TMS pins with an on-board buffer.

JTAG-chain device programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 4–24 shows multiple device JTAG configuration.

Figure 4–24. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 4–24:

(1) The pull-up resistor should be connected to the same supply voltage as the USB-Blaster, MasterBlaster (VIO pin), ByteBlaster II or ByteBlasterMV cable.

(2) Connect the nCONFIG and MSEL[1..0] pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect the nCONFIG pin to V_{CC}, and the MSEL[1..0] pins to ground. In addition, pull DCLK and DATA0 to either high or low, whichever is convenient on your board.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB-Blaster and ByteBlaster II cable, this pin is connected to nCE when it is used for AS programming, otherwise it is a no connect.

(4) nCE must be connected to ground or driven low for successful JTAG configuration.

(5) If using an EPCS4 or EPCS1 device, set MSEL[1..0] to 00. See Table 4–4 for more details.

Connect the nCE pin to GND or pull it low during JTAG configuration. In multiple device AS and PS configuration chains, connect the first device's nCE pin to GND and connect its nCEO pin to the nCE pin of the next device in the chain or you can use it as a user I/O pin after configuration.

After the first device completes configuration in a multiple device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the nCE pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multiple device configuration chain, the nCEO pin of the previous device will drive the nCE pin of the next device low when it has successfully been JTAG configured.

The Quartus II software sets the Cyclone II device `nCEO` pin as an output pin driving to ground by default. If the `nCEO` pin inputs to the next device's `nCE` pin, make sure that the `nCEO` pin is not used as a user I/O pin after configuration.

Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera FPGA Chains* chapter in the *Configuration Handbook*.

Jam STAPL

Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP). Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard. The Jam player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, see *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the Jam player, go to the Altera web site (**www.altera.com**).

Configuring Cyclone II FPGAs with JRunner

JRunner is a software driver that allows you to configure Cyclone II devices through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in **.rbf** format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on your embedded platform.

The RBF file used by the JRunner software driver can not be a compressed RBF file because JRunner uses JTAG-based configuration. During JTAG-based configuration, the real-time decompression feature is not available.

For more information on the JRunner software driver, see *JRunner Software Driver: An Embedded Solution for PLD JTAG Configuration* and the source files on the Altera web site.

Programming Serial Configuration Devices In-System Using the JTAG Interface

Cyclone II devices in a single device chain or in a multiple device chain support in-system programming of a serial configuration device using the JTAG interface via the serial flash loader design. The board's intelligent host or download cable can use the four JTAG pins on the Cyclone II device to program the serial configuration device in system, even if the host or download cable cannot access the configuration device's configuration pins (`DCLK`, `DATA`, `ASDI`, and `nCS` pins).

The serial flash loader design is a JTAG-based in-system programming solution for Altera serial configuration devices. The serial flash loader is a bridge design for the FPGA that uses its JTAG interface to access the EPCS JIC (JTAG Indirect Configuration Device Programming) file and then uses the AS interface to program the EPCS device. Both the JTAG interface and AS interface are bridged together inside the serial flash loader design.

In a multiple device chain, you only need to configure the master Cyclone II device which is controlling the serial configuration device. The slave devices in the multiple device chain which are configured by the serial configuration device do not need to be configured when using this feature. To use this feature successfully, set the `MSEL[1..0]` pins of the master Cyclone II device to select the AS configuration scheme or fast AS configuration scheme (see Table 4–1).

☞ The Quartus II software version 4.1 and higher supports serial configuration device ISP through an FPGA JTAG interface using a JIC file.

The serial configuration device in-system programming through the Cyclone II JTAG interface has three stages, which are described in the following sections.

Loading the Serial Flash Loader Design

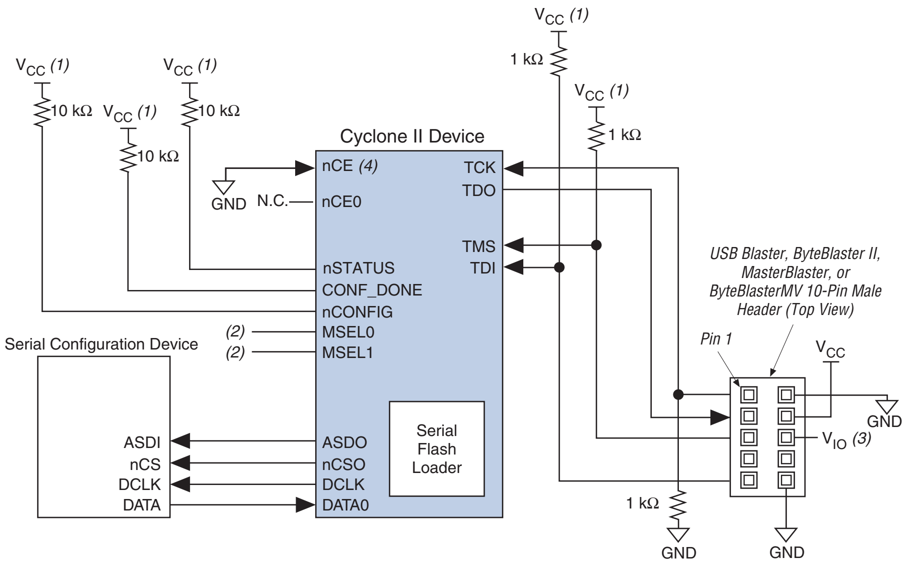
The serial flash loader design is a design inside the Cyclone II device that bridges the JTAG interface and AS interface inside the Cyclone II device using glue logic.

The intelligent host uses the JTAG interface to configure the master Cyclone II device with a serial flash loader design. The serial flash loader design allows the master Cyclone II device to control the access of four serial configuration device pins, also known as the Active Serial Memory

Interface (ASMI) pins, through the JTAG interface. The ASMI pins are the serial clock input (DCLK), serial data output (DATA), AS data input (ASDI), and an active-low chip select (nCS) pins.

If you configure a master Cyclone II device with a serial flash loader design, the master Cyclone II device can enter user mode even though the slave devices in the multiple device chain are not being configured. The master Cyclone II device can enter user mode with a serial flash loader design even though the CONF_DONE signal is externally held low by the other slave devices in chain. Figure 4–25 shows the JTAG configuration of a single Cyclone II device with a serial flash loader design.

Figure 4–25. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 4–25:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (V_{IO} pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The nCONFIG, MSEL[1..0] pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect nCONFIG to V_{CC}, and MSEL[3..0] to ground. Pull DCLK either high or low, whichever is convenient on your board.

(3) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cables, this pin is connected to nCE when it is used for active serial programming, otherwise it is a no connect.

(4) nCE must be connected to GND or driven low for successful JTAG configuration.

ISP of Serial Configuration Device

In the second stage, the serial flash loader design in the master Cyclone II device allows you to write the configuration data for the device chain into the serial configuration device by using the Cyclone II JTAG interface. The JTAG interface sends the programming data for the serial configuration device to the Cyclone II device first. The Cyclone II device then uses the ASMI pins to transmit the data to the serial configuration device.

Reconfiguration

After all the configuration data is written into the serial configuration device successfully, the intelligent host issues the `PULSE_NCONFIG` JTAG instruction to initialize the reconfiguration process. During reconfiguration, the master Cyclone II device will be reset and the serial flash loader design will no longer exist in the Cyclone II device and the serial configuration device will configure all the devices in the chain with your user design.

Device Configuration Pins

This section describes the connections and functionality of all the configuration related pins on the Cyclone II device. Table 4–11 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 4–11. Dedicated Configuration Pins on the Cyclone II Device (Part 1 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`MSEL[1..0]`	N/A	All	Input	This pin is a two-bit configuration input that sets the Cyclone II device configuration scheme. See Table 4–1 for the appropriate settings. You must connect these pins to V_{CCIO} or ground.
`nCONFIG`	N/A	All	Input	This pin is a configuration control input. If this pin is pulled low during user mode, the FPGA will lose its configuration data, enter a reset state, and tri-state all I/O pins. Transitioning this pin high initiates a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, you can connect the `nCONFIG` pin directly to V_{CC} or to the configuration device's `nINIT_CONF` pin.

Table 4–11. Dedicated Configuration Pins on the Cyclone II Device (Part 2 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nSTATUS	N/A	All	Bidirectional open-drain	The Cyclone II device drives nSTATUS low immediately after power-up and releases it after the POR time. This pin provides a status output and input for the Cyclone II device. If the Cyclone II device detects an error during configuration, it drives the nSTATUS pin low to stop configuration. If an external source (for example, another Cyclone II device) drives the nSTATUS pin low during configuration or initialization, the target device enters an error state. Driving nSTATUS low after configuration and initialization does not affect the configured device. If your design uses a configuration device, driving nSTATUS low causes the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on nSTATUS in user mode, the FPGA will not reconfigure. To initiate a reconfiguration, pull the nCONFIG pin low. The enhanced configuration devices' and EPC2 devices' OE and nCS pins are connected to the Cyclone II device's nSTATUS and CONF_DONE pins, respectively, and have optional internal programmable pull-up resistors. If you use these internal pull-up resistors on the enhanced configuration device, do not use external 10-kΩ pull-up resistors on these pins. When using EPC2 devices, you should only use external 10-kΩ pull-up resistors.

Table 4–11. Dedicated Configuration Pins on the Cyclone II Device (Part 3 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
CONF_DONE	N/A	All	Bidirectional open-drain	This pin is a status output and input. The target Cyclone II device drives the CONF_DONE pin low before and during configuration. Once the Cyclone II device receives all the configuration data without error and the initialization cycle starts, it releases CONF_DONE. Driving CONF_DONE low during user mode does not affect the configured device. Do not drive CONF_DONE low before the device enters user mode. After the Cyclone II device receives all the data, the CONF_DONE pin transitions high, and the device initializes and enters user mode. Driving CONF_DONE low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' OE and nCS pins are connected to the Cyclone II device's nSTATUS and CONF_DONE pins, respectively, and have optional internal programmable pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, external 10-kΩ pull-up resistors should not be used on these pins. When using EPC2 devices, you should only use external 10-kΩ pull-up resistors.
nCE	N/A	All	Input	This pin is an active-low chip enable. The nCE pin activates the device with a low signal to allow configuration. The nCE pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multiple device configuration, nCE of the first device is tied low while its nCEO pin is connected to nCE of the next device in the chain. The nCE pin must also be held low for successful JTAG programming of the FPGA.

Table 4–11. Dedicated Configuration Pins on the Cyclone II Device (Part 4 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCEO	N/A if option is on. I/O if option is off.	All	Output	This pin is an output that drives low when device configuration is complete. In single device configuration, you can leave this pin floating or use it as a user I/O pin after configuration. In multiple device configuration, this pin inputs the next device's nCE pin. The nCEO of the last device in the chain can be left floating or used as a user I/O pin after configuration. If you use the nCEO pin to feed next device's nCE pin, use an external 10-kΩ pull-up resistor to pull the nCEO pin high to the V_{CCIO} voltage of its I/O bank to help the internal weak pull-up resistor. Use the Quartus II software to make this pin a user I/O pin.
ASDO	N/A in AS mode I/O in PS and JTAG mode	AS	Output	This pin sends a control signal from the Cyclone II device to the serial configuration device in AS mode and is used to read out configuration data. In AS mode, ASDO has an internal pull-up that is always active.
nCSO	N/A in AS mode I/O in PS and JTAG mode	AS	Output	This pin sends an output control signal from the Cyclone II device to the serial configuration device in AS mode that enables the configuration device. In AS mode, nCSO has an internal pull-up resistor that is always active.

Table 4–11. Dedicated Configuration Pins on the Cyclone II Device (Part 5 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
DCLK	N/A	PS, AS	Input (PS) Output (AS)	In PS configuration, DCLK is the clock input used to clock data from an external source into the target device. Data is latched into the Cyclone II device on the rising edge of DCLK. In AS mode, DCLK is an output from the Cyclone II device that provides timing for the configuration interface. In AS mode, DCLK has an internal pull-up that is always active. After configuration, this pin is tri-stated. If you are using a configuration device, it drives DCLK low after configuration is complete. If your design uses a control host, drive DCLK either high or low. Toggling this pin after configuration does not affect the configured device.
DATA0	N/A	All	Input	This is the data input pin. In serial configuration modes, bit-wide configuration data is presented to the target device on the DATA0 pin. In AS mode, DATA0 has an internal pull-up resistor that is always active. After configuration, EPC1 and EPC1441 devices tri-state this pin, while enhanced configuration and EPC2 devices drive this pin high.

Table 4–12 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration, these pins function as user I/O pins and are tri-stated with weak pull-up resistors.

Table 4–12. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
`CLKUSR`	N/A if option is on. I/O if option is off.	Input	This is an optional user-supplied clock input that synchronizes the initialization of one or more devices. This pin is enabled by turning on the **Enable user-supplied start-up clock (CLKUSR)** option in the Quartus II software
`INIT_DONE`	N/A if option is on. I/O if option is off.	Output open-drain	This is a status pin that can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 10-kΩ pull-up resistor. 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 and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the **Enable INIT_DONE output** option in the Quartus II software.
`DEV_OE`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated. When this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the **Enable device-wide output enable (DEV_OE)** option in the Quartus II software.
`DEV_CLRn`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared. When this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the **Enable device-wide reset (DEV_CLRn)** option in the Quartus II software.

Table 4–13 describes the dedicated JTAG pins. JTAG pins must be kept stable before and during configuration to prevent accidental loading of JTAG instructions. The `TCK` pin has a weak internal pull-down resistor and the `TDI` and `TMS` JTAG input pins have weak internal pull-up resistors.

Table 4–13. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
`TDI`	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TDO`	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TCK`	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

Conclusion

Cyclone II devices can be configured in AS, PS or JTAG configuration schemes to fit your system's need. The AS configuration scheme supported by Cyclone II devices can now operate at a higher `DCLK` frequency (up to 40 MHz), which reduces your configuration time. In addition, Cyclone II devices can receive a compressed configuration bitstream and decompress this data on-the-fly in the AS or PS configuration scheme, which further reduces storage requirements and configuration time.

5. Configuring Cyclone FPGAs

C51013-1.3

Introduction

You can configure Cyclone™ FPGAs using one of several configuration schemes, including the active serial (AS) configuration scheme. This scheme is used with the low cost serial configuration devices. Passive serial (PS) and Joint Test Action Group (JTAG)-based configuration schemes are also supported by Cyclone FPGAs. Additionally, Cyclone FPGAs can receive a compressed configuration bit stream and decompress this data in real-time, reducing storage requirements and configuration time.

This chapter describes how to configure Cyclone devices using each of the three supported configuration schemes.

For more information on setting device configuration options or generating configuration files, see the *Software Settings* chapter in Volume II of the *Configuration Handbook*.

Device Configuration Overview

Cyclone FPGAs use SRAM cells to store configuration data. Since SRAM memory is volatile, configuration data must be downloaded to Cyclone FPGAs each time the device powers up. You can download configuration data to Cyclone FPGAs using the AS, PS, or JTAG interfaces (see Table 5–1).

Table 5–1. Cyclone FPGA Configuration Schemes

Configuration Scheme	Description
Active serial (AS) configuration	Configuration using: ● Serial configuration devices (EPCS1 or EPCS4)
Passive serial (PS) configuration	Configuration using: ● Enhanced configuration devices (EPC4, EPC8, and EPC16) ● EPC2, EPC1 configuration devices ● Intelligent host (microprocessor) ● Download cable
JTAG-based configuration	Configuration via JTAG pins using: ● Download cable ● Intelligent host (microprocessor) ● Jam™ Standard Test and Programming Language (STAPL) ● Ability to use SignalTap® II Embedded Logic Analyzer.

You can select a Cyclone FPGA configuration scheme by driving its `MSEL1` and `MSEL0` pins either high (`1`) or low (`0`), as shown in Table 5–2. If your application only requires a single configuration mode, the `MSEL` pins can be connected to V_{CC} (the I/O bank's V_{CCIO} voltage where the `MSEL` pin resides) or to ground. If your application requires more than one configuration mode, the `MSEL` pins can be switched after the FPGA has been configured successfully. Toggling these pins during user mode does not affect the device operation. However, the `MSEL` pins must be valid before initiating reconfiguration.

Table 5–2. Selecting Cyclone Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
`0`	`0`	AS
`0`	`1`	PS
`0`	`0` or `1` *(1)*	JTAG-based *(2)*

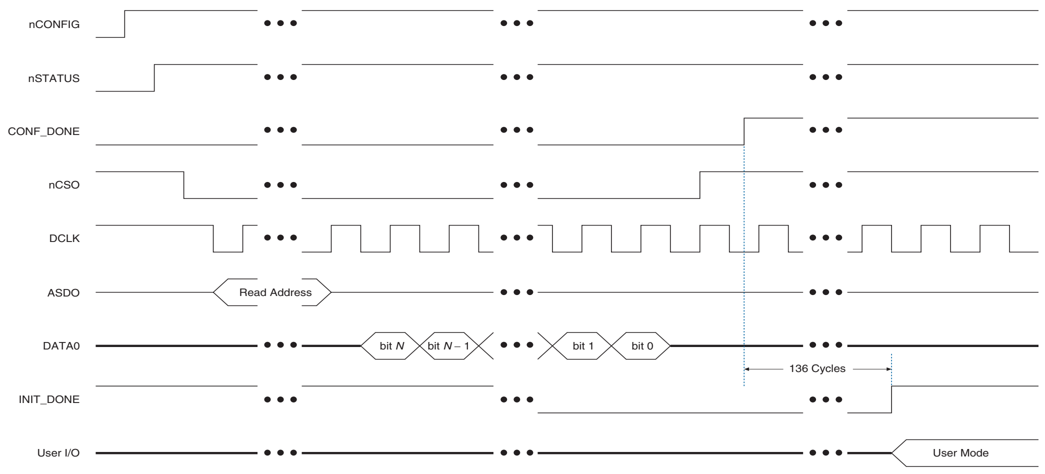
Notes to Table 5–2:

(1) Do not leave `MSEL` pins floating. Connect them to a low- or high-logic level. These pins support the non-JTAG configuration scheme used in production. If your design only uses JTAG configuration, you should connect the `MSEL0` pin to V_{CC}.

(2) JTAG-based configuration takes precedence over other schemes, which means that `MSEL` pin settings are ignored.

After configuration, Cyclone FPGAs will initialize registers and I/O pins, then enter user mode and function as per the user design. Figure 5–1 shows an AS configuration waveform.

Figure 5–1. AS Configuration Waveform



You can configure Cyclone FPGAs using the 3.3-, 2.5-, 1.8-, or 1.5-V LVTTL I/O standard on configuration and JTAG input pins. These devices do not feature a `VCCSEL` pin; therefore, you should connect the `VCCIO` pins of the I/O banks containing configuration or JTAG pins according to the I/O standard specifications.

Table 5–3 summarizes the approximate uncompressed configuration file size for each Cyclone FPGA. To calculate the amount of storage space required for multi-device configurations, add the file size of each device together.

Table 5–3. Cyclone Raw Binary File (.rbf) Sizes *Note (1)*

Device	Data Size (Bits)	Data Size (Bytes)
EP1C3	627,376	78,422
EP1C4	924,512	115,564
EP1C6	1,167,216	145,902
EP1C12	2,326,528	290,816
EP1C20	3,559,608	444,951

Note to Table 5–3:
(1) These values are preliminary.

You should only use the numbers in Table 5–3 to estimate the configuration file size before design compilation. Different file formats, such as **.hex** or **.ttf** files, have different file sizes. For any specific version of the Quartus® II software, any design targeted for the same device has the same uncompressed configuration file size. If compression is used, the file size can vary after each compilation.

Data Compression

Cyclone FPGAs are the first FPGAs to support decompression of configuration data. This feature allows you to store compressed configuration data in configuration devices or other memory, and transmit this compressed bit stream to Cyclone FPGAs. During configuration, the Cyclone FPGA decompresses the bit stream in real time and programs its SRAM cells.

Cyclone FPGAs support compression in the AS and PS configuration schemes. Compression is not supported for JTAG-based configuration.

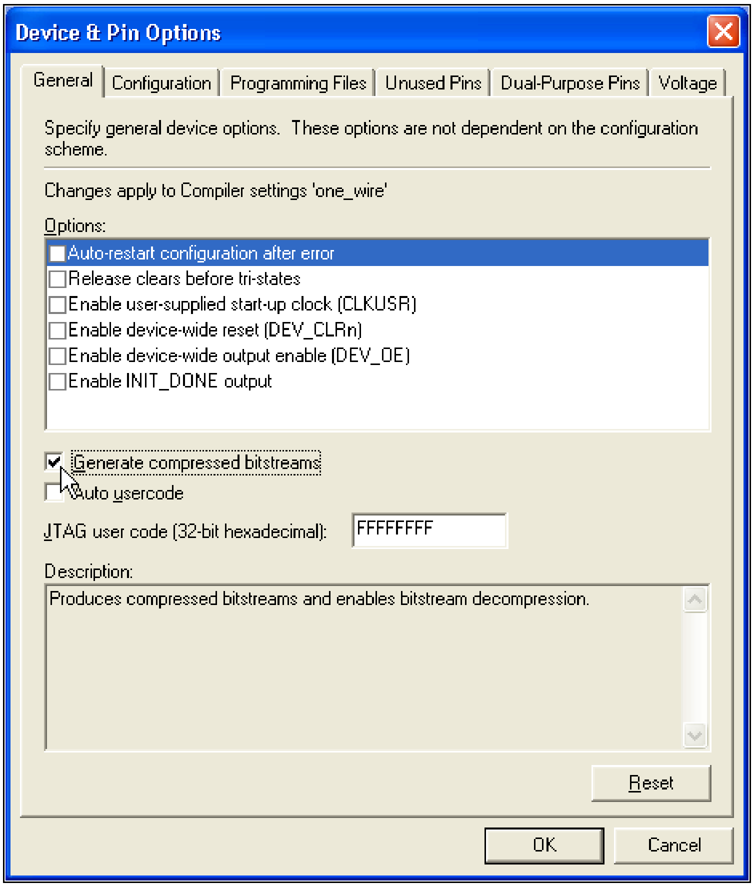
☞ Preliminary data indicates that compression reduces configuration bit stream size by 35 to 60%.

When you enable compression, the Quartus II software generates configuration files with compressed configuration data. This compression reduces the storage requirements in the configuration device or flash, and decreases the time needed to transmit the bit stream to the Cyclone FPGA.

There are two methods to enable compression for Cyclone bitstreams: before design compilation (in the Compiler Settings menu) and after design compilation (in the **Convert Programming Files** window).

To enable compression in the project's compiler settings, select **Device** under the Assignments menu to bring up the settings window. After selecting your Cyclone device open the **Device & Pin Options** window, and in the **General** settings tab enable the check box for **Generate compressed bitstreams** (as shown in Figure 5–2).

Figure 5–2. Enabling Compression for Cyclone Bitstreams in Compiler Settings



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

Changes apply to Compiler settings 'one_wire'

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:

Produces compressed bitstreams and enables bitstream decompression.

Compression can also be enabled when creating programming files from the **Convert Programming Files** window. See Figure 5–3.

1. Click **Convert Programming Files** (File menu).

2. Select the programming file type (POF, SRAM HEXOUT, RBF, or TTF).

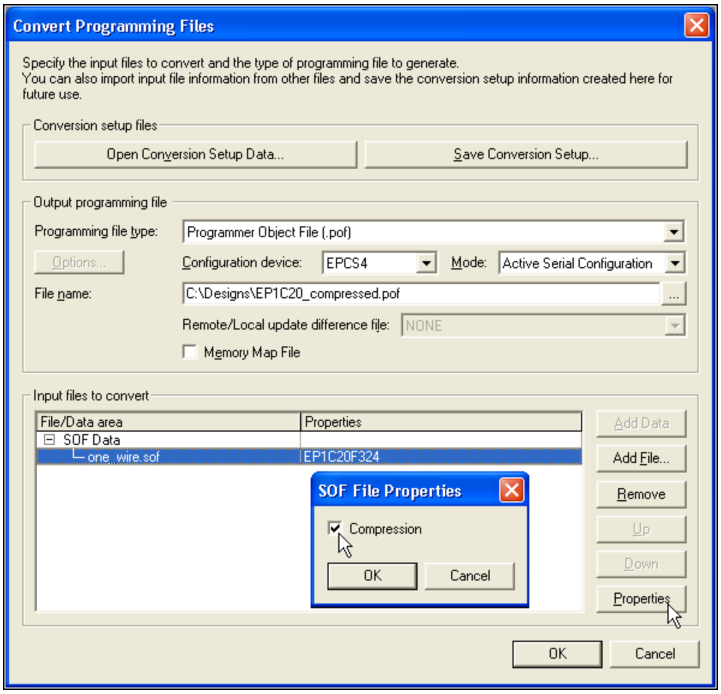
3. For POF output files, select a configuration device.

4. Select **Add File** and add a Cyclone SOF file(s).

5. Select the name of the file you added to the **SOF Data** area and click on **Properties**.

6. Check the **Compression** checkbox.

Figure 5–3. Enabling Compression for Cyclone Bitstreams in Convert Programming Files



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

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: Programmer Object File (.pof)

Options... | Configuration device: EPCS4 | Mode: Active Serial Configuration

File name: C:\Designs\EP1C20_compressed.pof

Remote/Local update difference file: NONE

☐ Memory Map File

Input files to convert

File/Data area	Properties
⊟ SOF Data	
└ one_wire.sof	EP1C20F324

Add Data | Add File... | Remove | Up | Down | Properties

SOF File Properties

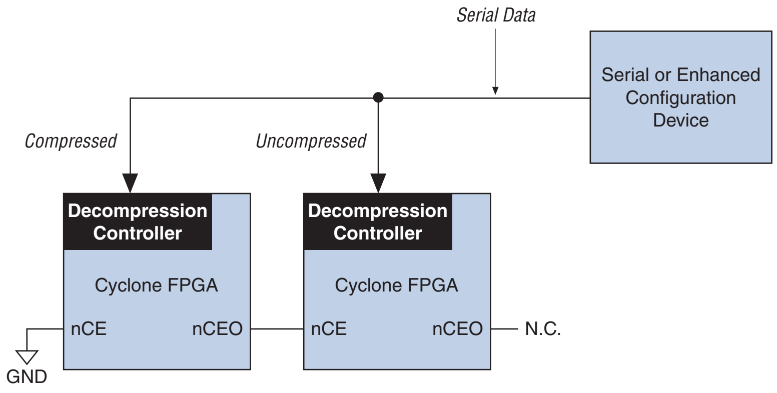
☑ Compression

OK | Cancel

OK | Cancel

When multiple Cyclone devices are cascaded, the compression feature can be selectively enabled for each device in the chain. Figure 5–4 depicts a chain of two Cyclone FPGAs. The first Cyclone FPGA has the compression feature enabled and therefore receives a compressed bit stream from the configuration device. The second Cyclone FPGA has the compression feature disabled and receives uncompressed data.

Figure 5–4. Compressed & Uncompressed Configuration Data in the Same Programming File *Note (1)*



Note to Figure 5–4:

(1) The first device in the chain should be set up in AS configuration mode (`MSEL[1..0]="00"`). The remaining devices in the chain must be set up in PS configuration mode (`MSEL[1..0]="01"`).

You can generate programming files for this setup from the **Convert Programming Files** window (File menu) in the Quartus II software.

The decompression feature supported by Cyclone FPGAs is separate from the decompression feature in enhanced configuration devices (EPC16, EPC8, and EPC4 devices). The data compression feature in the enhanced configuration devices allows them to store compressed data and decompress the bit stream before transmitting to the target devices. When using Cyclone FPGAs with enhanced configuration devices, Altera recommends using compression on one of the devices, not both (preferably the Cyclone FPGA since transmitting compressed data reduces configuration time).

Configuration Schemes

This section describes the various configuration schemes you can use to configure Cyclone FPGAs. Descriptions include an overview of the protocol, pin connections, and timing information. The schemes discussed are:

- AS configuration (serial configuration devices)
- PS configuration
- JTAG-based configuration

Active Serial Configuration (Serial Configuration Devices)

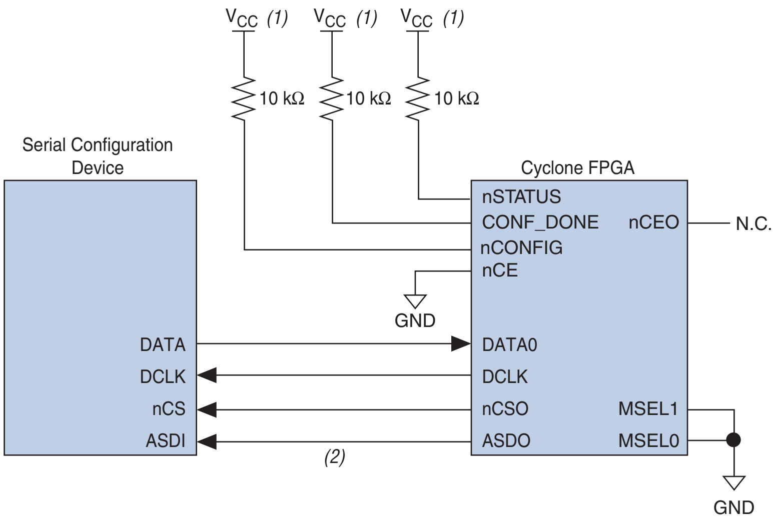
In the AS configuration scheme, Cyclone FPGAs are configured using the new serial configuration devices. These configuration devices are low cost devices with non-volatile memory that feature a simple four-pin interface and a small form factor. These features make serial configuration devices an ideal solution for configuring the low-cost Cyclone FPGAs.

For more information on serial configuration devices, see Chapter 4, Serial Configuration Devices (EPCS1, EPCS4, EPCS16 & EPCS64) Features.

Serial configuration devices provide a serial interface to access configuration data. During device configuration, Cyclone FPGAs read configuration data via the serial interface, decompress data if necessary, and configure their SRAM cells. This scheme is referred to as an AS configuration scheme because the FPGA controls the configuration interface. This scheme is in contrast to the PS configuration scheme where the configuration device controls the interface.

Serial configuration devices have a four-pin interface: serial clock input (`DCLK`), serial data output (`DATA`), AS data input (`ASDI`), and an active-low chip select (`nCS`). This four-pin interface connects to Cyclone FPGA pins as shown in Figure 5–5.

Figure 5–5. AS Configuration of a Single Cyclone FPGA



Notes to Figure 5–5:
(1) Connect the pull-up resistors to a 3.3-V supply.
(2) Cyclone FPGAs use the `ASDO` to `ASDI` path to control the configuration device.

Connecting the `MSEL[1..0]` pins to `00` selects the AS configuration scheme. The Cyclone chip enable signal, `nCE`, must also be connected to ground or driven low for successful configuration.

During system power up, both the Cyclone FPGA and serial configuration device enter a power-on reset (POR) period. As soon as the Cyclone FPGA enters POR, it drives `nSTATUS` low to indicate it is busy and drives `CONF_DONE` low to indicate that it has not been configured. After POR, which typically lasts 100 ms, the Cyclone FPGA releases `nSTATUS` and enters configuration mode when this signal is pulled high by the external 10-kΩ resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Cyclone devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in Chapter 4, DC & Switching Characteristics.

The serial clock (`DCLK`) generated by the Cyclone FPGA controls the entire configuration cycle (see Figure 5–1 on page 5–3) and this clock signal provides the timing for the serial interface. Cyclone FPGAs use an internal oscillator to generate DCLK. After configuration, this internal oscillator is turned off. Table 5–4 shows the active serial DCLK output frequencies.

Table 5–4. Active Serial DCLK Output Frequency

Minimum	Typical	Maximum	Units
14	17	20	MHz

The serial configuration device latches input/control signals on the rising edge of DCLK and drives out configuration data on the falling edge. Cyclone FPGAs drive out control signals on the falling edge of DCLK and latch configuration data on the rising edge of DCLK.

In configuration mode, the Cyclone FPGA enables the serial configuration device by driving its nCSO output pin low that is connected to the chip select (nCS) pin of the configuration device. The Cyclone FPGA's serial clock (DCLK) and serial data output (ASDO) pins send operation commands and read-address signals to the serial configuration device. The configuration device provides data on its serial data output (DATA) pin that is connected to the DATA0 input on Cyclone FPGAs.

After the Cyclone FPGA receives all configuration bits, it releases the open-drain CONF_DONE pin allowing the external 10-kΩ resistor to pull this signal to a high level. Initialization begins only after the CONF_DONE line reaches a high level.

You can select the clock used for initialization by using the **User Supplied Start-Up Clock** option in the Quartus II software. The Quartus II software uses the 10-MHz (typical) internal oscillator (separate from the AS internal oscillator) by default to initialize the Cyclone FPGA. After initialization, the internal oscillator is turned off. When you enable the **User Supplied Start-Up Clock** option, the software uses the CLKUSR pin as the initialization clock. Supplying a clock on the CLKUSR pin does not affect the configuration process. After all configuration data is accepted and the CONF_DONE signal goes high, Cyclone devices require 136 clock cycles to initialize properly.

An optional INIT_DONE pin is available. This pin signals the end of initialization and the start of user mode with a low-to-high transition. The **Enable INIT_DONE output** option is available in the Quartus II software. If the INIT_DONE pin is used, it is high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the

INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. In user mode, the user I/O pins do not have weak pull-ups and functions as assigned in your design.

If an error occurs during configuration, the Cyclone FPGA asserts the nSTATUS signal low indicating a data frame error, and the CONF_DONE signal stays low. With the **Auto-Restart Configuration on Frame Error** option enabled in the Quartus II software, the Cyclone FPGA resets the configuration device by pulsing nCSO, releases nSTATUS after a reset time-out period (about 30 μs), and retries configuration. If this option is turned off, the system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 40 μs to restart configuration. After successful configuration, the CONF_DONE signal is tri-stated by the target device and then pulled high by the pull-up resistor.

All AS configuration pins, DATA0, DCLK, nCSO, and ASDO, have weak internal pull-up resistors. These pull-up resistors are always active.

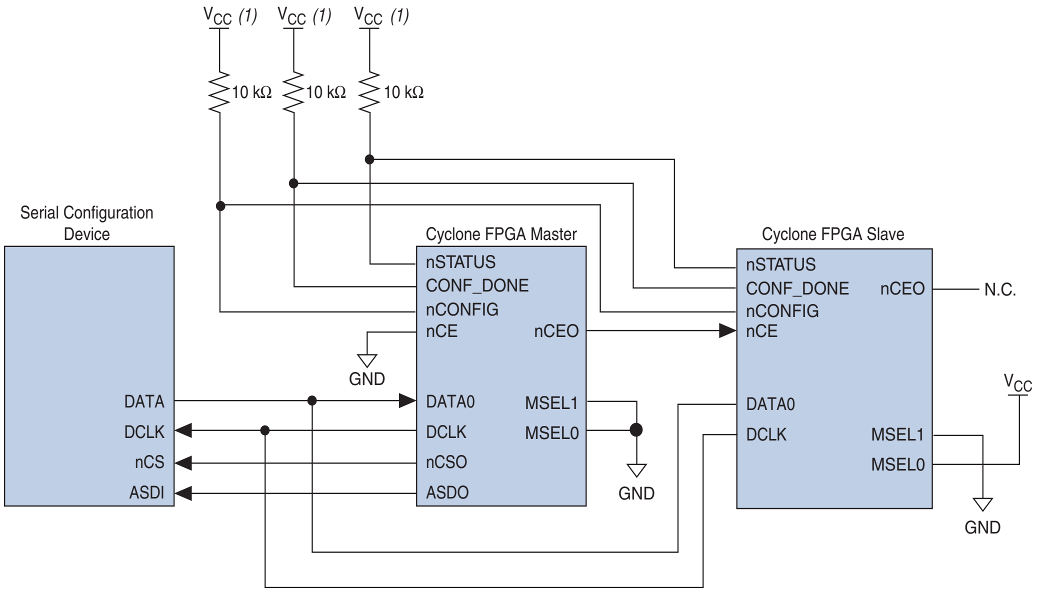
When the Cyclone FPGA is in user mode, you can initiate reconfiguration by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 40 μs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high level and nSTATUS is released by the Cyclone FPGA, reconfiguration begins.

Configuring Multiple Devices (Cascading)

You can configure multiple Cyclone FPGAs using a single serial configuration device. You can cascade multiple Cyclone FPGAs using the chip-enable (nCE) and chip-enable-out (nCEO) pins. The first device in the chain must have its nCE pin connected to ground. You must connect its nCEO pin to the nCE pin of the next device in the chain. When the first device captures all of its configuration data from the bit stream, it drives the nCEO pin low enabling the next device in the chain. You must leave the nCEO pin of the last device unconnected. The nCONFIG, nSTATUS, CONF_DONE, DCLK, and DATA0 pins of each device in the chain are connected (see Figure 5–6).

This first Cyclone FPGA in the chain is the configuration master and controls configuration of the entire chain. You must connect its MSEL pins to select the AS configuration scheme. The remaining Cyclone FPGAs are configuration slaves and you must connect their MSEL pins to select the PS configuration scheme. Figure 5–6 shows the pin connections for this setup.

Figure 5–6. Configuring Multiple Devices Using a Serial Configuration Device (AS)



Note to Figure 5–6:

(1) Connect the pull-up resistors to a 3.3-V supply.

As shown in Figure 5–6, the nSTATUS and CONF_DONE pins on all target FPGAs are connected together with external pull-up resistors. These pins are open-drain bidirectional pins on the FPGAs. When the first device asserts nCEO (after receiving all of its configuration data), it releases its CONF_DONE pin. But the subsequent devices in the chain keep this shared CONF_DONE line low until they have received their configuration data. When all target FPGAs in the chain have received their configuration data and have released CONF_DONE, the pull-up resistor drives a high level on this line and all devices simultaneously enter initialization mode. If an error occurs at any point during configuration, the nSTATUS line is driven low by the failing FPGA. If you enable the **Auto Restart Configuration on Frame Error** option, reconfiguration of the entire chain begins after a reset time-out period (a maximum of 40 µs). If the option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}.

While you can cascade Cyclone FPGAs, serial configuration devices cannot be cascaded or chained together.

If the configuration bit stream size exceeds the capacity of a serial configuration device, you must select a larger configuration device and/or enable the compression feature. While configuring multiple devices, the size of the bit stream is the sum of the individual devices' configuration bit streams.

Configuring Multiple Devices with the Same Data

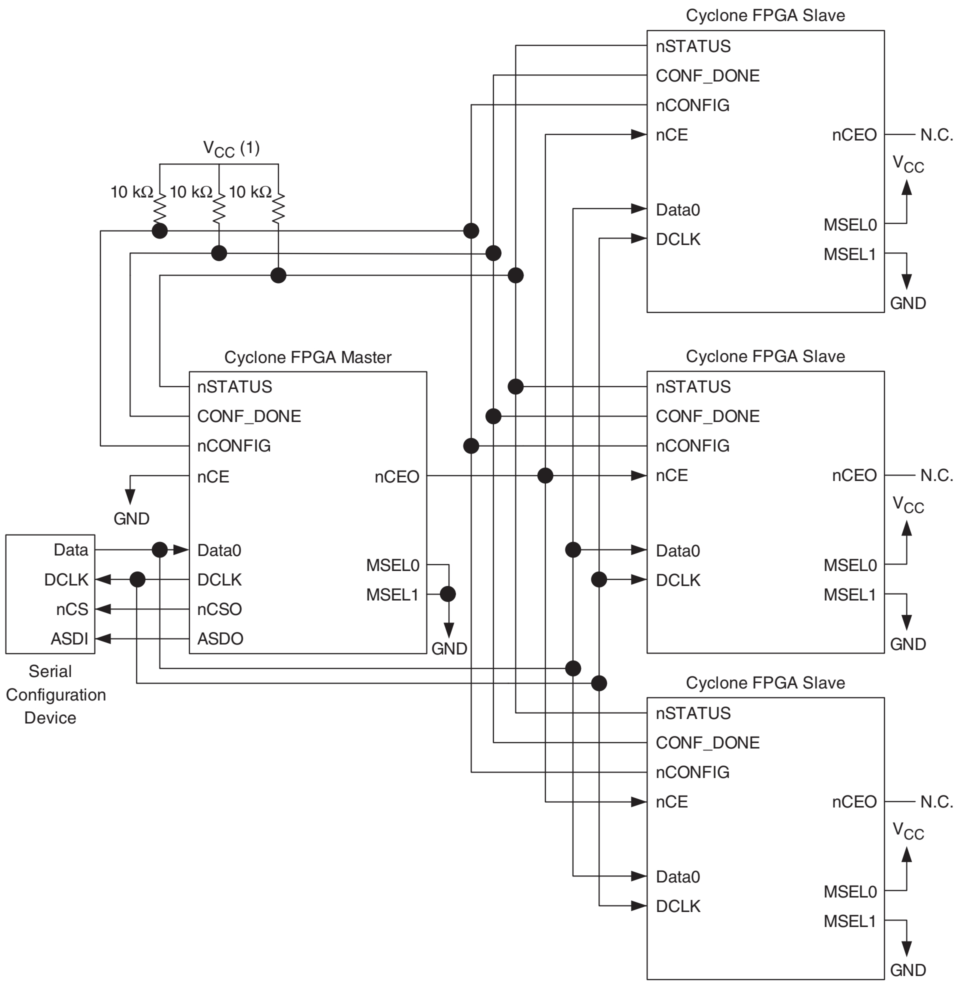
Certain applications require the configuration of multiple Cyclone devices with the same design through a configuration bit stream or SOF file. This can actually be done by two methods and they are shown below. For both methods, the serial configuration devices cannot be cascaded or chained together.

Method 1

For method 1, the serial configuration device stores two copies of the SOF file. The first copy configures the master Cyclone device, and the second copy configures all the remaining slave devices concurrently. The setup is similar to Figure 5–7 where the master is setup in AS mode (`MSEL=00`) and the slave devices are setup in PS mode (`MSEL01`).

To configure four identical Cyclone devices with the same SOF file, you could setup the chain similar to the example shown in Figure 5–6, except connect the three slave devices for concurrent configuration. The `nCEO` pin from the master device drives the `nCE` input pins on all three slave devices, and the `DATA` and `DCLK` pins connect in parallel to all four devices. During the first configuration cycle, the master device reads its configuration data from the serial configuration device while holding `nCEO` high. After completing its configuration cycle, the master drives `nCE` low and transmits the second copy of the configuration data to all three slave devices, configuring them simultaneously. The advantage of using the setup in Figure 5–7 is you can have a different SOF file for the Cyclone master device. However, all the Cyclone slave devices must be configured with the same SOF file.

Figure 5–7. Configuring Multiple Devices with the Same Design Using a Serial Configuration Device



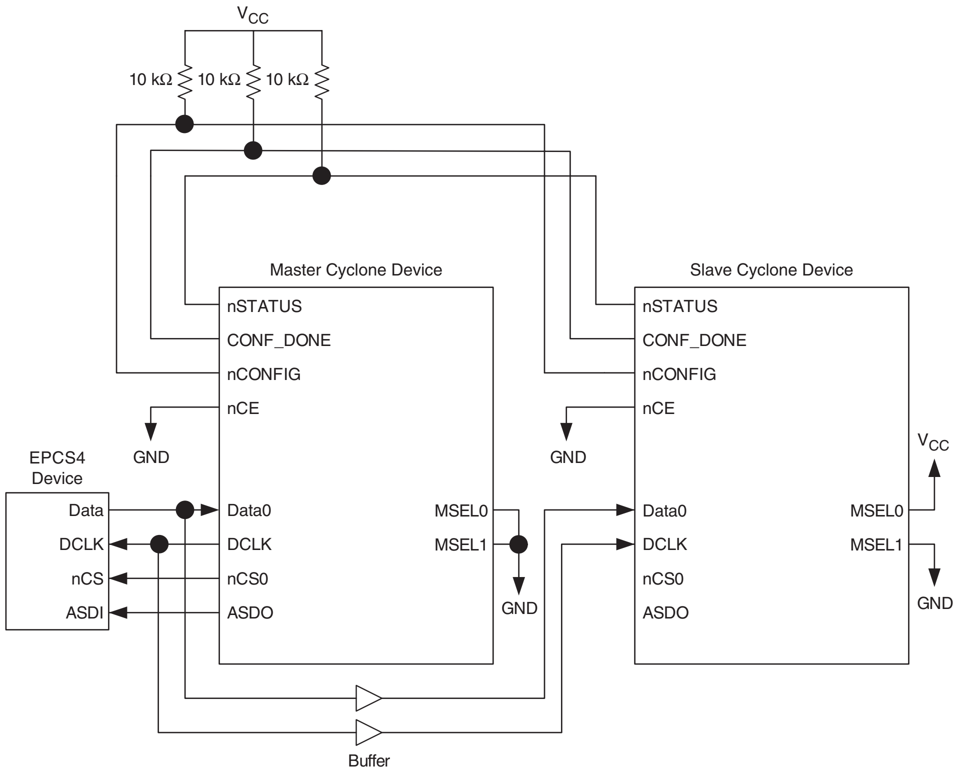
Note to Figure 5–7:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

Method 2

Method 2 configures multiple Cyclone devices with the same SOFs by storing only one copy of the SOF in the serial configuration device. This saves memory space in the serial configuration device for general-purpose use and may reduce costs. This method is shown in Figure 5–8 where the master device is set up in AS mode (`MSLE=00`), and the slave devices are set up in PS mode (MSEL=01). You could set up one or more slave devices in the chain and all the slave devices are set up in the same way as the design shown in Figure 5–8.

Figure 5–8. Configuring Multiple Devices with the Same Design Using a Serial Configuration Device



In this setup, all the Cyclone devices in the chain are connected for concurrent configuration. This reduces the active serial configuration time because all the Cyclone devices are configured in only one configuration cycle. To achieve this, the nCE input pins on all the Cyclone devices are connected to ground and the nCEO output pins on all the Cyclone devices are left unconnected. The DATA and DCLK pins connect in parallel to all the Cyclone devices.

It is recommended to add a buffer before the DATA and DCLK output from the master Cyclone to avoid signal strength and signal integrity issues. The buffer should not significantly change the DATA-to-DCLK relationships or delay them with respect to other ASMI signals, which are

ASDI and nCS signals. Also, the buffer should only drive the slave Cyclone devices, so that the timing between the master Cyclone device and serial configuration device is unaffected.

This setup can support both compressed and uncompressed SOFs. Therefore, if the configuration bit stream size exceeds the capacity of a serial configuration device, you can enable the compression feature on the SOF used or you can select a larger serial configuration device.

Estimating Active Serial Configuration Time

Active serial configuration time is dominated by the time it takes to transfer data from the serial configuration device to the Cyclone FPGA. This serial interface is clocked by the Cyclone DCLK output (generated from an internal oscillator). As listed in Table 5–4, the DCLK minimum frequency is 14 MHz (71 ns). Therefore, the maximum configuration time estimate for an EP1C3 device (0.628 MBits of uncompressed data) is:

(0.628 MBits × 71 ns) = 47 ms.

The typical configuration time is 33 ms.

Enabling compression reduces the amount of configuration data that is transmitted to the Cyclone device, reducing configuration time. On average, compression reduces configuration time by 50%.

Programming Serial Configuration Devices

Serial configuration devices are non-volatile, flash-memory-based devices. You can program these devices in-system using the ByteBlaster™ II download cable. Alternatively, you can program them using the Altera Programming Unit (APU) or supported third-party programmers.

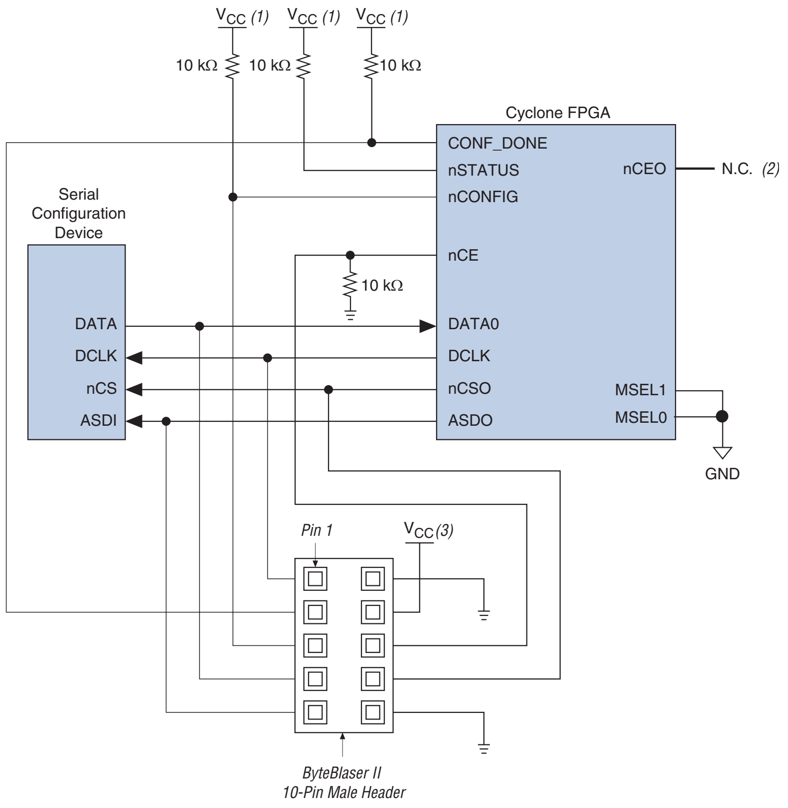
You can perform in-system programming of serial configuration devices via the AS programming interface. During in-system programming, the download cable disables FPGA access to the AS interface by driving the nCE pin high. Cyclone FPGAs are also held in reset by a low level on nCONFIG. After programming is complete, the download cable releases nCE and nCONFIG, allowing the pull-down and pull-up resistor to drive GND and VCC, respectively. Figure 5–9 shows the download cable connections to the serial configuration device.

For more information on the ByteBlaster II cable, see the *ByteBlaster II Download Cable Data Sheet*.

The serial configuration devices can be programmed in-system by an external microprocessor using SRunner. SRunner is a software driver developed for embedded serial configuration device programming that can be customized to fit in different embedded systems. The SRunner can read a Raw Programming Data file (**.rpd**) and write to the serial configuration devices. The programming time is comparable to the Quartus II software programming time.

For more information about SRunner, see the *"SRunner: An Embedded Solution for Serial Configuration Device Programming"* white paper and the source code on the Altera web site **(www.altera.com)**.

Figure 5–9. In-System Programming of Serial Configuration Devices



Notes to Figure 5–9:
(1) Connect these pull-up resistors to 3.3-V supply.
(2) The nCEO pin is left unconnected.
(3) Power up the ByteBlaster II cable's V_{CC} with a 3.3-V supply.

You can program serial configuration devices by using the Quartus II software with the APU and the appropriate configuration device programming adapter. All serial configuration devices are offered in an eight-pin small outline integrated circuit (SOIC) package and can be programmed using the PLMSEPC-8 adapter.

In production environments, serial configuration devices can be programmed using multiple methods. Altera programming hardware (APU) or other third-party programming hardware can be used to program blank serial configuration devices before they are mounted onto PCBs. Alternatively, you can use an on-board microprocessor to program the serial configuration device in-system using C-based software drivers provided by Altera.

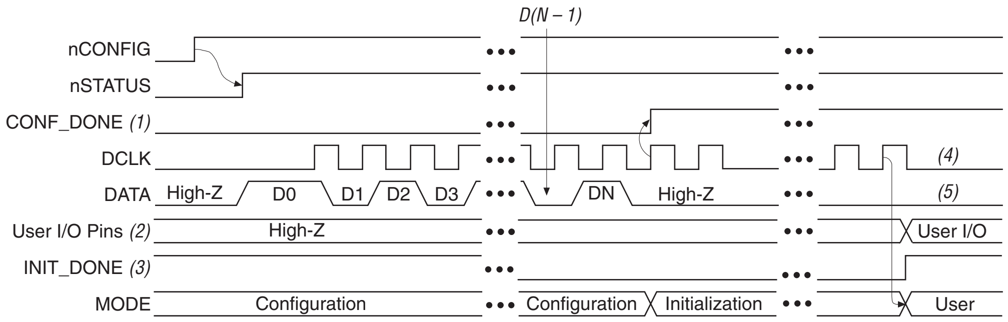
For more information on programming serial configuration devices, see the Cyclone Literature web page and the *Serial Configuration Devices (EPCS1 & EPCS4) Data Sheet*.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* chapter in Volume II of the *Configuration Handbook.*

Passive Serial Configuration

Cyclone FPGAs also feature the PS configuration scheme supported by all Altera FPGAs. In the PS scheme, an external host (configuration device, embedded processor, or host PC) controls configuration. Configuration data is clocked into the target Cyclone FPGAs via the `DATA0` pin at each rising edge of `DCLK`. The configuration waveforms for this scheme are shown in Figure 5–10.

Figure 5–10. PS Configuration Cycle Waveform



Notes to Figure 5–10:

(1) During initial power up and configuration, CONF_DONE is low. After configuration, CONF_DONE goes high to indicate successful configuration. If the device is reconfigured, CONF_DONE goes low after nCONFIG is driven low.

(2) User I/O pins are tri-stated during configuration. Cyclone FPGAs also have a weak pull-up resistor on I/O pins during configuration. After initialization, the user I/O pins perform the function assigned in the user's design.

(3) When used, the optional INIT_DONE signal is high when nCONFIG is low before configuration and during the first 136 clock cycles of configuration.

(4) In user mode, DCLK should be driven high or low when using the PS configuration scheme. When using the AS configuration scheme, DCLK is a Cyclone output pin and should not be driven externally.

(5) In user mode, DATA0 should be driven high or low.

PS Configuration Using Configuration Device

In the PS configuration device scheme, nCONFIG is usually tied to V_{CC} (when using EPC16, EPC8, EPC4, or EPC2 devices, you can connect nCONFIG to nINIT_CONF). Upon device power-up, the target Cyclone FPGA senses the low-to-high transition on nCONFIG and initiates configuration. The target device then drives the open-drain CONF_DONE pin low, which in-turn drives the configuration device's nCS pin low. When exiting POR, both the target and configuration device release the open-drain nSTATUS pin (typically Cyclone POR lasts 100 ms).

Before configuration begins, the configuration device goes through a POR delay of up to 100 ms (maximum) to allow the power supply to stabilize. You must power the Cyclone FPGA before or during the POR time of the enhanced configuration device. During POR, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When the target and configuration devices complete POR, they both release the nSTATUS to OE line, which is then pulled high by a pull-up resistor.

When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. When all devices are ready, the configuration device clocks out DATA and DCLK to the target devices using an internal oscillator.

After successful configuration, the Cyclone FPGA starts initialization using the 10-MHz internal oscillator as the reference clock. After initialization, this internal oscillator is turned off. The CONF_DONE pin is released by the target device and then pulled high by a pull-up resistor. When initialization is complete, the target Cyclone FPGA enters user mode.

If an error occurs during configuration, the target device drives its nSTATUS pin low, resetting itself internally and resetting the configuration device. If you turn on the **Auto-Restart Configuration on Frame Error** option, the device reconfigures automatically if an error occurs. To set this option, select **Compiler Settings** (Processing menu), and click on the **Chips & Devices** tab. Select **Device & Pin Options**, and click on the **Configuration** tab.

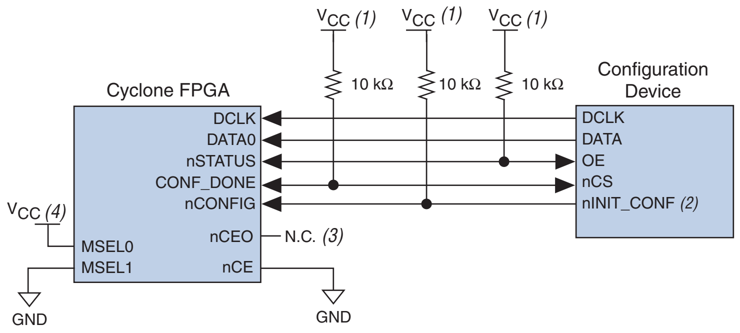
If the **Auto-Restart Configuration on Frame Error** option is turned off, the external system (configuration device or microprocessor) must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if it is under system control rather than tied to V_{CC}. When configuration is complete, the target device releases CONF_DONE, which disables the configuration device by driving nCS high. The configuration device drives DCLK low before and after configuration.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the target device has not configured successfully. (For CONF_DONE to reach a high state, enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit. EPC2 devices wait for 16 DCLK cycles.) In this case, the configuration device pulses its OE pin low for a few microseconds, driving the target device's nSTATUS pin low. If the **Auto-Restart Configuration on Frame Error** option is set in the Quartus II software, the target device resets and then releases its nSTATUS pin after a reset time-out period. When nSTATUS returns high, the configuration device reconfigures the target device.

You should not pull CONF_DONE low to delay initialization. Instead, use the Quartus II software's **User-Supplied Start-Up Clock** option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain initialize together since their CONF_DONE pins are tied together.

CONF_DONE goes high during the first few clock cycles of initialization. Hence, when using the CLKUSR feature you would not see the CONF_DONE signal high until you start clocking CLKUSR. However, the device does retain configuration data and waits for these initialization clocks to release CONF_DONE and go into user mode. Figure 5–11 shows how to configure one Cyclone FPGA with one configuration device.

Figure 5–11. Single Device Configuration Circuit



Notes to Figure 5–11:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. This pull-up resistor is 10 kΩ. The EPC16, EPC8, EPC4, and EPC2 devices' OE and nCS pins have internal, user-configurable pull-up resistors. If you use internal pull-up resistors, do not use external pull-up resistors on these pins.

(2) The nINIT_CONF pin is available on EPC16, EPC8, EPC4, and EPC2 devices and has an internal pull-up resistor that is always active. If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} through a resistor.

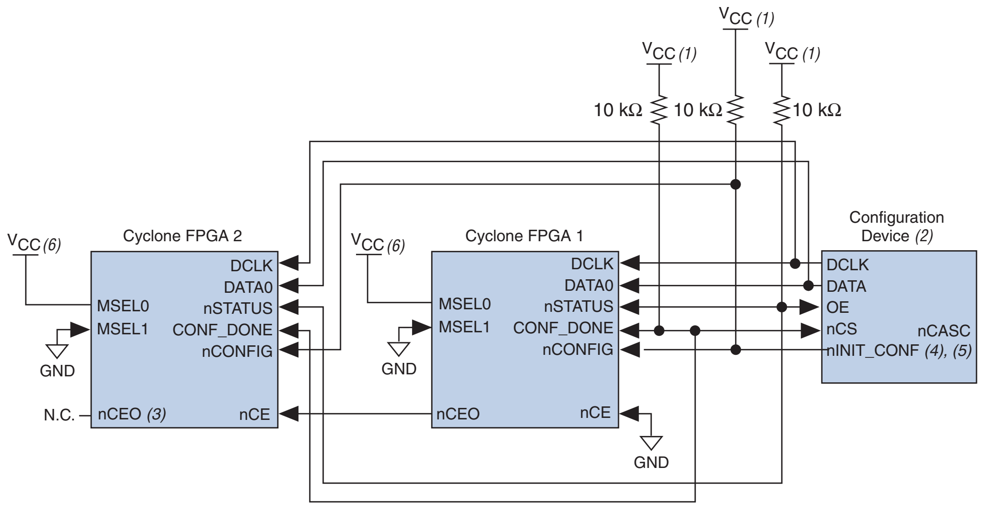
(3) The nCEO pin is left unconnected for the last device in the chain.

(4) Connect MSEL0 to the V_{CC} supply voltage of the I/O bank it resides in.

Configuring Multiple Cyclone FPGAs

You can use a single configuration device to configure multiple Cyclone FPGAs. In this setup, the nCEO pin of the first device is connected to the nCE pin of the second device in the chain. If there are additional devices, connect the nCE pin of the next device to the nCEO pin of the previous device. You should leave the nCEO pin on the last device in the chain unconnected. To configure properly, all of the target device CONF_DONE and nSTATUS pins must be tied together. Figure 5–12 shows an example of configuring multiple Cyclone FPGAs using a single configuration device.

Figure 5–12. Configuring Multiple Cyclone FPGAs with a Single Configuration Device



Notes to Figure 5–12:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device. The EPC16, EPC8, EPC4, and EPC2 devices' `OE` and `nCS` pins have internal, user-configurable pull-up resistors. If you use internal pull-up resistors, do not use external pull-up resistors on these pins.

(2) EPC16, EPC8, and EPC4 configuration devices cannot be cascaded.

(3) The `nCEO` pin is left unconnected for the last device in the chain.

(4) The `nINIT_CONF` pin is available on EPC16, EPC8, EPC4, and EPC2 devices. If `nINIT_CONF` is not used, `nCONFIG` must be pulled to V_{CC} through a resistor.

(5) The `nINIT_CONF` pin has an internal pull-up resistor that is always active in EPC16, EPC8, EPC4, and EPC2 devices. These devices do not need an external pull-up resistor on the `nINIT_CONF` pin.

(6) Connect `MSEL0` to the V_{CC} supply voltage of the I/O bank it resides in.

When performing multi-device PS configuration, you must generate the configuration device programming file (**.sof**) from each project. Then you must combine multiple **.sof** files using the Quartus II software through the **Convert Programming Files** dialog box.

For more information on how to create Programmer Object Files (**.pof**) for enhanced configuration devices, see *AN 218: Using Enhanced Configuration Devices*.

After the first Cyclone FPGA completes configuration during multi-device configuration, its `nCEO` pin activates the second device's `nCE` pin, prompting the second device to begin configuration. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

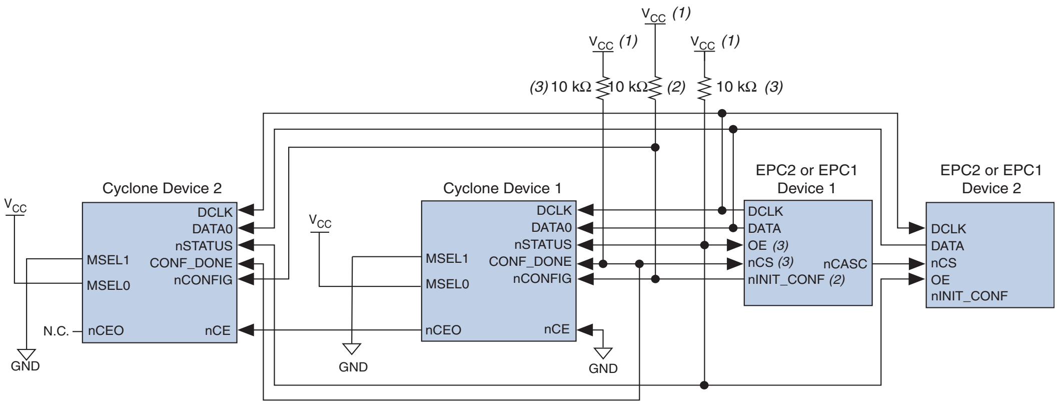
In addition, all nSTATUS pins are tied together; therefore, if any device (including the configuration device) detects an error, configuration stops for the entire chain. Also, if the configuration device does not detect CONF_DONE going high at the end of configuration, it resets the chain by pulsing its OE pin low for a few microseconds. For CONF_DONE to reach a high state, enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit. EPC2 devices wait for 16 DCLK cycles.

If the **Auto-Restart Configuration on Frame Error** option is turned on in the Quartus II software, the Cyclone FPGA releases its nSTATUS pins after a reset time-out period (about 30 µs). When the nSTATUS pins are released and pulled high, the configuration device reconfigures the chain. If the **Auto-Restart Configuration on Frame Error** option is not turned on, the devices drive nSTATUS low until they are reset with a low pulse on nCONFIG.

You can also cascade several EPC2 or EPC1 configuration devices to configure multiple Cyclone FPGAs. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the subsequent EPC2 or EPC1 device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted. You cannot cascade EPC16, EPC8, and EPC4 configuration devices.

Figure 5–13 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 5–13. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 5–13:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning an external pull-up resistor should not be used on the nINIT_CONF-nCONFIG line. The nINIT_CONF pin does not need to be connected if its function is not used. If nINIT_CONF is not used or not available (such as on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

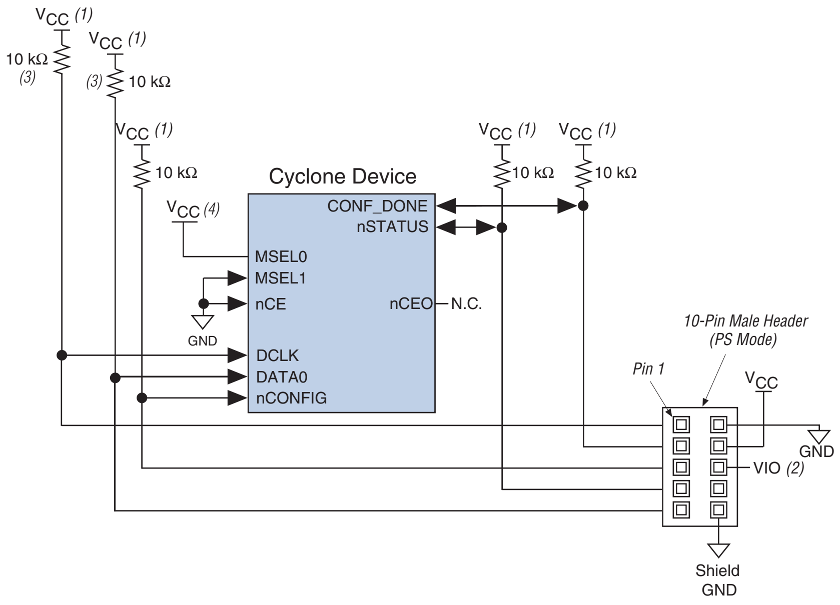
(3) The enhanced configuration devices' and EPC2 devices' OE and nCS pins have internal programmable pull-up resistors. External 10-kΩ pull-up resistors should be used. To turn off the internal pull-up resistors, check the **Disable nCS and OE pull-ups on configuration device** option when generating programming files.

PS Configuration Using a Download Cable

Using a download cable in PS configuration, an intelligent host (for example, your PC) transfers data from a storage device (for example, your hard drive) to the Cyclone FPGA through a USB Blaster, ByteBlaster II, MasterBlaster, or ByteBlasterMV cable. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin. The programming hardware then sends the configuration data one bit at a time on the device's `DATA0` pin. The data is clocked into the target device using `DCLK` until the `CONF_DONE` goes high.

When using programming hardware for the Cyclone FPGA, turning on the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle because the Quartus II software must restart configuration when an error occurs. Figure 5–14 shows the PS configuration setup for the Cyclone FPGA using a USB Blaster, ByteBlaster II, MasterBlaster, or ByteBlasterMV cable.

Figure 5–14. PS Configuration Circuit with a Download Cable



Notes to Figure 5–14:

(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (`VIO` pin) or ByteBlasterMV cable.

(2) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. This pin is a no-connect pin for the ByteBlasterMV header.

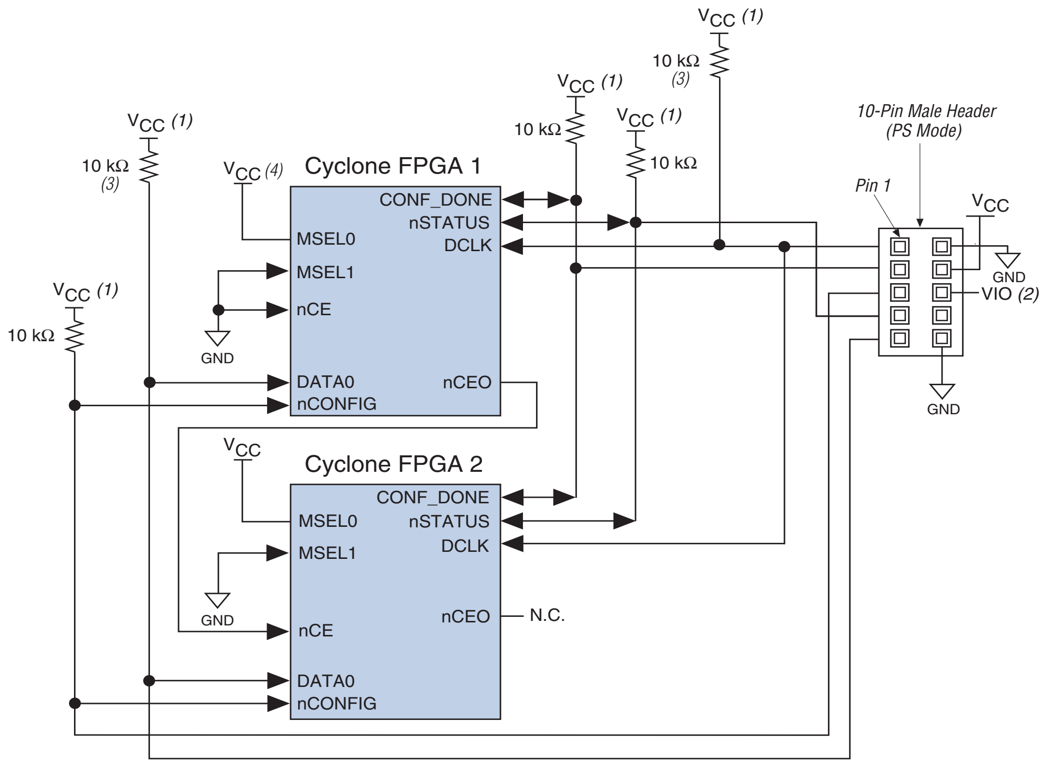
(3) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

(4) Connect `MSEL0` to the V_{CC} supply voltage of the I/O bank it resides in.

You can use the download cable to configure multiple Cyclone FPGAs by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. All other configuration pins are connected to each device in the chain.

Because all `CONF_DONE` pins are tied together, all devices in the chain initialize and enter user mode at the same time. In addition, because the `nSTATUS` pins are tied together, the entire chain halts configuration if any device detects an error. In this situation, the Quartus II software must restart configuration; the **Auto-Restart Configuration on Frame Error** option does not affect the configuration cycle. Figure 5–15 shows how to configure multiple Cyclone FPGAs with a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable.

Figure 5–15. Multi-Device PS Configuration with a Download Cable



Notes to Figure 5–15:

(1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (`VIO` pin) or ByteBlasterMV cable.

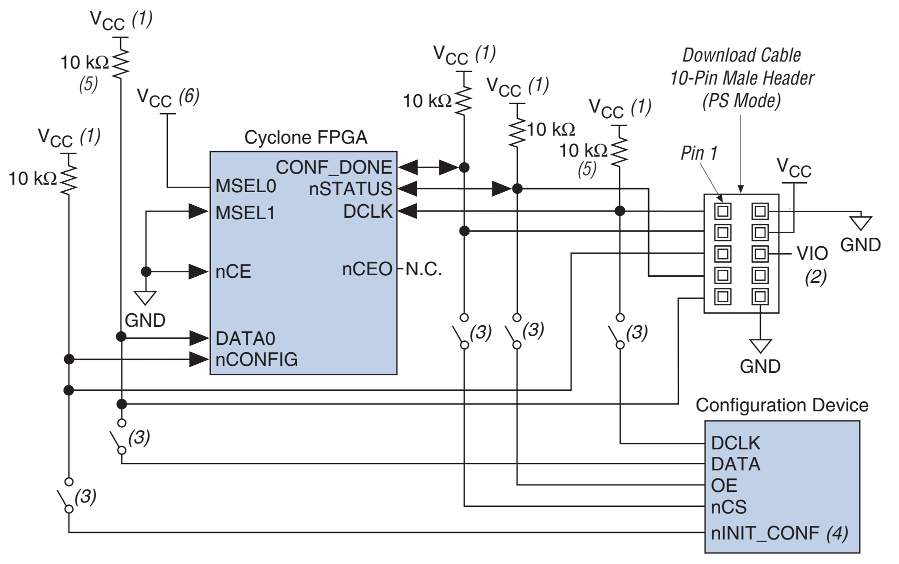
(2) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value.

(3) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

(4) Connect `MSEL0` to the V_{CC} supply voltage of the I/O bank it resides in.

If you are using a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable to configure device(s) on a board that also is populated with configuration devices, you should electrically isolate the configuration devices from the target device(s) and cable. One way to isolate the configuration devices is to add logic, such as a multiplexer, that can select between the configuration devices and the cable. The multiplexer allows bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`CONF_DONE`, `nSTATUS`, `DCLK`, nCONFIG, and DATA0) between the cable and the configuration devices. The last option is to remove the configuration devices from the board when configuring with the cable. Figure 5–16 shows a combination of a configuration device and a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable to configure a Cyclone FPGA.

Figure 5–16. Configuring with a Combined PS & Configuration Device Scheme



Notes to Figure 5–16:

(1) You should connect the pull-up resistor to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a V_{IO} reference voltage for the MasterBlaster output driver. V_{IO} should match the target device's V_{CCIO}. This is a no-connect pin for the ByteBlasterMV header.

(3) You should not attempt configuration with a ByteBlaster II, MasterBlaster, or ByteBlasterMV cable while a configuration device is connected to a Cyclone FPGA. Instead, 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. Remove the ByteBlaster II, MasterBlaster, or ByteBlasterMV cable when configuring with a configuration device.

(4) If nINIT_CONF is not used, nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(5) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

(6) Connect MSEL0 to the V_{CC} supply voltage of the I/O bank it resides in.

For more information on how to use the ByteBlaster II, MasterBlaster, or ByteBlasterMV cables, see the following documents:

- *ByteBlaster II Parallel Port Download Cable Data Sheet*
- *MasterBlaster Serial/USB Communications Cable Data Sheet*
- *ByteBlasterMV Parallel Port Download Cable Data Sheet*

PS Configuration from a Microprocessor

In PS configuration with a microprocessor, a microprocessor transfers data from a storage device to the target Cyclone FPGA. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the nCONFIG pin and the target device must release nSTATUS. The microprocessor then places the configuration data one bit at a time on the DATA0 pin of the Cyclone FPGA. The least significant bit (LSB) of each data byte must be presented first. Data is clocked continuously into the target device using DCLK until the CONF_DONE signal goes high.

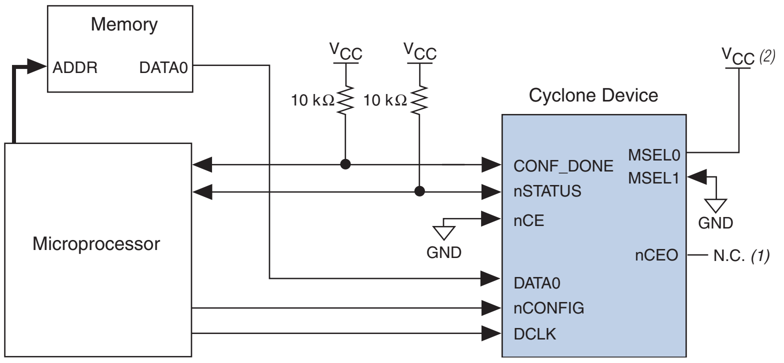
The Cyclone FPGA starts initialization using the internal oscillator after all configuration data is transferred. After initialization, this internal oscillator is turned off. The device's CONF_DONE pin goes high to show successful configuration and the start of initialization. During configuration and initialization and before the device enters user ode the microprocessor must not drive CONF_DONE low. Driving DCLK to the device after configuration does not affect device operation.

Since the PS configuration scheme is a synchronous scheme, the configuration clock speed must be below the specified maximum frequency to ensure successful configuration. Maximum DCLK frequency supported by Cyclone FPGAs is 100 MHz (see Table 5–5 on page 5–30). No maximum DCLK period (i.e., minimum DCLK frequency) exists. You can pause configuration by halting DCLK for an indefinite amount of time.

If the target device detects an error during configuration, it drives its nSTATUS pin low to alert the microprocessor. The microprocessor can then pulse nCONFIG low to restart the configuration process. Alternatively, if the **Auto-Restart Configuration on Frame Error** option is turned on in the Quartus II software, the target device releases nSTATUS after a reset time-out period. After nSTATUS is released, the microprocessor can reconfigure the target device without needing to pulse nCONFIG low.

The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration and initialization. If the microprocessor sends all data, but CONF_DONE and INIT_DONE has not gone high, it must reconfigure the target device. Figure 5–17 shows the circuit for PS configuration with a microprocessor.

Figure 5–17. PS Configuration Circuit with a Microprocessor



Notes to Figure 5–17:

(1) The `nCEO` pin is left unconnected.

(2) Connect `MSEL0` to the V_{CC} supply voltage of the I/O bank it resides in.

Configuring Cyclone FPGAs with the MicroBlaster Software

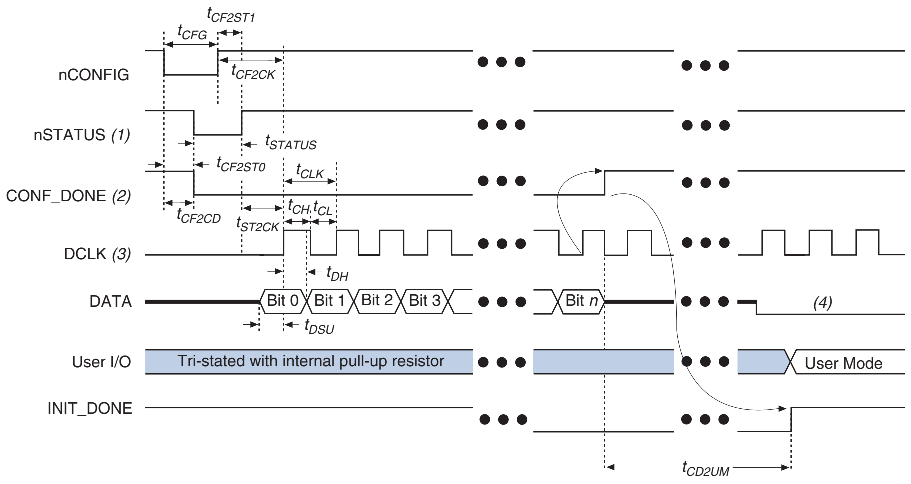
The MicroBlaster™ software driver allows you to configure Altera FPGAs, including Cyclone FPGAs, through the ByteBlaster II or ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a Raw Binary File (**.rbf**) programming input file and is targeted for embedded PS configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, see the *Configuring the MicroBlaster Passive Serial Software Driver White Paper* and source files on the Altera web site at www.altera.com.

Passive Serial Timing

For successful configuration using the PS scheme, several timing parameters such as setup, hold, and maximum clock frequency must be satisfied. The enhanced configuration and EPC2 devices are designed to meet these interface timing specifications. If you use a microprocessor or another intelligent host to control the PS interface, ensure that you meet these timing requirements.

Figure 5–18 shows the PS timing waveform for Cyclone FPGAs.

Figure 5–18. PS Timing Waveform for Cyclone FPGAs



Notes to Figure 5–18:

(1) Upon power-up, the Cyclone FPGA holds `nSTATUS` low for about 100 ms.
(2) Upon power-up and before configuration, `CONF_DONE` is low.
(3) In user mode, `DCLK` should be driven high or low when using the PS configuration scheme. When using the AS configuration scheme, `DCLK` is a Cyclone output pin and should not be driven externally.
(4) `DATA` should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Table 5–5 contains the PS timing information for Cyclone FPGAs.

Table 5–5. PS Timing Parameters for Cyclone Devices Note (1) (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		800	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		800	ns
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		40 (4)	µs
t_{CFG}	`nCONFIG` low pulse width (2)	40		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 (4)	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	7		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns

Table 5–5. PS Timing Parameters for Cyclone Devices Note (1) (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CH}	DCLK high time	4		ns
t_{CL}	DCLK low time	4		ns
t_{CLK}	DCLK period	10		ns
f_{MAX}	DCLK maximum frequency		100	MHz
t_{CD2UM}	CONF_DONE high to user mode *(3)*	6	20	µs

Notes to Table 5–5:

(1) This information is preliminary.

(2) This value applies only if the internal oscillator is selected as the clock source for device initialization. If the clock source is CLKUSR, multiply the clock period by 270 to obtain this value. CLKUSR must be running during this period to reset the device.

(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for device initialization. If the clock source is CLKUSR, multiply the clock period by 140 to obtain this value.

(4) You can obtain this value if you do not delay configuration by extending the nSTATUS low-pulse width.

Device configuration options and how to create configuration files are discussed further in the *Software Settings* chapter in Volume II of the *Configuration Handbook*.

JTAG-Based Configuration

JTAG has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on printed circuit boards (PCBs) with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. You can also use the JTAG circuitry to shift configuration data into Cyclone FPGAs. The Quartus II software automatically generates **.sof** files that can be used for JTAG configuration.

For more information on JTAG boundary-scan testing, see *AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

To use the SignalTap II Embedded Logic Analyzer, you need to connect the JTAG pins of your Cyclone device to a download cableheader on your PCB.

For more information on SignalTap II, see the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in the *Quartus II Handbook*.

Cyclone devices are designed such that JTAG instructions have precedence over any device operating modes. So JTAG configuration can take place without waiting for other configuration to complete (e.g., configuration with serial or enhanced configuration devices). If you attempt JTAG configuration in Cyclone FPGAs during non-JTAG configuration, non-JTAG configuration is terminated and JTAG configuration is initiated.

☞ The Cyclone configuration data decompression feature is not supported in JTAG-based configuration.

A device operating in JTAG mode uses four required pins: `TDI`, `TDO`, `TMS`, and `TCK`. Cyclone FPGAs do not support the optional `TRST` pin. The three JTAG input pins, `TCK`, `TDI`, and `TMS`, have weak internal pull-up resistors, whose values are approximately 20 to 40 kΩ. All user I/O pins are tri-stated during JTAG configuration.

Table 5–6 shows each JTAG pin's function.

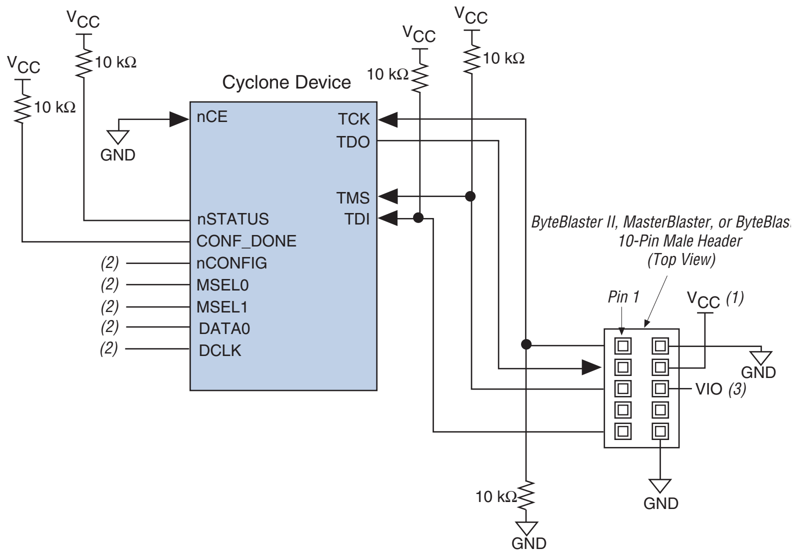
Table 5–6. JTAG Pin Descriptions

Pin	Description	Function
`TDI`	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TDO`	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	Test mode select	Input pin that provides the control signal to determine the transitions of the Test Access Port (TAP) controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
`TCK`	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled, by connecting this pin to GND.

JTAG Configuration Using a Download Cable

During JTAG configuration, data is downloaded to the device on the board through a USB Blaster, ByteBlaster II, ByteBlasterMV, or MasterBlaster download cable. Configuring devices through a cable is similar to programming devices in-system. See Figure 5–19 for pin connection information.

Figure 5–19. JTAG Configuration of Single Cyclone FPGA



Notes to Figure 5–19:

(1) You should connect the pull-up resistor to the same supply voltage as the download cable.

(2) You should connect the nCONFIG, MSEL0, and MSEL1 pins to support a non-JTAG configuration scheme. If you only use JTAG configuration, connect nCONFIG to V_{CC}, and MSEL0 and MSEL1 to ground. Pull DATA0 and DCLK to high or low.

(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlaster MV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming; otherwise it is a no connect.

(4) nCE must be connected to GND or driven low for successful configuration.

To configure a single device in a JTAG chain, the programming software places all other devices in bypass mode. In bypass mode, devices pass programming data from the TDI pin to the TDO pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the TDO pin one clock cycle later.

The Quartus II software verifies successful JTAG configuration upon completion. The software checks the state of CONF_DONE through the JTAG port. If CONF_DONE is not high, the Quartus II software indicates that configuration has failed. If CONF_DONE is high, the software indicates that configuration was successful. After the configuration bit stream is transmitted serially via the JTAG TDI port, the TCK port is clocked an additional 134 cycles to perform device initialization.

☞ If V_{CCIO} is tied to 3.3-V, both the I/O pins and the JTAG `TDO` port drive at 3.3-V levels.

Cyclone FPGAs have dedicated JTAG pins. Not only can you perform JTAG testing on Cyclone FPGAs before and after, but also during configuration. While other device families do not support JTAG testing during configuration, Cyclone FPGAs support the `BYPASS`, `IDCODE`, and `SAMPLE` instructions during configuration without interrupting configuration. All other JTAG instructions may only be issued by first interrupting configuration and reprogramming I/O pins using the `CONFIG_IO` instruction.

The `CONFIG_IO` instruction allows I/O buffers to be configured via the JTAG port, and when issued, interrupts configuration. This instruction allows you to perform board-level testing prior to configuring the Cyclone FPGA or waiting for a configuration device to complete configuration. Once configuration has been interrupted and JTAG testing is complete, the part must be reconfigured via JTAG (`PULSE_CONFIG` instruction) or by pulsing `nCONFIG` low.

The chip-wide reset and output enable pins on Cyclone FPGAs do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of Cyclone FPGAs, you should consider the dedicated configuration pins. Table 5–7 shows how you should connect these pins during JTAG configuration.

Table 5–7. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	Drive all Cyclone devices in the chain low by connecting nCE to ground, pulling it down via a resistor, or driving it low by some control circuitry. For devices in a multi-device PS and AS configuration chains, connect the nCE pins to ground during JTAG configuration or configure them via JTAG in the same order as the configuration chain.
nCEO	For all Cyclone devices in a chain, the nCEO pin can be left floating or connected to the nCE pin of the next device. See nCE description above.
nSTATUS	Pulled to V_{CC} through a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, pull up each nSTATUS pin to V_{CC} individually. *(1)*
CONF_DONE	Pulled to V_{CC} through a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, pull up each CONF_DONE pin to V_{CC} individually. *(1)*
nCONFIG	Driven high by connecting to V_{CC}, pulling up through a resistor, or driving it high by some control circuitry.
MSEL0, MSEL1	Do not leave these pins floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie these pins to ground.
DCLK	Do not leave these pins floating. Drive low or high, whichever is more convenient.
DATA0	Do not leave these pins floating. Drive low or high, whichever is more convenient.

Note to Table 5–7:

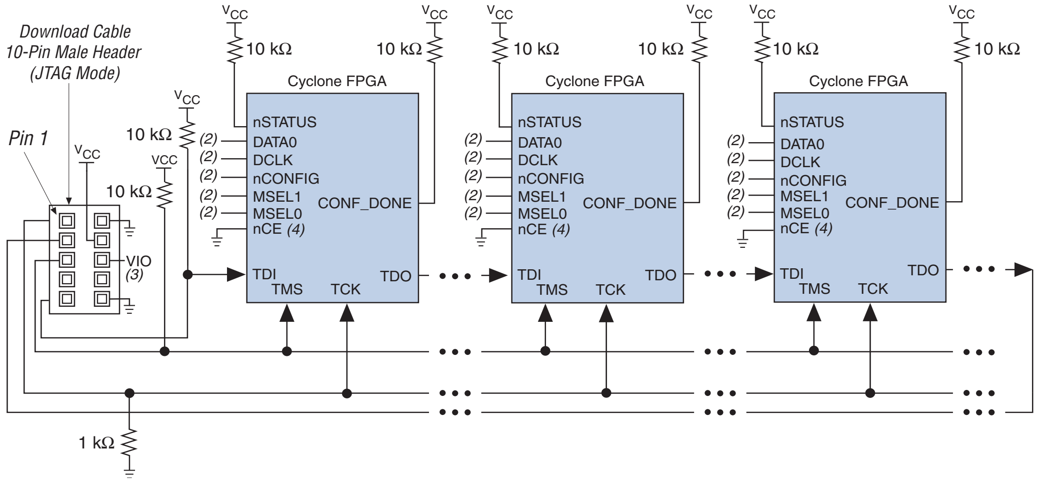
(1) nSTATUS going low in the middle of JTAG configuration indicates that an error has occurred; CONF_DONE going high at the end of JTAG configuration indicates successful configuration.

JTAG Configuration of Multiple Devices

When programming a JTAG device chain, one JTAG-compatible header, such as the ByteBlaster II header, is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capacity of the download cable. However, when four or more devices are connected in a JTAG chain, Altera recommends buffering the TCK, TDI, and TMS pins with an on-board buffer.

JTAG-chain device configuration is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 5–20 shows multi-device JTAG configuration.

Figure 5–20. Multi-Device JTAG Configuration *Note (1)*



Notes to Figure 5–20:

(1) Cyclone, Stratix, Stratix GX, APEX™ II, APEX 20K, Mercury™, ACEX® 1K, and FLEX® 10K devices can be placed within the same JTAG chain for device programming and configuration.

(2) Connect the `nCONFIG`, `MSEL0`, and `MSEL1` pins to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground. Pull `DATA0` and `DCLK` to either high or low.

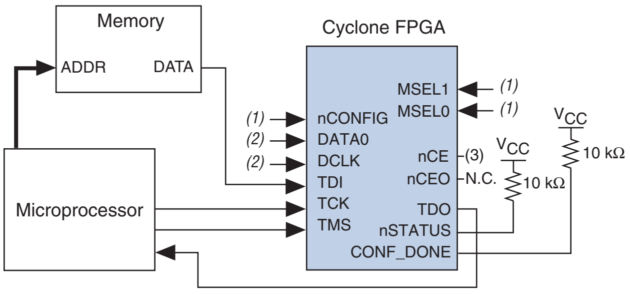
(3) V_{IO} is a reference voltage for the MasterBlaster output driver. V_{IO} should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlaster MV, this pin is a no connect. In the USB Blaster and ByteBlaster, this pin is connected to `nCE` when it is used for Active Serial programming; otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful configuration.

Connect the `nCE` pin to ground or drive it low during JTAG configuration. In multi-device PS and AS configuration chains, connect the first device's `nCE` pin to ground and connect the `nCEO` pin to the `nCE` pin of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, it's `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to ground during JTAG configuration or that the devices are configured via JTAG in the same order as the configuration chain. As long as the devices are configured in the same order as the multi-device configuration chain, the `nCEO` pin of the previous device drives the `nCE` pin of the next device low when it has successfully been configured.

Figure 5–21 shows the JTAG configuration of a Cyclone FPGA with a microprocessor.

Figure 5–21. JTAG Configuration of Cyclone FPGAs with a Microprocessor



Notes to Figure 5–21:

(1) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

(2) Pull `DATA0` and `DCLK` to either high or low.

(3) `nCE` must be connected to GND or driver low for succesful JTAG configuration.

For more information about JTAG programming in an embedded environment, refer to *AN 122: Using JamSTAPL for ISP &ICR via an Embedded Processor.*

Configuring Cyclone FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including Cyclone FPGAs, through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in **.rbf** format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on other platforms. For more information on the JRunner software driver, see JRunner Software Driver: An Embedded Solution to the JTAG Configuration and the source files on the Altera web site.

Jam STAPL

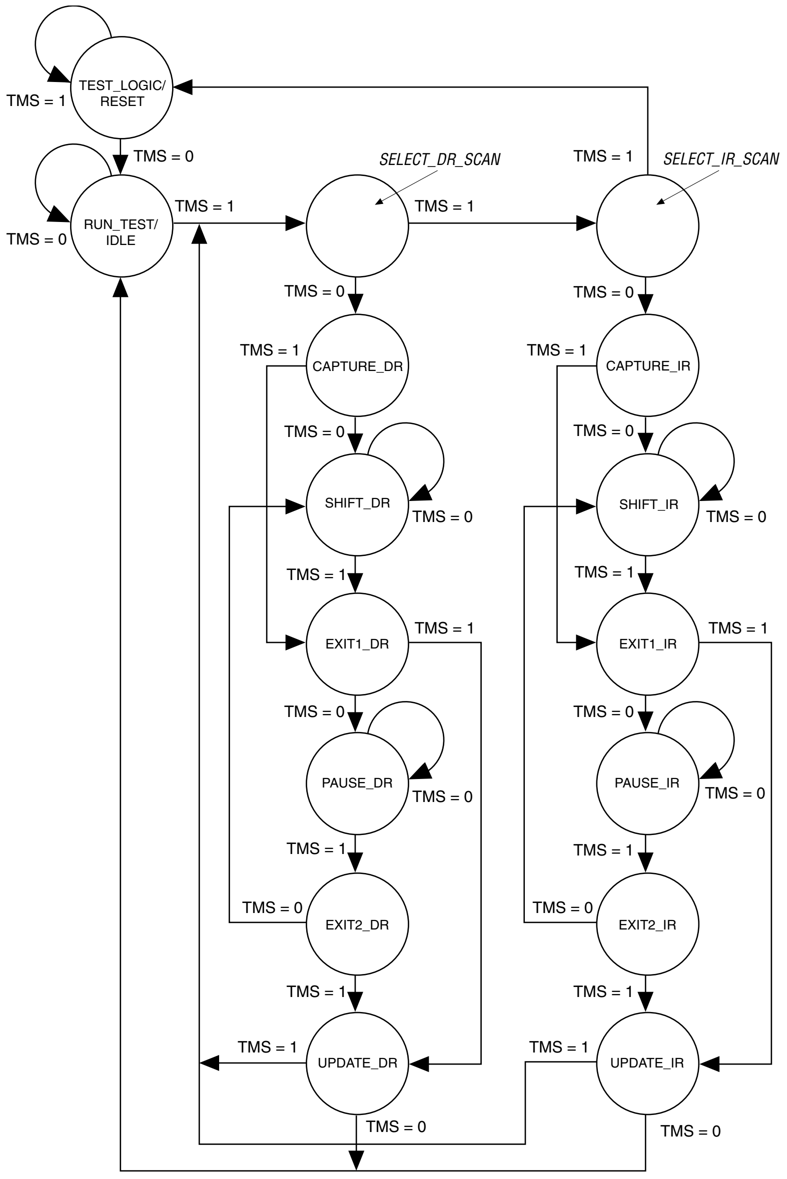
Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

☞ Both JTAG connection methods should include space for the MasterBlaster or ByteBlasterMV header connection. The header is useful during prototyping because it allows you to verify or modify the Cyclone FPGA's contents. During production, you can remove the header to save cost.

Program Flow

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine. The TAP controller is a 16-state, state machine that is clocked on the rising edge of `TCK`, and uses the `TMS` pin to control JTAG operation in a device. Figure 5–22 shows the flow of an IEEE Std. 1149.1 TAP controller state machine.

Figure 5–22. JTAG TAP Controller State Machine



While the Jam Player provides a driver that manipulates the TAP controller, the Jam Byte-Code File (**.jbc**) provides the high-level intelligence needed to program a given device. All Jam instructions that send JTAG data to the device involve moving the TAP controller through either the data register leg or the instruction register leg of the state machine. For example, loading a JTAG instruction involves moving the TAP controller to the `SHIFT_IR` state and shifting the instruction into the instruction register through the `TDI` pin. Next, the TAP controller is moved to the `RUN_TEST/IDLE` state where a delay is implemented to allow the instruction time to be latched. This process is identical for data register scans, except that the data register leg of the state machine is traversed.

The high-level Jam instructions are the `DRSCAN` instruction for scanning the JTAG data register, the `IRSCAN` instruction for scanning the instruction register, and the `WAIT` command that causes the state machine to sit idle for a specified period of time. Each leg of the TAP controller is scanned repeatedly, according to instructions in the **.jbc** file, until all of the target devices are programmed.

Figure 5–23 shows the functional behavior of the Jam Player when it parses the **.jbc** file. When the Jam Player encounters a `DRSCAN`, `IRSCAN`, or `WAIT` instruction, it generates the proper data on `TCK`, `TMS`, and `TDI` to complete the instruction. The flow diagram shows branches for the `DRSCAN`, `IRSCAN`, and `WAIT` instructions. Although the Jam Player supports other instructions, they are omitted from the flow diagram for simplicity.

Figure 5–23. Jam Player Flow Diagram (Part 1 of 2)

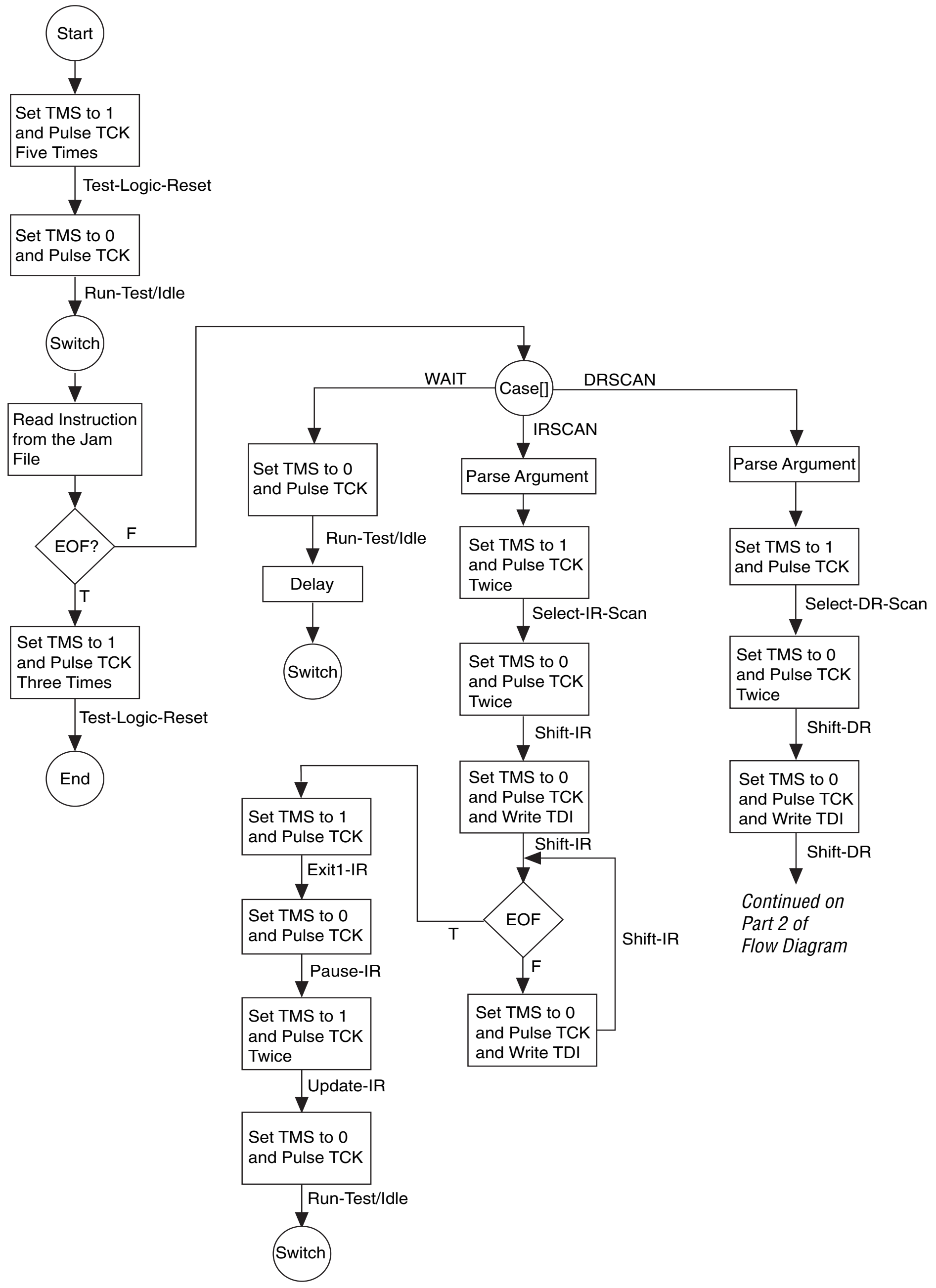
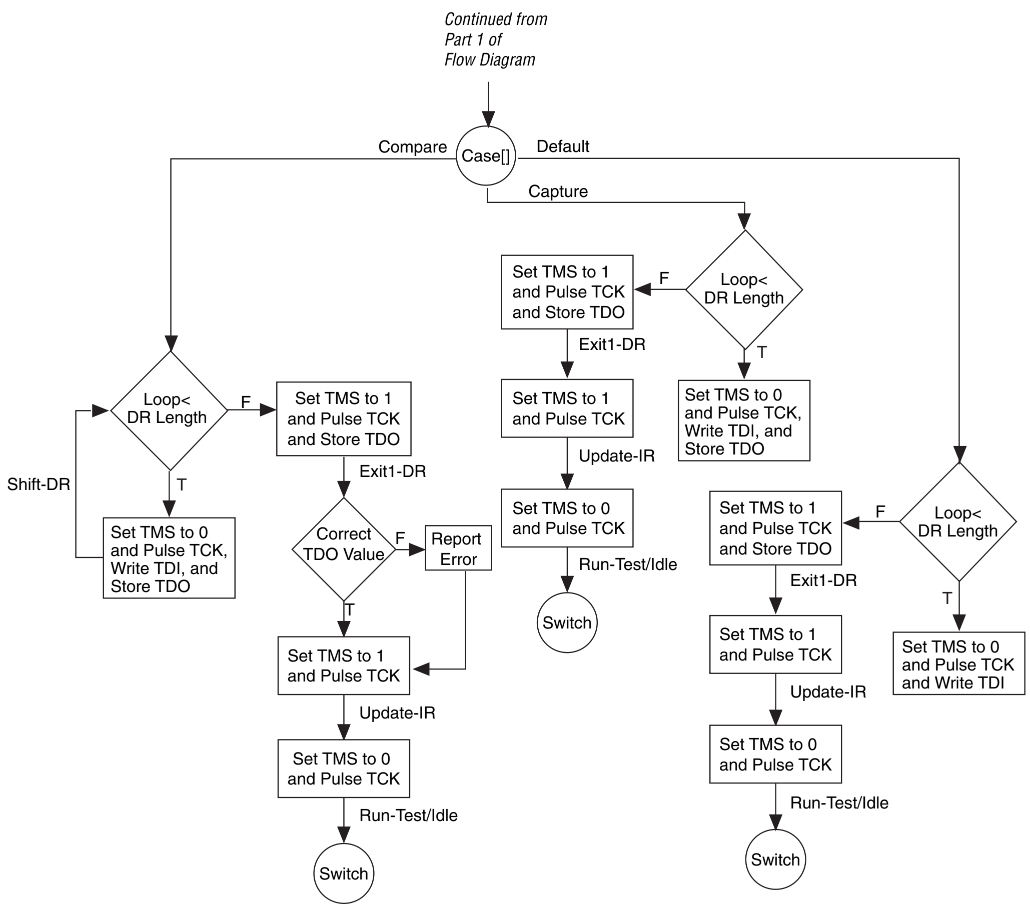

Start
Set TMS to 1 and Pulse TCK Five Times
Test-Logic-Reset
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Read Instruction from the Jam File
EOF?
F
T
Set TMS to 1 and Pulse TCK Three Times
Test-Logic-Reset
End
WAIT
Case[]
DRSCAN
IRSCAN
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Delay
Switch
Parse Argument
Set TMS to 1 and Pulse TCK Twice
Select-IR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-IR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-IR
EOF
T
F
Shift-IR
Set TMS to 0 and Pulse TCK and Write TDI
Set TMS to 1 and Pulse TCK
Exit1-IR
Set TMS to 0 and Pulse TCK
Pause-IR
Set TMS to 1 and Pulse TCK Twice
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Parse Argument
Set TMS to 1 and Pulse TCK
Select-DR-Scan
Set TMS to 0 and Pulse TCK Twice
Shift-DR
Set TMS to 0 and Pulse TCK and Write TDI
Shift-DR
Continued on Part 2 of Flow Diagram

Figure 5–24. Jam Player Flow Diagram (Part 2 of 2)


Continued from Part 1 of Flow Diagram
Case[]
Compare
Default
Capture
Loop< DR Length
F
T
Shift-DR
Set TMS to 1 and Pulse TCK and Store TDO
Set TMS to 0 and Pulse TCK, Write TDI, and Store TDO
Exit1-DR
Correct TDO Value
Report Error
Set TMS to 1 and Pulse TCK
Update-IR
Set TMS to 0 and Pulse TCK
Run-Test/Idle
Switch
Set TMS to 0 and Pulse TCK and Write TDI

Execution of a Jam program starts at the beginning of the program. The program flow is controlled using `GOTO`, `CALL/RETURN`, and `FOR/NEXT` structures. The `GOTO` and `CALL` statements refer to labels that are symbolic names for program statements located elsewhere in the Jam program. The language itself enforces almost no constraints on the organizational structure or control flow of a program.

The Jam language does not support linking multiple Jam programs together or including the contents of another file into a Jam program.

Jam Instructions

Each Jam statement begins with one of the instruction names listed in Table 5–8. The instruction names, including the names of the optional instructions, are reserved keywords that you cannot use as variable or label identifiers in a Jam program.

Table 5–8. Instruction Names

BOOLEAN	INTEGER	PREIR
CALL	IRSCAN	PRINT
CRC	IRSTOP	PUSH
DRSCAN	LET	RETURN
DRSTOP	NEXT	STATE
EXIT	NOTE	WAIT
EXPORT	POP	VECTOR *(1)*
FOR	POSTDR	VMAP *(1)*
GOTO	POSTIR	-
IF	PREDR	-

Note to Table 5–8:
(1) This instruction name is an optional language extension.

Table 5–9 shows the state names that are reserved keywords in the Jam language. These keywords correspond to the state names specified in the IEEE Std. 1149.1 JTAG specification.

Table 5–9. Reserved Keywords (Part 1 of 2)

IEEE Std. 1149.1 JTAG State Names	Jam Reserved State Names
Test-Logic-Reset	RESET
Run-Test-Idle	IDLE
Select-DR-Scan	DRSELECT
Capture-DR	DRCAPTURE
Shift-DR	DRSHIFT
Exit1-DR	DREXIT1
Pause-DR	DRPAUSE
Exit2-DR	DREXIT2
Update-DR	DRUPDATE
Select-IR-Scan	IRSELECT
Capture-IR	IRCAPTURE

Table 5–9. Reserved Keywords (Part 2 of 2)

IEEE Std. 1149.1 JTAG State Names	Jam Reserved State Names
`Shift-IR`	`IRSHIFT`
`Exit1-IR`	`IREXIT1`
`Pause-IR`	`IRPAUSE`
`Exit2-IR`	`IREXIT2`
`Update-IR`	`IRUPDATE`

Example Jam File that Reads the IDCODE

The following illustrates the flexibility and utility of the Jam STAPL. The example code reads the IDCODE out of a single device in a JTAG chain.

☞ The array variable, `I_IDCODE`, is initialized with the IDCODE instruction bits ordered the LSB first (on the left) to most significant bit (MSB) (on the right). This order is important because the array field in the IRSCAN instruction is always interpreted and sent, MSB to LSB.

Example Jam File Reading IDCODE

```
BOOLEAN read_data[32];
BOOLEAN I_IDCODE[10] = BIN 1001101000; ‘assumed
BOOLEAN ONES_DATA[32] = HEX FFFFFFFF;
INTEGER i;
‘Set up stop state for IRSCAN
IRSTOP IRPAUSE;
‘Initialize device
STATE RESET;
IRSCAN 10, I_IDCODE[0..9]; ‘LOAD IDCODE INSTRUCTION
STATE IDLE;
WAIT 5 USEC, 3 CYCLES;
DRSCAN 32, ONES_DATA[0..31], CAPTURE read_data[0..31];
‘CAPTURE IDCODE
PRINT “IDCODE:”;
FOR i=0 to 31;
PRINT read_data[i];
NEXT i;
EXIT 0;
```

Combining Configuration Schemes

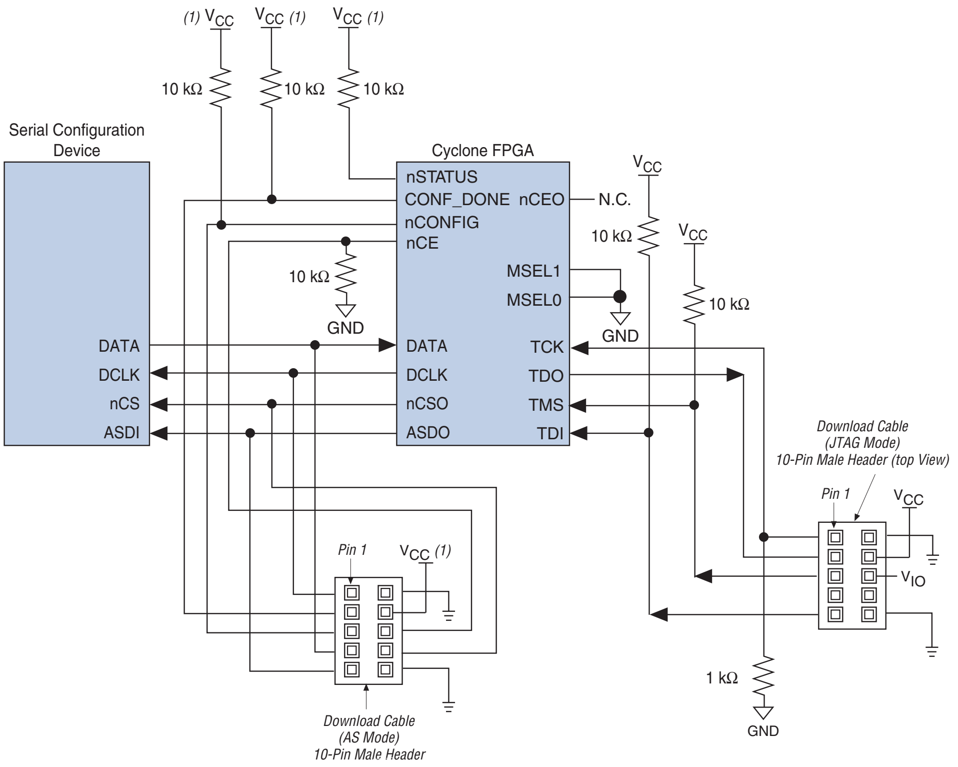
This section shows you how to configure Cyclone FPGAs using multiple configuration schemes on the same board.

Active Serial & JTAG

You can combine the AS configuration scheme with JTAG-based configuration. Set the `MSEL[1..0]` pins to `00` in this setup, as shown in Figure 5–25. This setup uses two 10-pin download cable headers on the board. The first header programs the serial configuration device in-system via the AS programming interface, and the second header configures the Cyclone FPGA directly via the JTAG interface.

If you try configuring the device using both schemes simultaneously, JTAG configuration takes precedence and AS configuration is terminated.

Figure 5–25. Combining AS & JTAG Configuration



Notes to Figure 5–25:

(1) Connect these pull-up resistors to 3.3 V.

Device Configuration Pins

Tables 5–10 through 5–12 describe the connections and functionality of all the configuration related pins on the Cyclone device. Table 5–10 describes the dedicated configuration pins. These pins are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 5–10. Dedicated Cyclone Device Configuration Pins (Part 1 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`MSEL1` `MSEL0`	–	All	Input	Two-bit configuration input that set the Cyclone device configuration scheme (see Table 5–2). Use these pins to select the Cyclone configuration schemes for the appropriate connections. These pins must remain at a valid state during power-up before `nCONFIG` is pulled low to initiate a reconfiguration and during configuration. This pin uses Schmitt trigger input buffers.
`nCONFIG`	–	All	Input	Configuration control input. Pulling this pin low during user-mode causes the FPGA to lose its configuration data, enter a reset state, and tri-state all I/O pins. Returning this pin to a logic high initiates a reconfiguration. If the configuration scheme uses an enhanced configuration device or EPC2 device, the `nCONFIG` pin can be tied directly to V_{CC} or to the configuration device's `nINIT_CONF` pin. This pin uses Schmitt trigger input buffers

Table 5–10. Dedicated Cyclone Device Configuration Pins (Part 2 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nSTATUS	–	All	Bidirectional open-drain	The device drives nSTATUS low immediately after power-up and releases it within 5 µs. (When using a configuration device, the configuration device holds nSTATUS low for up to 200 ms.) Status output. If an error occurs during configuration, nSTATUS is pulled low by the target device. Status input. If an external source drives the nSTATUS pin low during configuration or initialization, the target device enters an error state. Driving nSTATUS low after configuration and initialization does not affect the configured device. If the design uses a configuration device, driving nSTATUS low causes the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on nSTATUS in user-mode, the FPGA does not reconfigure. To initiate a reconfiguration, nCONFIG must be pulled low. The OE and nCS pins in the enhanced configuration devices and EPC2 devices have optional internal programmable pull-up resistors. If the design uses internal pull-up resistors, do not use external 10-kΩ pull-up resistors on these pins. This pin uses Schmitt trigger input buffers
CONF_DONE	–	All	Bidirectional open-drain	Status output. The target device drives the CONF_DONE pin low before and during configuration. Once all configuration data is received without error and the initialization clock cycle starts, the target device releases CONF_DONE. Status input. After all data is received and CONF_DONE goes high, the target device initializes and enters user mode. Driving CONF_DONE low after configuration and initialization does not affect the configured device. The OE and nCS pins in the enhanced configuration devices and EPC2 devices have optional internal programmable pull-up resistors. If the design uses internal pull-up resistors, do not use external 10-kΩ pull-up resistors on these pins. This pin uses Schmitt trigger input buffers

Table 5–10. Dedicated Cyclone Device Configuration Pins (Part 3 of 3)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
DCLK	–	PS AS	Input (PS) Output (AS)	In PS configuration, the clock input clocks data from an external source into the target device. Data is latched into the FPGA on the rising edge of DCLK. In AS configuration, DCLK is an output from the Cyclone FPGA that provides timing for the configuration interface. After configuration, the logic levels on this pin do not affect the Cyclone FPGA. This pin uses Schmitt trigger input buffers
ASDO	I/O in PS mode, N/A in AS mode	AS	Output	Control signal from the Cyclone FPGA to the serial configuration device in AS mode used to read out configuration data.
nCSO	I/O in PS mode, N/A in AS mode	AS	Output	Output control signal from the Cyclone FPGA to the serial configuration device in AS mode that enables the configuration device.
nCE	–	All	Input	Active-low chip enable. The nCE pin activates the device with a low signal to allow configuration. The nCE pin must be held low during configuration, initialization, and user mode. In single device configuration, tie the nCE pin low. In multi-device configuration, the first device's nCE pin is tied low while its nCEO pin is connected to nCE of the next device in the chain. Hold the nCE pin low for programming the FPGA via JTAG. This pin uses Schmitt trigger input buffers
nCEO	–	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's nCE pin. The nCEO of the last device in the chain is left floating.
DATA0	–	All	Input	Data input. In serial configuration mode, bit-wide configuration data is presented to the target device on the DATA0 pin. Toggling DATA0 after configuration does not affect the configured device. This pin uses Schmitt trigger input buffers

Table 5–11 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration, these pins function as user I/O pins and are tri-stated with weak pull-ups.

Table 5–11. Optional Cyclone Device Configuration Pins

Pin Name	User Mode	Pin Type	Description
`CLKUSR`	N/A if option is on, I/O if option is off	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the **Enable user-supplied start-up clock (CLKUSR)** option in the Quartus II software.
`INIT_DONE`	N/A if option is on, I/O if option is off	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. The `INIT_DONE` pin must be pulled to V_{CC} with a 10-kΩ resistor. The `INIT_DONE` pin drives low during configuration. Before and after configuration, the `INIT_DONE` pin is released and is pulled to V_{CC} by an external pull-up resistor. Because `INIT_DONE` is tri-stated before configuration, it is pulled high by the external pull-up resistor. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the **Enable INIT_DONE output** option in the Quartus II software.
`DEV_OE`	N/A if the option is on, I/O if the option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the **Enable device-wide output enable (DEV_OE)** option in the Quartus II software.
`DEV_CLRn`	N/A if the option is on, I/O if the option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the **Enable device-wide reset (DEV_CLRn)** option in the Quartus II software.

Table 5–12 describes the dedicated JTAG pins. JTAG pins must be kept stable before and during configuration to prevent accidental loading of JTAG instructions.

Table 5–12. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
`TDI`	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}. This pin uses Schmitt trigger input buffers
`TDO`	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
`TMS`	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}. This pin uses Schmitt trigger input buffers
`TCK`	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to ground. This pin uses Schmitt trigger input buffers

6. Configuring APEX II Devices

CF51004-2.0

Introduction

APEX™ II devices can be configured using one of four configuration schemes. All configuration schemes use either a microprocessor or configuration device.

APEX II devices can be configured using the passive serial (PS), fast passive parallel (FPP), passive parallel asynchronous (PPA), and Joint Test Action Group (JTAG) configuration schemes. The configuration scheme used is selected by driving the APEX II device `MSEL1` and `MSEL0` pins either high or low as shown in Table 6–1. If your application only requires a single configuration mode, the `MSEL` pins can be connected to V_{CC} (V_{CCIO} of the I/O bank where the `MSEL` pin resides) or to ground. If your application requires more than one configuration mode, you can switch the `MSEL` pins after the FPGA is configured successfully. Toggling these pins during user-mode does not affect the device operation; however, the `MSEL` pins must be valid before a reconfiguration is initiated.

Table 6–1. APEX II Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	PS
1	0	FPP
1	1	PPA
(1)	*(1)*	JTAG Based *(2)*

Notes to Table 6–1:
(1) Do not leave the `MSEL` pins floating; connect them to a low- or high-logic level. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used, you should connect the `MSEL` pins to ground.
(2) JTAG-based configuration takes precedence over other configuration schemes, which means `MSEL` pin settings are ignored.

Table 6–2 shows the approximate configuration file sizes for APEX II devices.

Table 6–2. APEX II Raw Binary File (.rbf) Sizes

Device	Data Size (Bits)	Data Size (Bytes)
EP2A15	4,358,512	544,814
EP2A25	6,275,200	784,400
EP2A40	9,640,528	1,208,320
EP2A70	17,417,088	2,177,136

Use the data in Table 6–2 only to estimate the file size before design compilation. Different configuration file formats, such as a Hexidecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II or MAX+PLUS® II software, all designs targeted for the same device will have the same configuration file size.

The following chapter describes in detail how to configure APEX II devices using the supported configuration schemes. The last section describes the device configuration pins available. In this chapter, the generic term device(s) or FPGA(s) will include all APEX II devices.

For more information on setting device configuration options or creating configuration files, *see Section II, Software Settings, in Volume 2*.

Passive Serial Configuration

You can perform APEX II PS configuration using an Altera configuration device, an intelligent host (e.g., a microprocessor or Altera® MAX® device), or a download cable.

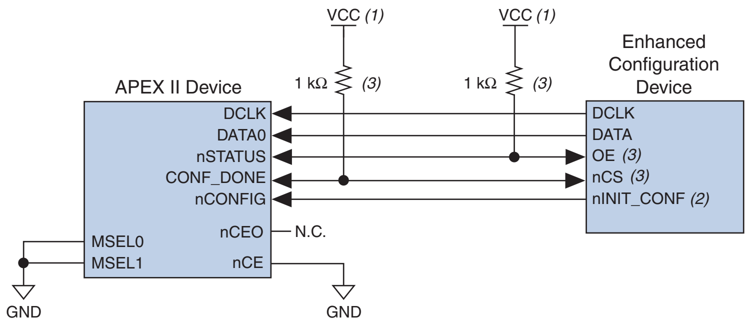
PS Configuration Using a Configuration Device

You can use an Altera configuration device, such as an enhanced configuration device, EPC2, or EPC1 device, to configure APEX II devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 6–1 shows the configuration interface connections between the APEX II device and a configuration device.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on the enhanced configuration device and flash interface pins (e.g., PGM[2..0], EXCLK, PORSEL, A[20..0], and DQ[15..0]), see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* in the Configuration Handbook.

Figure 6–1. Single Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 6–1:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the Operating Conditions table of the Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet or the Configuration Devices for SRAM-based LUT Devices Data Sheet.

When using enhanced configuration devices or EPC2 devices, nCONFIG of the FPGA can be connected to nINIT_CONF, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor. An internal pull-up on the nINIT_CONF pin is always active in enhanced configuration devices and EPC2 devices, which means an external pull-up resistor is not required if nCONFIG is tied to nINIT_CONF.

Upon power-up, the APEX II device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for EPC2, EPC1, and EPC1441 devices is 200 ms (maximum), and for enhanced configuration devices, the POR time can be set to either 100 ms or 2 ms, depending on its PORSEL pin setting. If the PORSEL pin is connected to GND, the POR delay is 100 ms. During this time, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When both devices complete POR, they release their open-drain OE or nSTATUS pin, which is then pulled high by a pull-up resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. The beginning of configuration can be delayed by holding the nCONFIG or nSTATUS pin low.

VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the nSTATUS pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up on the OE pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 1-kΩ pull-up resistor on the OE/nSTATUS line is required. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins.

When nSTATUS is pulled high, OE of the configuration device also goes high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The APEX II device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK.

After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up is not used, an external 1-kΩ pull-up resistor on the nCS/CONF_DONE line is required. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In APEX II devices, the initialization clock source is either the APEX II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX II device will supply itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. You can turn on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data is accepted and CONF_DONE goes high, APEX II devices require 40 clock cycles to properly initialize.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 1-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. The enhanced configuration device drives DCLK low and DATA high at the end of configuration.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the *Auto-Restart Configuration After Error* option available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box is turned on, the FPGA automatically initiates reconfiguration if an error occurs. The APEX II device will release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the FPGA has not configured successfully. 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. In this case, the configuration device pulls its OE pin low, which in turn drives the target device's nSTATUS pin low. If the *Auto-Restart Configuration After Error* option is set in the software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns high, the configuration device tries to reconfigure the FPGA.

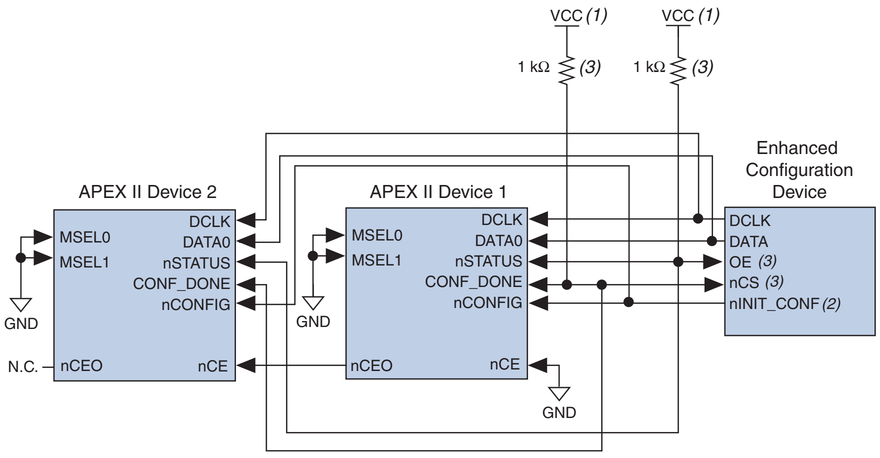
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully; therefore, your system should not pull CONF_DONE low to delay initialization. Instead, use the *CLKUSR* option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this will activate the configuration device since it will see its nCS pin drive low. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 6–2 shows how to configure multiple devices with a configuration device. This circuit is similar to the configuration device circuit for a single device, except APEX II devices are cascaded for multi-device configuration.

Figure 6–2. Multi-Device PS Configuration Using an Enhanced Configuration Device



Notes to Figure 6–2:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

Enhanced configuration devices (EPC4, EPC8, and EPC16) cannot be cascaded.

When performing multi-device configuration, you must generate the configuration device's Programmer Object File (**.pof**) from each project's SRAM Object File (**.sof**). You can combine multiple SOFs using the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, *see Section II, Software Settings, in Volume 2*.

In multi-device PS configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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.

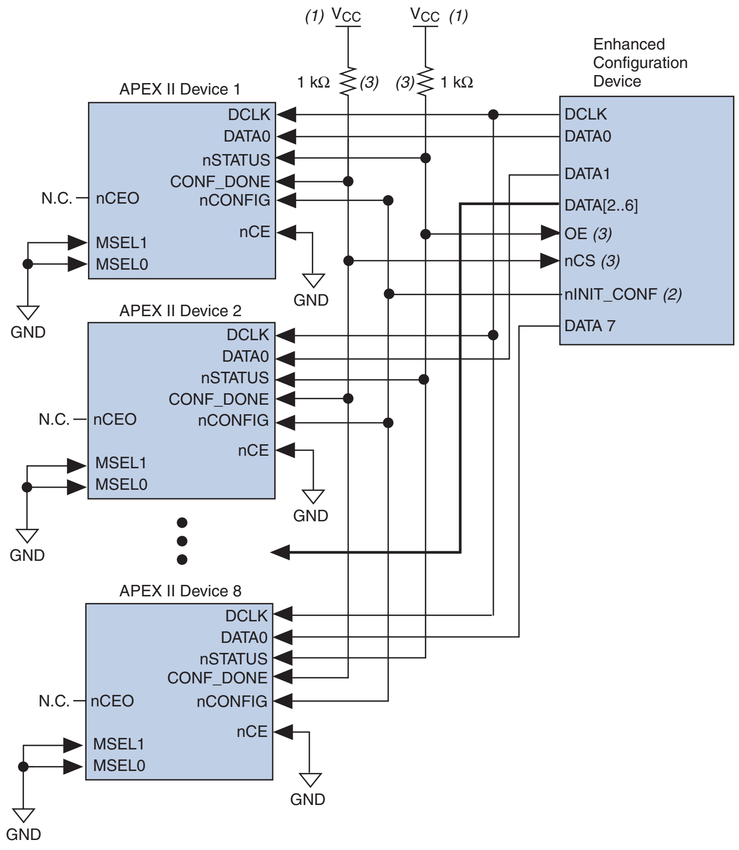
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the enhanced configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the *Auto-Restart Configuration After Error* option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

The enhanced configuration devices also support parallel configuration of up to eight devices. The n-bit (n = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure FPGAs or a chain of FPGAs. In addition, these devices do not have to be the same device family or density; they can be any combination of Altera FPGAs. An individual enhanced configuration device DATA line is available for each targeted FPGA. Each DATA line can also feed a daisy chain of FPGAs. Figure 6–3 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 6–3. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device



Notes to Figure 6–3:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

The Quartus II software only allows the selection of n-bit PS configuration modes, where n must be 1, 2, 4, or 8. 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 6–3 to select the appropriate configuration mode for the fastest configuration times.

Table 6–3. Recommended Configuration Using n-Bit PS Modes

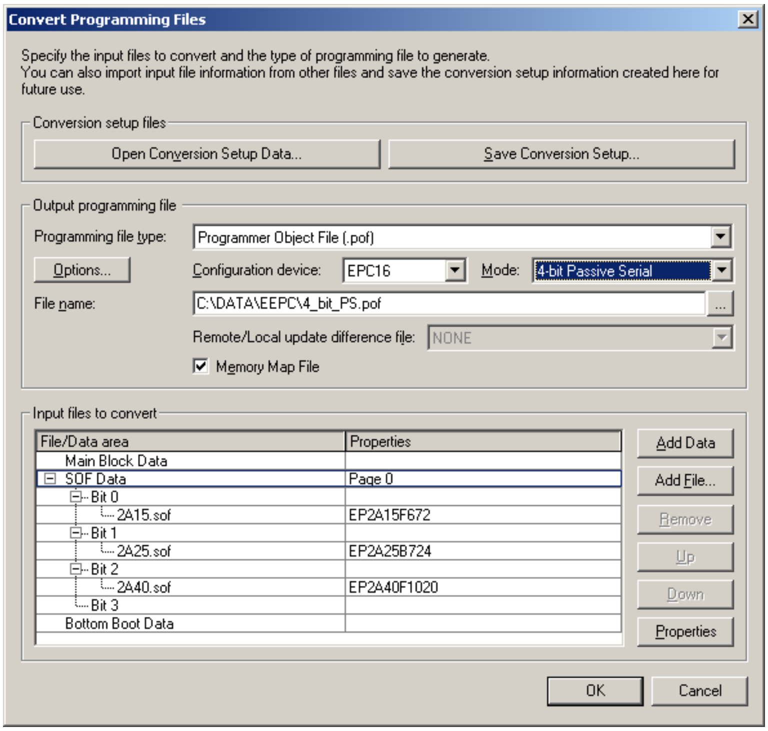
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 6–3:

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

For example, if you configure three FPGAs, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding SOF data is transmitted from the configuration device to the FPGA. 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 6–4 shows the Quartus II **Convert Programming Files** window (Tools menu) setup for this scheme.

Figure 6–4. Software Settings for Configuring Devices Using n-Bit PS Modes


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: 4-bit Passive Serial

File name: C:\DATA\EEPC\4_bit_PS.pof

Remote/Local update difference file: NONE

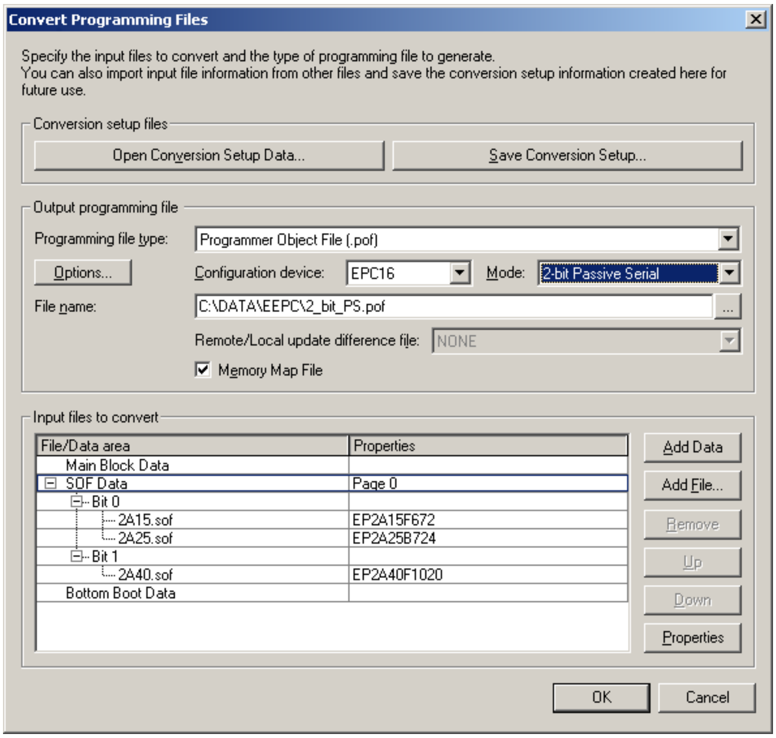
☑ Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
⊟ SOF Data	Page 0
⊟ Bit 0	
2A15.sof	EP2A15F672
⊟ Bit 1	
2A25.sof	EP2A25B724
⊟ Bit 2	
2A40.sof	EP2A40F1020
Bit 3	
Bottom Boot Data	

Alternatively, you can daisy chain two FPGAs 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 FPGAs with `DATA Bit0` (EP2A15 and EP2A25 devices) and the third device (the EP2A40 device) with `DATA Bit1`. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but can increase the total system configuration time. See Figure 6–5.

Figure 6–5. Software Settings for Daisy Chaining Two FPGAs on 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: C:\DATA\EEPC\2_bit_PS.pof

Remote/Local update difference file: NONE

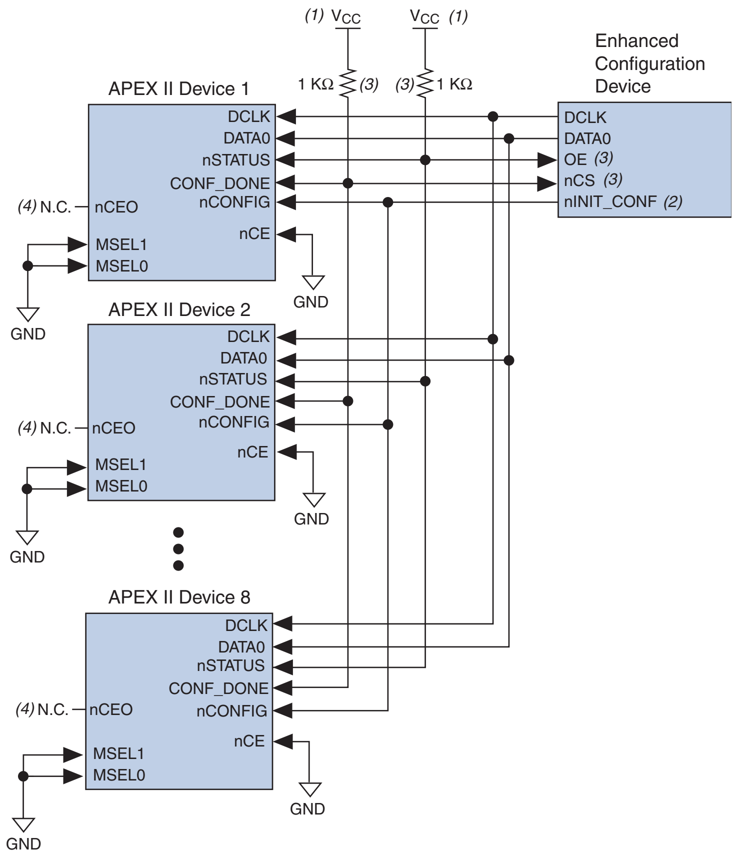
☑ Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
⊟ SOF Data	Page 0
⊟ Bit 0	
2A15.sof	EP2A15F672
2A25.sof	EP2A25B724
⊟ Bit 1	
2A40.sof	EP2A40F1020
Bottom Boot Data	

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 6–6 shows multi-device PS configuration when the APEX II devices are receiving the same configuration data.

Figure 6–6. Multiple-Device PS Configuration Using an Enhanced Configuration Device When FPGAs Receive the Same Data



Notes to Figure 6–6:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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) The `nCEO` pins of all devices are left unconnected when configuring the same configuration data into multiple devices.

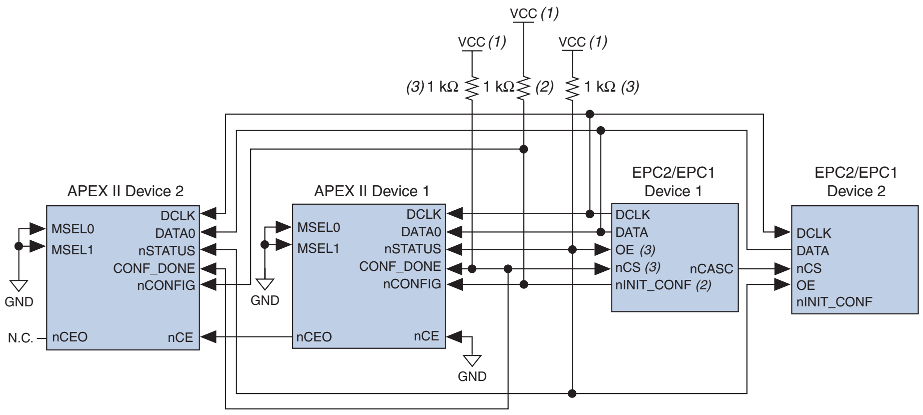
You can cascade several EPC2 or EPC1 devices to configure multiple APEX II devices. The first configuration device in the chain is the master configuration device, while the subsequent devices are the slave devices. The master configuration device sends DCLK to the APEX II devices and to the slave configuration devices. The first EPC device's nCS pin is connected to the CONF_DONE pins of the FPGAs, while its nCASC pin is connected to nCS of the next configuration device in the chain. The last device's nCS input comes from the previous device, while its nCASC pin is left floating. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the next configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

Enhanced configuration devices EPC4, EPC8, and EPC16 cannot be cascaded.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect CONF_DONE going high at the end of configuration, it resets the entire chain by pulling its OE pin low. This low signal drives the OE pin low on the slave configuration device(s) and drives nSTATUS low on all FPGAs, causing them to enter a reset state. This behavior is similar to the FPGA detecting an error in the configuration data.

Figure 6–7 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 6–7. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 6–7:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

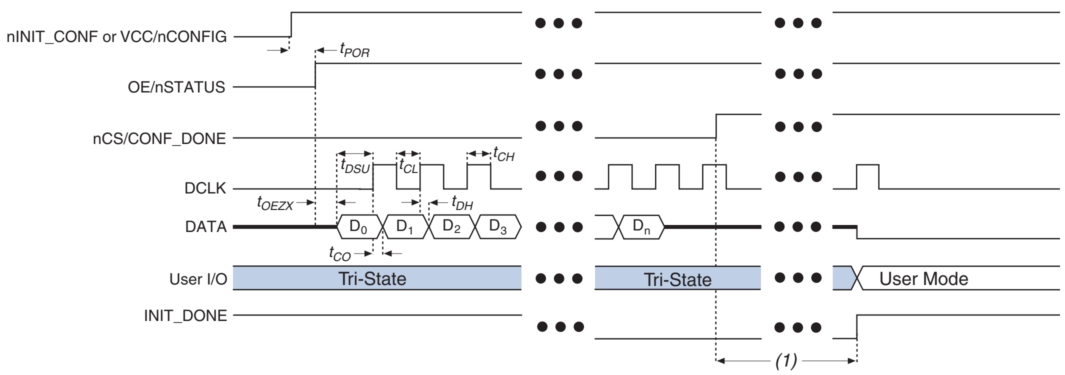
When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up resistor on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`. If multiple EPC2 devices are used to configure an APEX II device(s), only the first EPC2 has its `nINIT_CONF` pin tied to the device's `nCONFIG` pin.

You can use a single configuration chain to configure APEX II devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the *Configuration Handbook*.

Figure 6–8 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 6–8. APEX II PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 6–8:

(1) APEX II devices enter user-mode 40 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the APEX II internal oscillator or the `CLKUSR` pin.

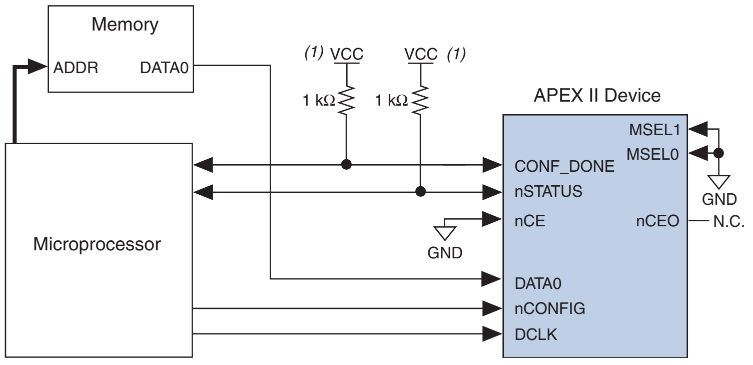
For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the *Configuration Handbook*.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the *Configuration Handbook*.

PS Configuration Using a Microprocessor

In the PS configuration scheme, an intelligent host (e.g., a microprocessor or CPLD) can transfer configuration data from a storage device (e.g., flash memory) to the target APEX II devices. Configuration data can be stored in RBF, HEX, or TTF format. Figure 6–9 shows the configuration interface connections between the APEX II device and a microprocessor for single device configuration.

Figure 6–9. Single Device PS Configuration Using a Microprocessor



Note to Figure 6–9:

(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device.

Upon power-up, the APEX II device goes through a POR for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

The configuration cycle consists of three stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the nCONFIG pin.

VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one bit at a time on the DATA0 pin. The least significant bit (LSB) of each data byte must be sent first.

The APEX II device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK. Data is continuously clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In APEX II devices, the initialization clock source is either the APEX II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX II device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, APEX II devices require 40 clock cycles to initialize properly.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. To ensure DCLK and DATA are not left floating at the end of configuration, the microprocessor must drive them either high or low, whichever is convenient on your board.

Handshaking signals are not used in PS configuration mode. Therefore, the configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the APEX II device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

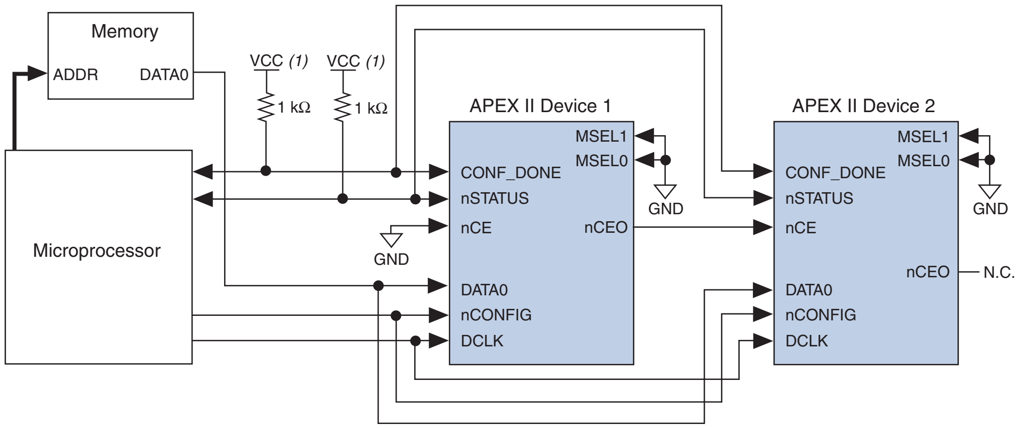
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, you can initiate a reconfiguration by transitioning the nCONFIG pin low-to-high. The nCONFIG pin must be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 6–10 shows how to configure multiple devices using a microprocessor. This circuit is similar to the PS configuration circuit for a single device, except APEX II devices are cascaded for multi-device configuration.

Figure 6–10. Multi-Device PS Configuration Using a Microprocessor



Note to Figure 6–10:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

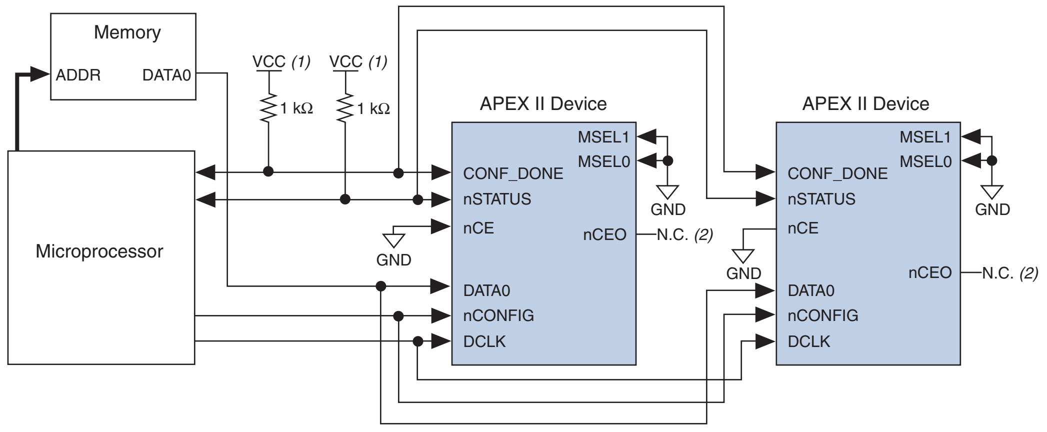
In multi-device PS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on `nSTATUS`, it resets the chain by pulling its `nSTATUS` pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you can have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 6–11 shows multi-device PS configuration when both APEX II devices are receiving the same configuration data.

Figure 6–11. Multiple-Device PS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 6–11:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

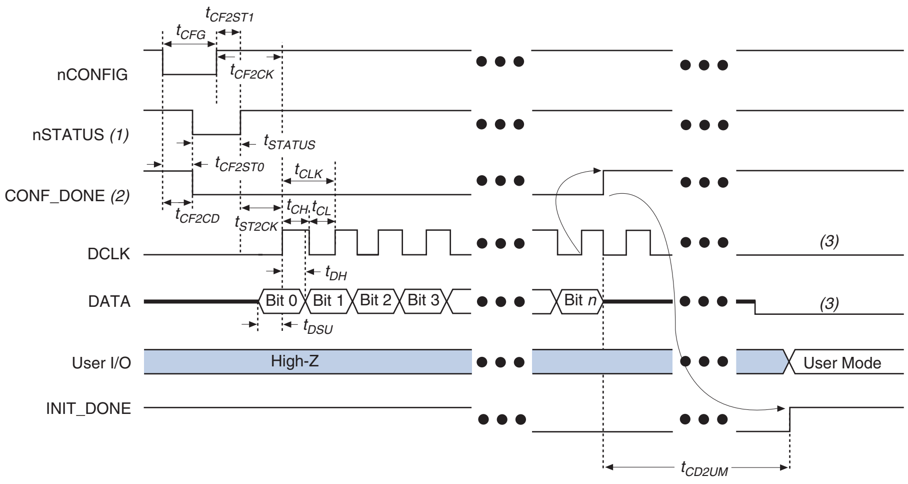
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX II devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the *Configuration Handbook*.

Figure 6–12 shows the timing waveform for the PS configuration for APEX II devices using a microprocessor.

Figure 6–12. APEX II PS Configuration Using a Microprocessor Timing Waveform



Notes to Figure 6–12:

(1) Upon power-up, the APEX II device holds nSTATUS low for not more than 5 µs after V_{CC} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, CONF_DONE is low.

(3) DATA0 and DCLK should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Table 6–4 defines the timing parameters for APEX II devices for PS configuration.

Table 6–4. PS Timing Parameters for APEX II Devices Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(2)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	7.5		ns
t_{CL}	`DCLK` low time	7.5		ns
t_{CLK}	`DCLK` period	15		ns
f_{MAX}	`DCLK` maximum frequency		66	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	2	8	µs

Notes to Table 6–4:
(1) This information is preliminary.
(2) This value is applicable if users do not delay configuration by extending the `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting the device. If the clock source is `CLKUSR`, multiply the clock period by 40 for APEX II devices to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the *Configuration Handbook*.

Configuring Using the MicroBlaster Driver

The MicroBlaster^{TM} software driver allows you to configure Altera's FPGAs through the ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a RBF programming input file and is targeted for embedded passive serial configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, go to the Altera web site (http://www.altera.com).

PS Configuration Using a Download Cable

In this section, the generic term “download cable” includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable.

In PS configuration with a download cable, an intelligent host (e.g., a PC) transfers data from a storage device to the FPGA via the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Upon power-up, the APEX II device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

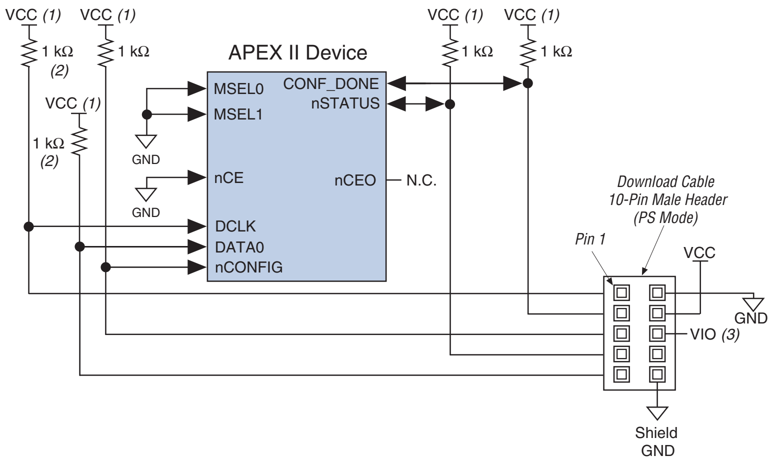
The configuration cycle consists of 3 stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration, and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. The programming hardware or download cable then places the configuration data one bit at a time on the device’s `DATA0` pin. The configuration data is clocked into the target device until `CONF_DONE` goes high.

When using a download cable, setting the *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs. Additionally, the *Enable user-supplied start-up clock (CLKUSR)* option has no affect on the device initialization since this option is disabled in the SOF when programming the FPGA using the Quartus II programmer and download cable. Therefore, if you turn on the `CLKUSR` option, you do not need to provide a clock on CLKUSR when you are configuring the FPGA with the Quartus II programmer and a download cable. Figure 6–13 shows PS configuration for APEX II devices using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cable.

Figure 6–13. PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 6–13:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (VIO pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

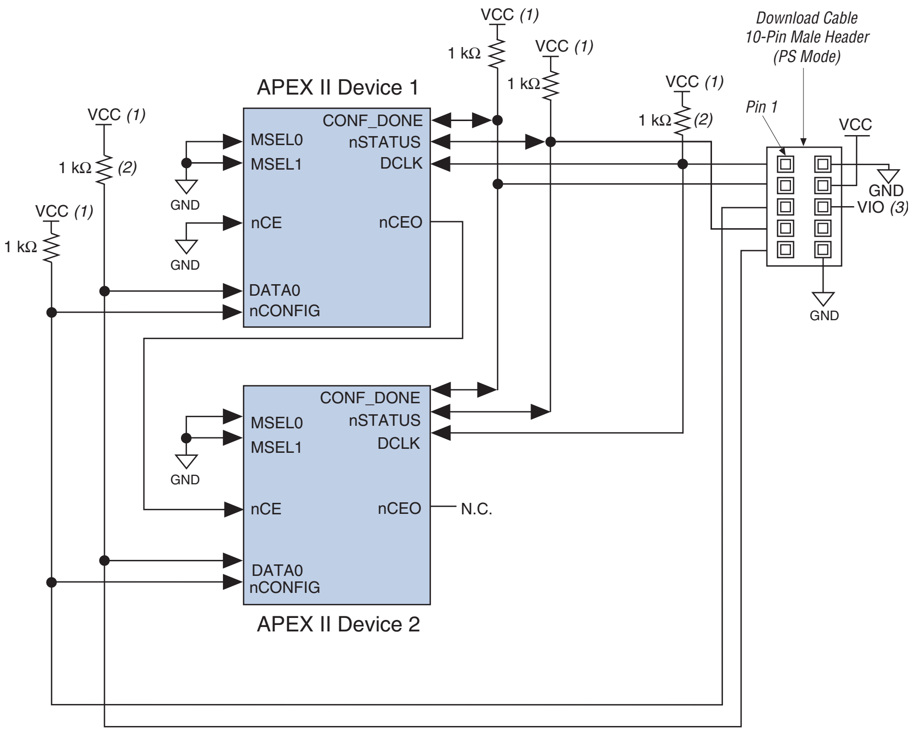
(3) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

You can use a download cable to configure multiple APEX II devices by connecting each device's nCEO pin to the subsequent device's nCE pin. The first device's nCE pin is connected to GND while its nCEO pin is connected to the nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. All other configuration pins, nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE are connected to every device in the chain. Because all CONF_DONE pins are tied together, all devices in the chain initialize and enter user mode at the same time.

In addition, because the nSTATUS pins are tied together, the entire chain halts configuration if any device detects an error. The *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs.

Figure 6–14 shows how to configure multiple APEX II devices with a download cable.

Figure 6–14. Multi-Device PS Configuration using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 6–14:

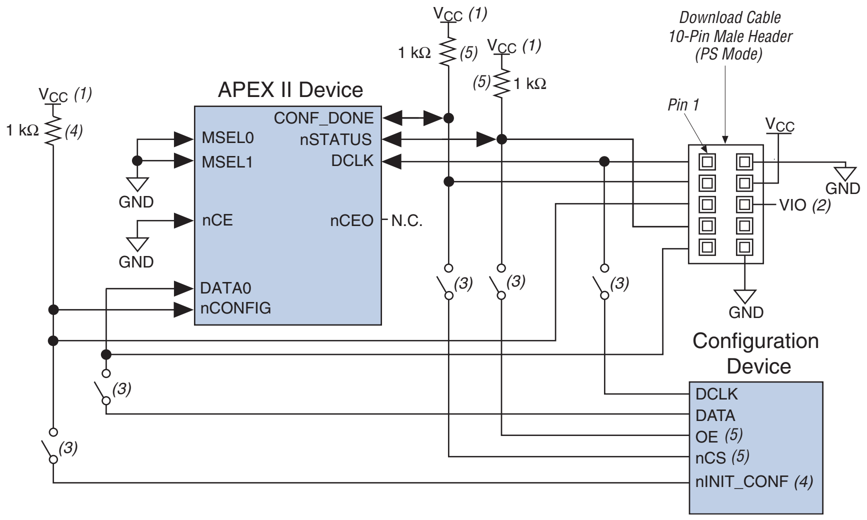
(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (VIO pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

(3) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

If you are using a download cable to configure device(s) on a board that also has configuration devices, you should electrically isolate the configuration device from the target device(s) and cable. One way to isolate the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer chip should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) between the cable and the configuration device. The last option is to remove the configuration device from the board when configuring the FPGA with the cable. Figure 6–15 shows a combination of a configuration device and a download cable to configure an FPGA.

Figure 6–15. PS Configuration with a Download Cable and Configuration Device Circuit



Notes to Figure 6–15:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to an APEX II device. Instead, 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.

(4) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) 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 (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} either directly or through a resistor.

(5) The enhanced configuration devices' and EPC2 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-up resistors on configuration device* option when generating programming files.

For more information on how to use the USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cables, refer to the following data sheets.

- 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

Fast Passive Parallel Configuration

Fast Passive Parallel (FPP) configuration in APEX II devices is designed to meet the continuously increasing demand for faster configuration times. APEX II devices are designed with the capability of receiving byte-wide configuration data per clock cycle, and guarantee a configuration time of less than 100 ms with a 66-MHz configuration clock.

FPP configuration of APEX II devices can be performed using an Altera enhanced configuration device or an intelligent host, such as a microprocessor.

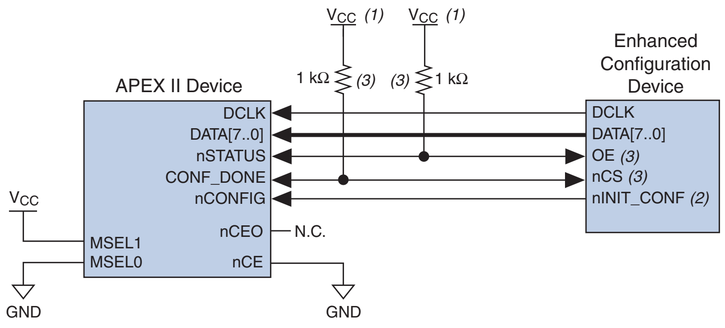
FPP Configuration Using an Enhanced Configuration Device

In the FPP configuration scheme, an enhanced configuration device sends a byte of configuration data every `DCLK` cycle to the APEX II device. Configuration data is stored in the configuration device. Figure 6–16 shows the configuration interface connections between the APEX II device and the enhanced configuration device for single device configuration.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on the enhanced configuration device and flash interface pins, such as `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`, refer to the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* in the Configuration Handbook.

Figure 6–16. Single Device FPP Configuration Using an Enhanced Configuration Device



Notes to Figure 6–16:

(1) The pull-up resistor 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.

The value of the internal pull-up resistors on the enhanced configuration devices can be found in the Operating Conditions table of the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* in the Configuration Handbook.

When using enhanced configuration devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. 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. An internal pull-up on the `nINIT_CONF` pin is always active in the enhanced configuration devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`.

Upon power-up, the APEX II device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for enhanced configuration devices can be set to either 100 ms or 2 ms, depending on its PORSEL pin setting. If the PORSEL pin is connected to GND, the POR delay is 100 ms. During this time, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When both devices complete POR, they release their open-drain OE or nSTATUS pin, which is then pulled high by a pull-up resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of 3 stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. The beginning of configuration can be delayed by holding the nCONFIG or nSTATUS pin low.

VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the nSTATUS pin, which is pulled high by a pull-up resistor. Enhanced configuration devices have an optional internal pull-up on the OE pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up is not used, an external 1-kΩ pull-up on the OE/nSTATUS line is required. Once nSTATUS is released the FPGA is ready to receive configuration data and the configuration stage begins.

When nSTATUS is pulled high, OE of the configuration device also goes high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The APEX II device receives configuration data on its DATA[7..0] pins and the clock is received on the DCLK pin. A byte of data is latched into the FPGA on the rising edge of DCLK.

After the FPGA has received all configuration data successfully it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration devices have an optional internal pull-up on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up is not used, an external 1kΩ pull-up on the nCS/CONF_DONE line is required. A low to high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In APEX II devices, the initialization clock source is either the APEX II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX II device will take care to provide itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, APEX II devices require 40 clock cycles to initialize properly.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. The enhanced configuration device will drive DCLK low and DATA[7..0] high at the end of configuration.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA will automatically initiate reconfiguration if an error occurs. The APEX II device will release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG, if nCONFIG is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the FPGA has not configured successfully. Enhanced configuration devices wait for 64 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. If the *Auto-Restart Configuration After Error* option is set in the software, the target device resets and then release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns high, the configuration device will try to reconfigure the FPGA.

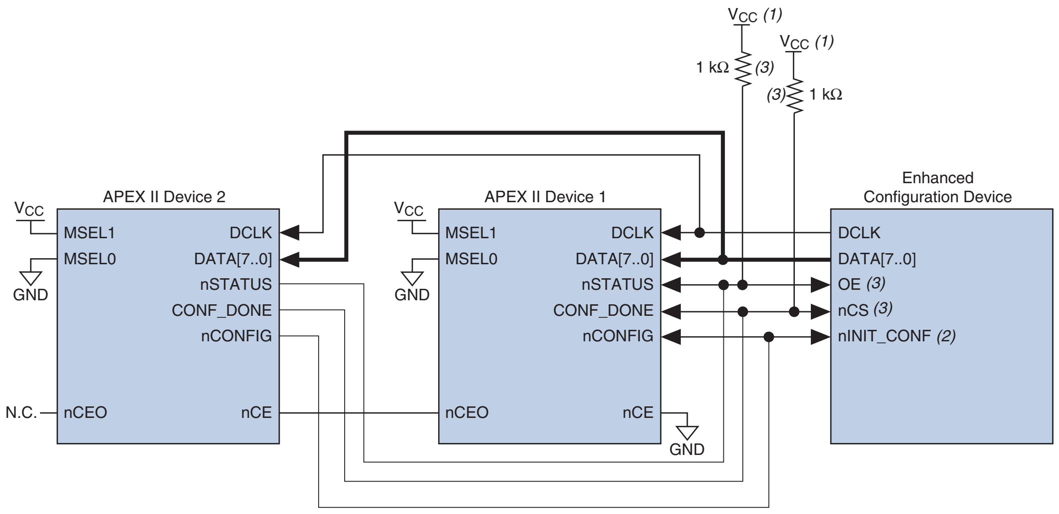
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully; therefore, your system should not pull CONF_DONE low to delay initialization. Instead, you should use the *CLKUSR* option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this will activate the configuration device since it will see its nCS pin drive low. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 6–17 shows how to configure multiple APEX II devices with an enhanced configuration device. This circuit is similar to the configuration device circuit for a single device, except the APEX II devices are cascaded for multi-device configuration.

Figure 6–17. Multi-Device FPP Configuration Using an Enhanced Configuration Device



Notes to Figure 6–17:

(1) The pull-up resistor 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-up resistors on configuration device* option when generating programming files.

Enhanced configuration devices (EPC4/8/16) cannot be cascaded.

When performing multi-device configuration, you must generate the configuration device's POF from each project's SOF. You can combine multiple SOFs using the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, *see Section II, Software Settings, in Volume 2*.

In multi-device FPP configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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.

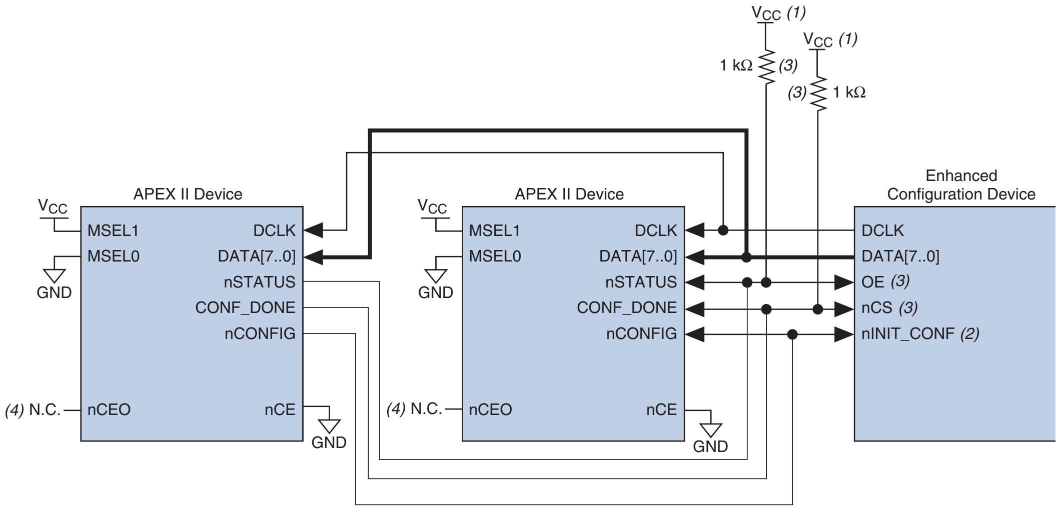
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the enhanced configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the *Auto-Restart Configuration After Error* option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 6–18 shows multi-device FPP configuration when both APEX II devices are receiving the same configuration data.

Figure 6–18. Multiple-Device FPP Configuration Using an Enhanced Configuration Device When Both FPGAs Receive the Same Data



Notes to Figure 6–18:

(1) The pull-up resistor 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.

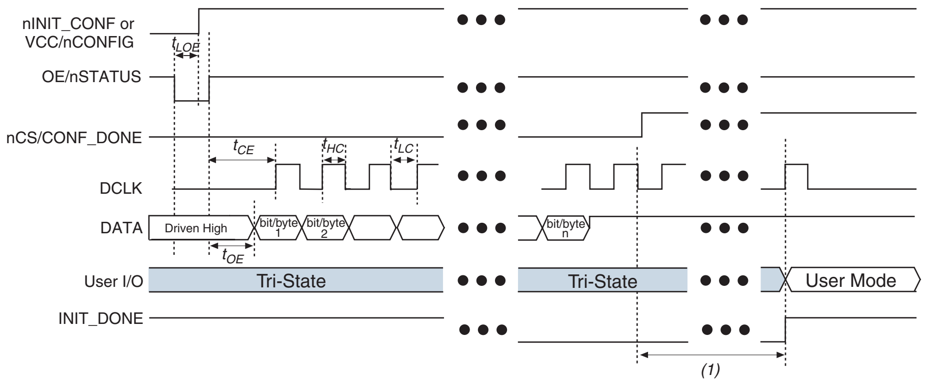
(4) The `nCEO` pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single enhanced configuration chain to configure multiple APEX II devices with other Altera devices that support FPP configuration, such as Stratix® and Stratix GX devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 6–19 shows the timing waveform for the FPP configuration scheme using an enhanced configuration device.

Figure 6–19. APEX II FPP Configuration Using an Enhanced Configuration Device Timing Waveform



Note to Figure 6–19:

(1) APEX II devices enter user mode 40 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the APEX II internal oscillator or the `CLKUSR` pin.

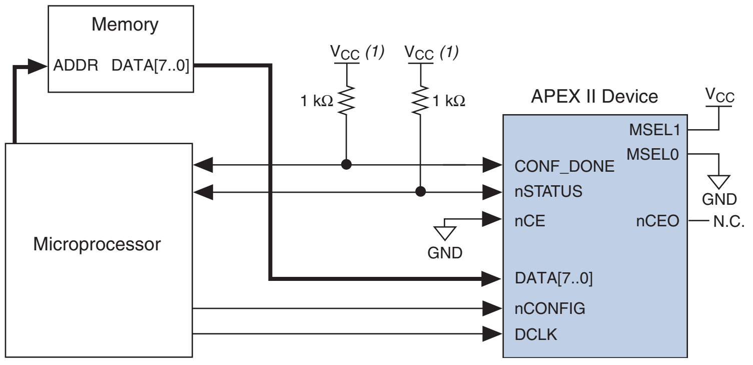
For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* in the Configuration Handbook.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the Configuration Handbook.

FPP Configuration Using a Microprocessor

In the FPP configuration scheme, an intelligent host, such as a microprocessor or CPLD, can transfer configuration data from a storage device, such as flash memory, to the target APEX II device. Configuration data can be stored in RBF, HEX or TTF format. Figure 6–20 shows the configuration interface connections between the APEX II device and a microprocessor for single device configuration.

Figure 6–20. Single Device FPP Configuration Using a Microprocessor



Note to Figure 6–20:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

Upon power-up, the APEX II device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

The configuration cycle consists of three stages: reset, configuration and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the nCONFIG pin.

VCCINT and VCCIO pins on the banks where the configuration, and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one byte at a time on the DATA[7..0] pins.

The APEX II device receives configuration data on its `DATA[7..0]` pins and the clock is received on the `DCLK` pin. Data is latched into the FPGA on the rising edge of `DCLK`. Data is continuously clocked into the target device until `CONF_DONE` goes high. After the FPGA has received all configuration data successfully, it releases the open-drain `CONF_DONE` pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on `CONF_DONE` indicates configuration is complete and initialization of the device can begin.

In APEX II devices, the initialization clock source is either the APEX II internal oscillator (typically 10 MHz) or the optional `CLKUSR` pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX II device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving `DCLK` to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the `CLKUSR` option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on `CLKUSR` will not affect the configuration process. After all configuration data has been accepted and `CONF_DONE` goes high, APEX II devices require 40 clock cycles to initialize properly.

An optional `INIT_DONE` pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the `INIT_DONE` pin is used it will be high due to an external 1-kΩ pull-up resistor when `nCONFIG` is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DCLK and DATA0 are not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. The DATA[7..1] pins are available as user I/O pins after configuration. When the FPP scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

Handshaking signals are not used in FPP configuration mode. Therefore, the configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

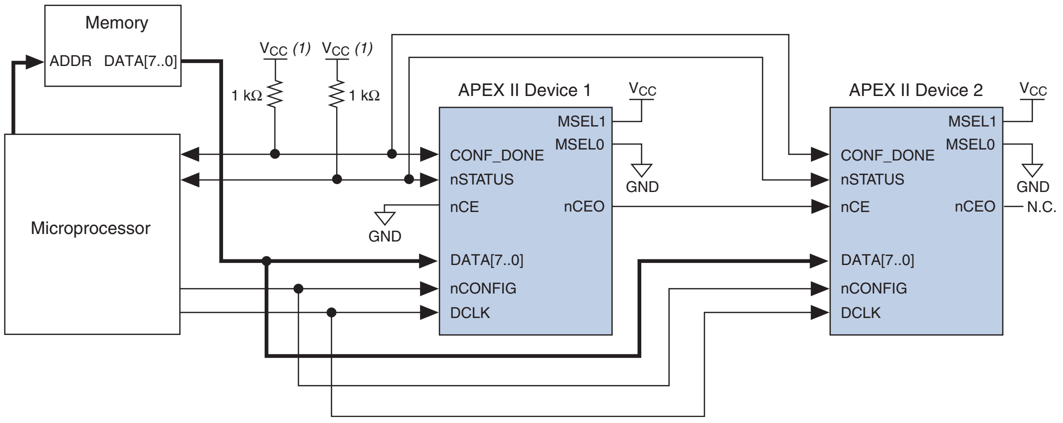
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 6–21 shows how to configure multiple devices using a microprocessor. This circuit is similar to the FPP configuration circuit for a single device, except the APEX II devices are cascaded for multi-device configuration.

Figure 6–21. Multi-Device FPP Configuration Using a Microprocessor



Note to Figure 6–21:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

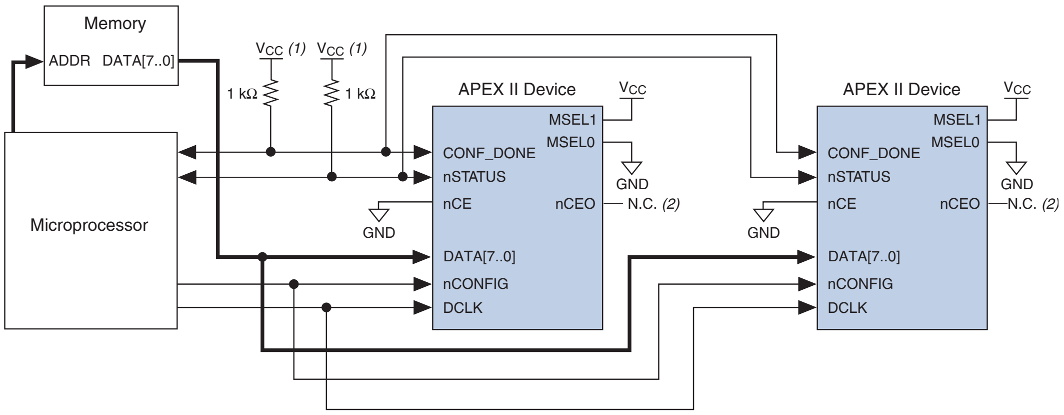
In multi-device FPP configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 6–22 shows multi-device FPP configuration when both APEX II devices are receiving the same configuration data.

Figure 6–22. Multiple-Device FPP Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 6–22:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

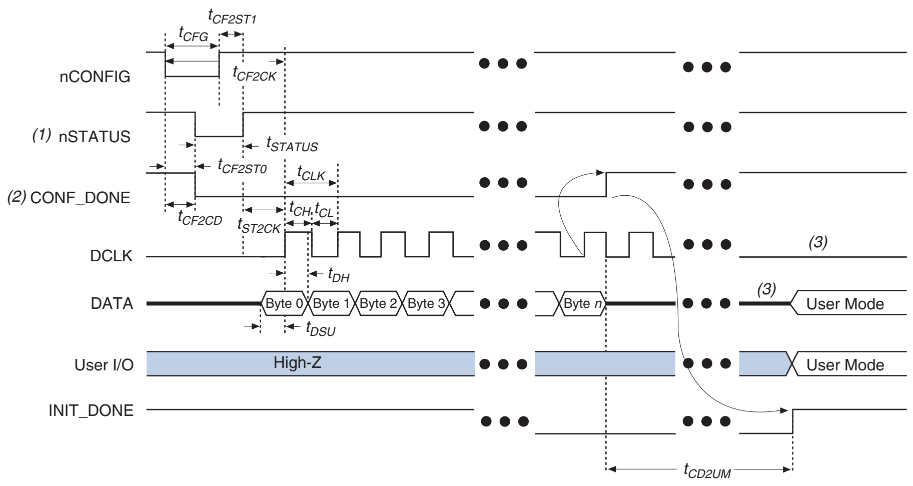
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX II devices with other Altera devices that support FPP configuration, such as Stratix. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 6–23 shows the timing waveform for the FPP configuration scheme using a microprocessor.

Figure 6–23. APEX II FPP Configuration Using a Microprocessor Timing Waveform



Notes to Figure 6–23:

(1) Upon power-up, the APEX II device holds nSTATUS low for not more than 5 µs after V_{CC} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, CONF_DONE is low.

(3) DATA0 and DCLK should not be left floating after configuration. It should be driven high or low, whichever is more convenient. DATA[7..1] are available as user I/O pins after configuration and the state of theses pins depends on the design programmed into the device.

Table 6–5 defines the timing parameters for APEX II devices for FPP configuration.

Table 6–5. FPP Timing Parameters for APEX II Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	8		µs
t_{STATUS}	nSTATUS low pulse width	10	40 (2)	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		1 (2)	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	40		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns

Table 6–5. FPP Timing Parameters for APEX II Devices (Part 2 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CH}	DCLK high time	7.5		ns
t_{CL}	DCLK low time	7.5		ns
t_{CLK}	DCLK period	15		ns
f_{MAX}	DCLK frequency		66	MHz
t_{CD2UM}	CONF_DONE high to user mode (3)	2	8	µs

Notes to Table 6–5:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 40 to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the Configuration Handbook.

Configuring Using the MicroBlaster Driver

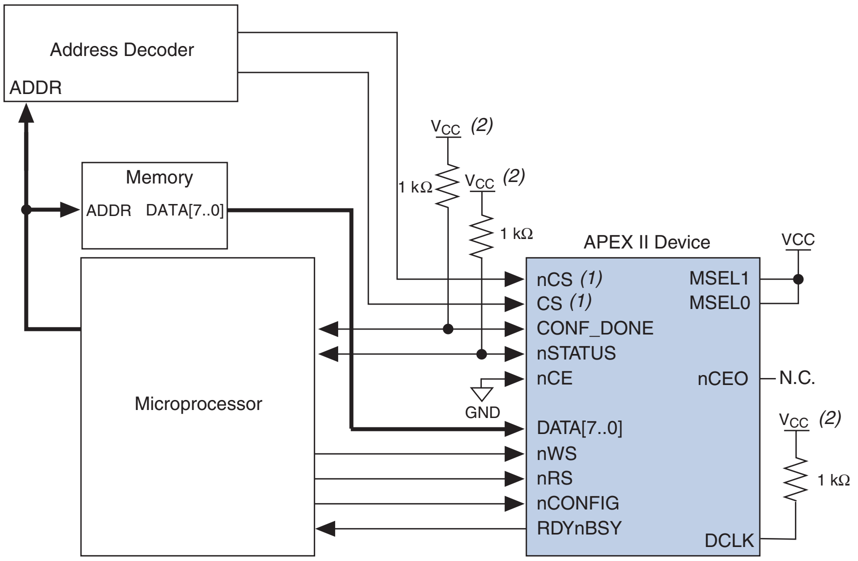
The MicroBlaster^{TM} software driver supports a RBF programming input file and is targeted for embedded fast passive parallel configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, go to the Altera web site (http://www.altera.com).

Passive Parallel Asynchronous Configuration

Passive Parallel Asynchronous (PPA) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target APEX II device. Configuration data can be stored in TTF, RBF or HEX format. The host system outputs byte-wide data and the accompanying strobe signals to the FPGA. When using PPA, you should pull the DCLK pin high through a 1-kΩ pull-up resistor to prevent unused configuration input pins from floating.

Figure 6–24 shows the configuration interface connections between the FPGA and a microprocessor for single device PPA configuration. The microprocessor or an optional address decoder can control the device's chip select pins, nCS and CS. The address decoder allows the microprocessor to select the APEX II device by accessing a particular address, which simplifies the configuration process. The nCS and CS pins must be held active during configuration and initialization.

Figure 6–24. Single Device PPA Configuration Using a Microprocessor Note (1)



Notes to Figure 6–24:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

During PPA configuration, it is only required to use either the nCS or CS pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, nCS can be tied to ground while CS is toggled to control configuration. The device's nCS or CS pins can be toggled during PPA configuration if the design meets the specifications set for t_{CSSU}, t_{WSP}, and t_{CSH} listed in Table 6–6.

Upon power-up, the APEX II device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX II devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX II Programmable Logic Device Family Data Sheet*.

The configuration cycle consists of three stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

☞ `VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should then assert the target device's `nCS` pin low and/or `CS` pin high. Next, the microprocessor places an 8-bit configuration word (one byte) on the target device's `DATA[7..0]` pins and pulses the `nWS` pin low.

On the rising edge of `nWS`, the target device latches in a byte of configuration data and drives its `RDYnBSY` signal low, which indicates it is processing the byte of configuration data. The microprocessor can then perform other system functions while the APEX II device is processing the byte of configuration data.

During the time `RDYnBSY` is low, the APEX II device internally processes the configuration data using its internal oscillator (typically 10 MHz). When the device is ready for the next byte of configuration data, it will drive `RDYnBSY` high. If the microprocessor senses a high signal when it polls `RDYnBSY`, the microprocessor sends the next byte of configuration data to the FPGA.

Alternatively, the `nRS` signal can be strobed low, causing the `RDYnBSY` signal to appear on `DATA7`. Because `RDYnBSY` does not need to be monitored, this pin doesn't need to be connected to the microprocessor. Data should not be driven onto the data bus while `nRS` is low because it will cause contention on the `DATA7` pin. If the `nRS` pin is not used to monitor configuration, it should be tied high.

To simplify configuration and save an I/O port, the microprocessor can wait for the total time of t_{BUSY}(max) + t_{RDY2WS} + t_{W2SB} before sending the next data byte. In this set-up, `nRS` should be tied high and `RDYnBSY` does not need to be connected to the microprocessor. The t_{BUSY}, t_{RDY2WS} and t_{W2SB} timing specifications are listed in Table 6–6.

Next, the microprocessor checks nSTATUS and CONF_DONE. If nSTATUS is not low and CONF_DONE is not high, the microprocessor sends the next data byte. However, if nSTATUS is not low and all the configuration data has been received, the device is ready for initialization. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In APEX II devices, the initialization clock source is either the APEX II internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX II device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, APEX II devices require 40 clock cycles to initialize properly.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DATA0 is not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. After configuration, the nCS, CS, nRS, nWS, RDYnBSY, and DATA[7..1] pins can be used as user I/O pins. When the PPA scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

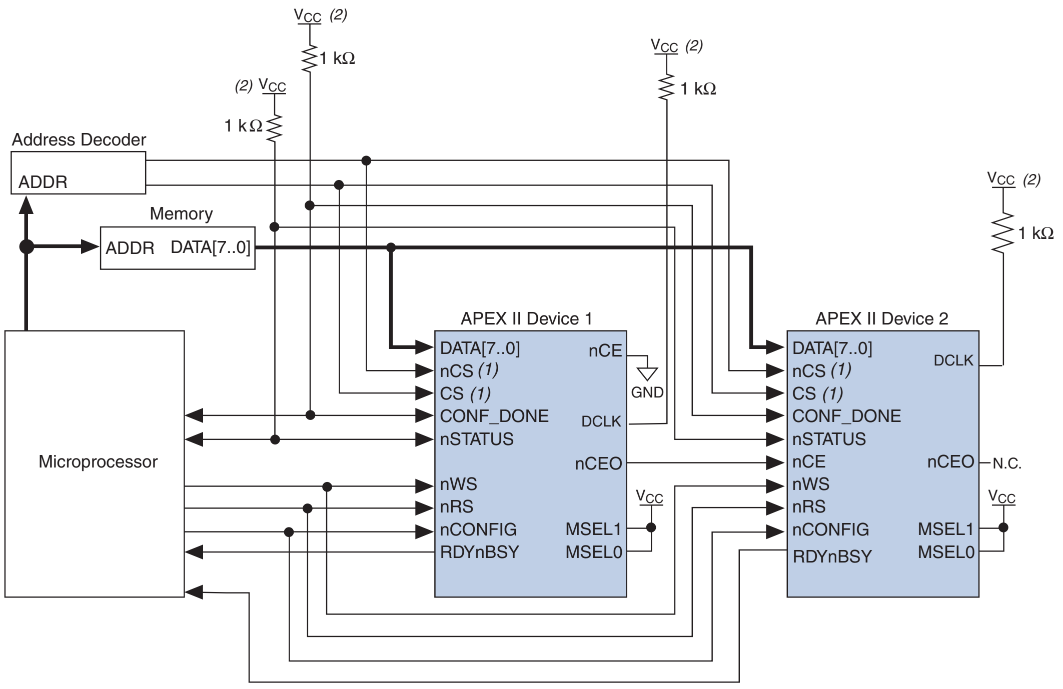
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE has not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 6–25 shows how to configure multiple APEX II devices using a microprocessor. This circuit is similar to the PPA configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 6–25. Multi-Device PPA Configuration Using a Microprocessor



Notes to Figure 6–25:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPA configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Each device's RDYnBSY pin can have a separate input to the microprocessor. Alternatively, if the microprocessor is pin limited, all the RDYnBSY pins can feed an AND gate and the output of the AND gate can feed the microprocessor. For example, if you have 2 devices in a PPA configuration chain, the second device's RDYnBSY pin will be high during the time that the first device is being configured. When the first device has been successfully configured, it will driven nCEO low to activate the next device in the chain and drive its RDYnBSY pin high. Therefore, since RDYnBSY signal is driven high before configuration and after configuration before entering user-mode, the device being configured will govern the output of the AND gate.

The nRS signal can be used in multi-device PPA chain since the APEX II device will tri-state its DATA[7..0] pins before configuration and after configuration before entering user-mode to avoid contention. Therefore, only the device that is currently being configured will respond to the nRS strobe by asserting DATA7.

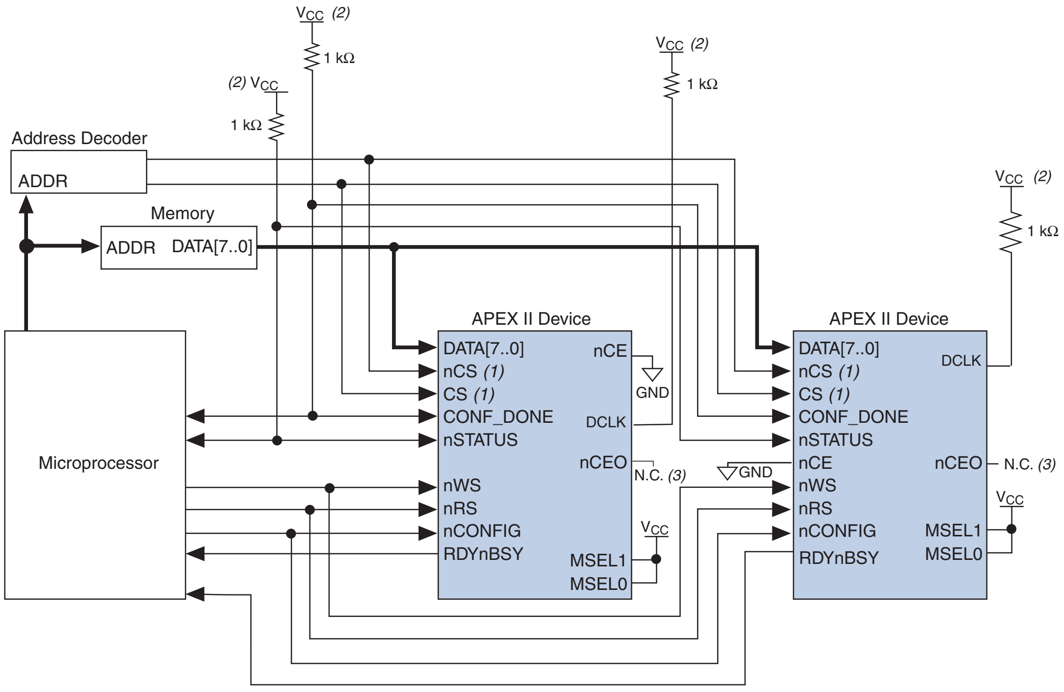
All other configuration pins (nCONFIG, nSTATUS, DATA[7..0], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DATA[7..1], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DATA lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 6–26 shows multi-device PPA configuration when both devices are receiving the same configuration data.

Figure 6–26. Multiple-Device PPA Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 6–26:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

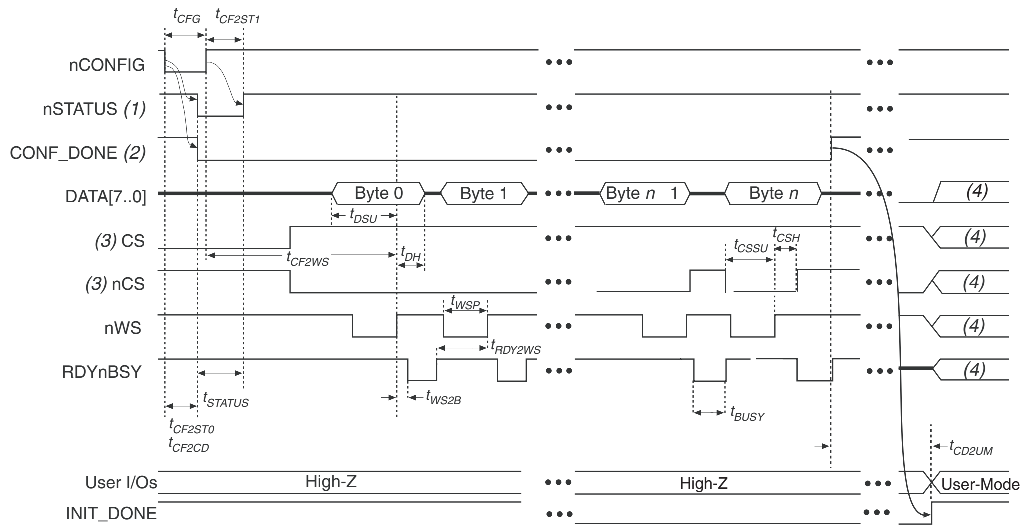
(3) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX II devices with other Altera devices that support PPA configuration, such as Stratix, Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 6–27 shows the timing waveform for the PPA configuration scheme using a microprocessor.

Figure 6–27. APEX II PPA Configuration Timing Waveform



Notes to Figure 6–27:

(1) Upon power-up, the APEX II device holds `nSTATUS` low for not more than 5 µs after V_{CCINT} reaches its minimum requirement.

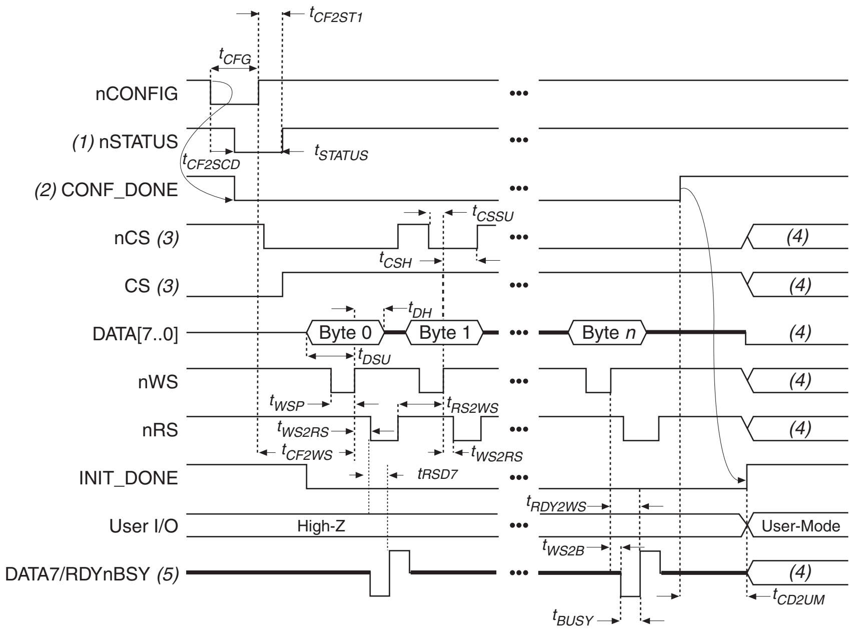
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS` and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

Figure 6–28 shows the timing waveform for the PPA configuration scheme when using a strobed `nRS` and `nWS` signal.

Figure 6–28. APEX II PPA Configuration Timing Waveform Using nRS & nWS



Notes to Figure 6–28:

(1) Upon power-up, the APEX II device holds `nSTATUS` low for not more than 5 µs after V_{CCINT} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS`, and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

(5) `DATA7` is a bidirectional pin. It is an input for configuration data input, but it is an output to show the status of `RDYnBSY`.

Table 6–6 defines the timing parameters for APEX II devices for PPA configuration.

Table 6–6. PPA Timing Parameters for APEX II Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 (2)	µs

Table 6–6. PPA Timing Parameters for APEX II Devices (Part 2 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 (2)	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.1	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode (3)	2	8	µs

Notes to Table 6–6:

(1) This information is preliminary.

(2) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.

(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 for APEX II devices to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings*, *Volume 2* of the Configuration Handbook.

JTAG Configuration

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The JTAG circuitry can also be used to shift configuration data into the device.

For more information on JTAG boundary-scan testing, see *Application Note 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing* in Altera Devices.

A device operating in JTAG mode uses four required pins, `TDI`, `TDO`, `TMS`, and `TCK`, and one optional pin, `TRST`. All user I/O pins are tri-stated during JTAG configuration. APEX II devices are designed such that JTAG instructions have precedence over any device configuration modes. This means that JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of APEX II FPGAs during PS configuration, PS configuration will be terminated and JTAG configuration will begin.

Table 6–7 explains each JTAG pin's function.

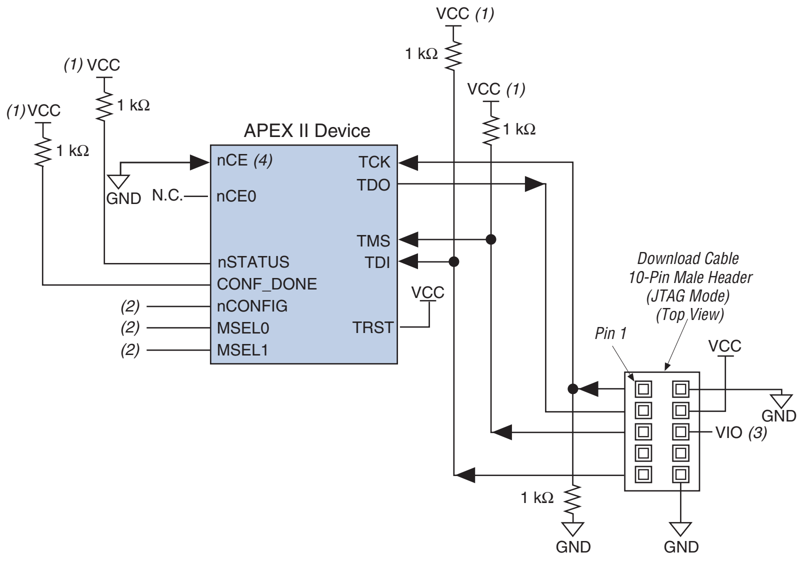
Table 6–7. JTAG Pin Descriptions

Pin	Description	Function
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

If V_{CCIO} of the bank where the JTAG pins reside, are tied to 3.3-V, both the I/O pins and JTAG `TDO` port will drive at 3.3-V levels.

During JTAG configuration, data can be downloaded to the device on the PCB through the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV header. Configuring devices through a cable is similar to programming devices in-system, except the `TRST` pin should be connected to V_{CC}. This ensures that the TAP controller is not reset. Figure 6–29. shows JTAG configuration of a single APEX II device.

Figure 6–29. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 6–29:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

To configure a single device in a JTAG chain, the programming software places all other devices in BYPASS mode. In BYPASS mode, devices pass programming data from the `TDI` pin to the `TDO` pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the `TDO` pin one clock cycle later.

APEX II devices have dedicated JTAG pins that always function as JTAG pins. JTAG testing can be performed on APEX II devices both before and after configuration, but not during configuration. The chip-wide reset (`DEV_CLRn`) and chip-wide output enable (`DEV_OE`) pins on APEX II devices do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of APEX II devices, the dedicated configuration pins should be considered. Table 6–8 shows how these pins should be connected during JTAG configuration.

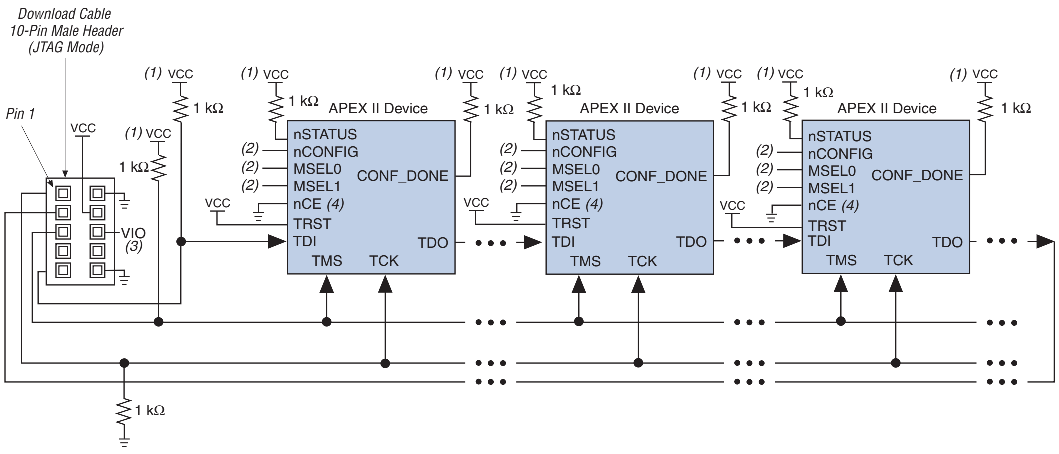
Table 6–8. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all APEX II devices in the chain, `nCE` should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, FPP, or PPA configuration chains, the `nCE` pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all APEX II devices in the chain, `nCEO` can be left floating or connected to the `nCE` of the next device. See `nCE` description above.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie both pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling high via a resistor, or driven by some control circuitry.
nSTATUS	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `nSTATUS` pin should be pulled up to V_{CC} individually. `nSTATUS` pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `CONF_DONE` pin should be pulled up to V_{CC} individually. `CONF_DONE` going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.
DATA0	Should not be left floating. Drive low or high, whichever is more convenient on your board.

When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. When four or more devices are connected in a JTAG chain, Altera recommends buffering the `TCK`, `TDI`, and `TMS` pins with an on-board buffer.

JTAG-chain device programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 6–30 shows multi-device JTAG configuration.

Figure 6–30. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 6–30:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to GND or driven low during JTAG configuration. In multi-device PS, FPP, and PPA configuration chains, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device will drive `nCE` of the next device low when it has successfully been JTAG configured.

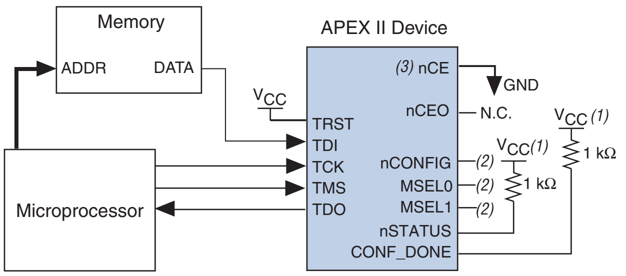
Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

For more information about configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, the software checks the state of `CONF_DONE` through the JTAG port. If `CONF_DONE` is not high, the Quartus II software indicates that configuration has failed. If `CONF_DONE` is high, the software indicates that configuration was successful. When Quartus II generates a JAM file for a multi-device chain, it contains instructions so that all the devices in the chain will be initialized at the same time.

Figure 6–31 shows JTAG configuration of an APEX II FPGA with a microprocessor.

Figure 6–31. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 6–31:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

(2) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

(3) `nCE` must be connected to GND or driven low for successful JTAG configuration.

Jam STAPL

Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, see *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the jam player, visit the Altera web site:

www.altera.com/support/software/download/programming/jam/jam-index.jsp

Configuring APEX II FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including APEX II FPGAs, through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in RBF format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on other platforms.

For more information on the JRunner software driver, see the *JRunner Software Driver: An Embedded Solution to the JTAG Configuration White Paper* and the source files.

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the APEX II device. Table 6–9 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 6–9. Dedicated Configuration Pins on the APEX II Device (Part 1 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
MSEL0 MSEL1	N/A	All	Input	Two-bit configuration input that sets the APEX II device configuration scheme. See Table 6–3 for the appropriate connections. These pins must remain at a valid state during power-up, before `nCONFIG` is pulled low to initiate a reconfiguration and during configuration.
VCCSEL	N/A	All	Input	Dedicated input that ensures the configuration related I/O banks have powered up to the appropriate 1.8-V or 2.5-V/3.3-V voltage levels before starting configuration. A logic high (1.5 V, 1.8 V, 2.5 V, 3.3 V) means 1.8 V, and a logic low means 2.5-V/3.3 V.

Table 6–9. Dedicated Configuration Pins on the APEX II Device (Part 2 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCONFIG	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode will cause the FPGA to lose its configuration data, enter a reset state, tri-state all I/O pins, and returning this pin to a logic high level will initiate a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, `nCONFIG` can be tied directly to V_{CC} or to the configuration device's `nINIT_CONF` pin.
nSTATUS	N/A	All	Bidirectional open-drain	The FPGA drives `nSTATUS` low immediately after power-up and releases it within 5 µs. (When using a configuration device, the configuration device holds `nSTATUS` low for up to 200 ms.) Status output. If an error occurs during configuration, `nSTATUS` is pulled low by the target device. Status input. If an external source drives the `nSTATUS` pin low during configuration or initialization, the target device enters an error state. Driving `nSTATUS` low after configuration and initialization does not affect the configured device. If a configuration device is used, driving `nSTATUS` low will cause the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on `nSTATUS` in user-mode, the FPGA will not reconfigure. To initiate a reconfiguration, `nCONFIG` must be pulled low. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.
CONF_DONE	N/A	All	Bidirectional open-drain	Status output. The target FPGA drives the `CONF_DONE` pin low before and during configuration. Once all configuration data is received without error and the initialization cycle starts, the target device releases `CONF_DONE`. Status input. After all data is received and `CONF_DONE` goes high, the target device initializes and enters user mode. Driving `CONF_DONE` low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.

Table 6–9. Dedicated Configuration Pins on the APEX II Device (Part 3 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCE	N/A	All	Input	Active-low chip enable. The `nCE` pin activates the device with a low signal to allow configuration. The `nCE` pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, `nCE` of the first device is tied low while its `nCEO` pin is connected to `nCE` of the next device in the chain. The `nCE` pin must also be held low for successful JTAG programming of the FPGA.
nCEO	N/A	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's `nCE` pin. The `nCEO` of the last device in the chain is left floating.
DCLK	N/A	Synchronous configuration schemes (PS and FPP)	Input	Clock input used to clock data from an external source into the target device. Data is latched into the FPGA on the rising edge of `DCLK`. In PPA mode, `DCLK` should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, `DCLK` will be driven low after configuration is done. In schemes that use a control host, `DCLK` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA0	N/A	All	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the `DATA0` pin. After configuration, EPC1 and EPC1441 devices tri-state this pin, while EPC2 devices drive this pin high. In schemes that use a control host, `DATA0` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA[7..1]	I/O	Parallel configuration schemes (FPP and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on `DATA[7..0]`. In serial configuration schemes, they function as user I/O pins during configuration, which means they are tri-stated. After PPA or FPP configuration, `DATA[7..1]` are available as a user I/O pins and the state of these pin depends on the Dual-Purpose Pin settings.

Table 6–9. Dedicated Configuration Pins on the APEX II Device (Part 4 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
DATA7	I/O	PPA	Bidirectional	In the PPA configuration scheme, the `DATA7` pin presents the `RDYnBSY` signal after the `nRS` signal has been strobed low. In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `DATA7` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nWS	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the `DATA[7..0]` pins. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nWS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nRS	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the `RDYnBSY` signal on the `DATA7` pin. If the `nRS` pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nRS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.

Table 6–9. Dedicated Configuration Pins on the APEX II Device (Part 5 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
RDYnBSY	I/O	PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPA configuration schemes, this pin will drive out high after power-up, before configuration and after configuration before entering user-mode. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `RDYnBSY` is available as a user I/O and the state of this pin depends on the dual-purpose pin settings.
nCS/CS	I/O	PPA	Input	Chip-select inputs. A low on `nCS` and a high on `CS` select the target device for configuration. The `nCS` and `CS` pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the `nCS` or `CS` pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, `nCS` can be tied to ground while `CS` is toggled to control configuration. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nCS` and `CS` are available as a user I/O pins and the state of these pins depends on the dual-purpose pin settings.

Table 6–10 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration they function as user I/O pins, which means they are tri-stated with weak pull-up resistors.

Table 6–10. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
CLKUSR	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software
INIT_DONE	N/A if option is on. I/O if option is off.	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 1-kΩ pull-up resistor. 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 and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the *Enable INIT_DONE output* option in the Quartus II software.
DEV_OE	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the *Enable device-wide output enable (DEV_OE)* option in the Quartus II software.
DEV_CLRn	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the *Enable device-wide reset (DEV_CLRn)* option in the Quartus II software.

JTAG pins must be kept stable before and during configuration. JTAG pin stability prevents accidental loading of JTAG instructions. Table 6–11 describes the dedicated JTAG pins.

Table 6–11. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
TDI	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

7. Configuring APEX 20KE & APEX 20KC Devices

CF51005-2.1

Introduction

APEX™ 20KE and APEX 20KC devices can be configured using one of four configuration schemes. All configuration schemes use either a microprocessor or configuration device.

This section covers how to configure APEX 20KE and APEX 20KC Devices, which use a 1.8-V voltage supply for V_{CCINT}. APEX 20K (non-E and non-C) devices use a 2.5-V voltage supply for the core. If your target FPGA is an APEX 20K device which uses a 2.5-V V_{CCINT}, see *Configuring Mercury, APEX 20K (2.5 V), ACEX 1K and FLEX 10K Devices* in the Configuration Handbook.

APEX 20KE and APEX 20KC devices can be configured using the passive serial (PS), passive parallel synchronous (PPS), passive parallel asynchronous (PPA), and Joint Test Action Group (JTAG) configuration schemes. The configuration scheme used is selected by driving the APEX 20KE or APEX 20KC device `MSEL1` and `MSEL0` pins either high or low as shown in Table 7–1. If your application only requires a single configuration mode, the `MSEL` pins can be connected to V_{CC} (V_{CCIO} of the I/O bank where the `MSEL` pin resides) or to ground. If your application requires more than one configuration mode, you can switch the `MSEL` pins after the FPGA is configured successfully. Toggling these pins during user-mode does not affect the device operation; however, the `MSEL` pins must be valid before a reconfiguration is initiated.

Table 7–1. APEX 20KE & APEX 20KC Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	PS
1	0	PPS
1	1	PPA
(1)	*(1)*	JTAG Based *(2)*

Notes to Table 7–1:

(1) Do not leave the `MSEL` pins floating; connect them to a low- or high-logic level. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used, you should connect the `MSEL` pins to ground.

(2) JTAG-based configuration takes precedence over other configuration schemes, which means `MSEL` pin settings are ignored.

Table 7–2 shows the approximate configuration file sizes for APEX 20KE and APEX 20KE devices.

Table 7–2. APEX 20KE & APEX 20KC Raw Binary File (.rbf) Sizes

Device	Data Size (Bits)	Data Size (Bytes)
EP20K30E	354,832	44,354
EP20K60E	648,016	81,002
EP20K100E	1,008,016	126,002
EP20K160E	1,524,016	190,502
EP20K200E EP20K200C	1,968,016	246,002
EP20K300E	2,741,616	342,702
EP20K400E EP20K400C	3,909,776	488,722
EP20K600E EP20K600C	5,673,936	709,242
EP20K1000E EP20K1000C	8,960,016	1,120,002
EP20K1500E	12,042,256	1,505,282

Use the data in Table 7–2 only to estimate the file size before design compilation. Different configuration file formats, such as a Hexidecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II or MAX+PLUS® II software, all designs targeted for the same device will have the same configuration file size.

The following sections describe in detail how to configure APEX 20KE and APEX 20KC devices using the supported configuration schemes. The Device Configuration Pins section describes the device configuration pins available. The last section applies only to APEX 20KE devices and provides guidelines that you must follow to ensure successful configuration upon power-up and recovery from brown-out conditions. In this chapter, the generic term "device(s)" or "FPGA(s)" will include all APEX 20KE and APEX 20KC devices.

For more information on setting device configuration options or creating configuration files, *see Section II, Software Settings, in Volume 2*.

Passive Serial Configuration

You can perform APEX 20KE and APEX 20KC PS configuration using an Altera configuration device, an intelligent host (e.g., a microprocessor or Altera® MAX® device), or a download cable.

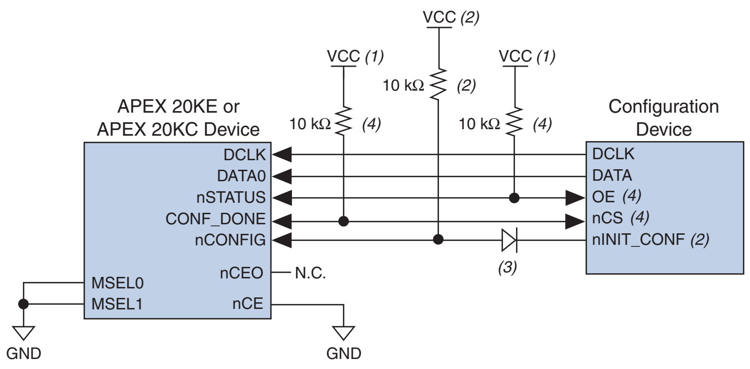
PS Configuration Using a Configuration Device

You can use an Altera configuration device, such as an enhanced configuration device, EPC2, or EPC1 device, to configure APEX 20KE and APEX 20KC devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 7–1 shows the configuration interface connections between the APEX 20KE or APEX 20KC device and a configuration device for single device configuration.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on the enhanced configuration device and flash interface pins (e.g., `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`), see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* in the Configuration Handbook.

Figure 7–1. Single Device PS Configuration Using a Configuration Device



Notes to Figure 7–1:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} through a 10-kΩ resistor. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}. For APEX 20KC devices, `nCONFIG` should be connected to the same supply voltage as the configuration device.

(3) The nINIT_CONF pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up resistor to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(4) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the Operating Conditions table of the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the Configuration Handbook.

When using enhanced configuration devices or EPC2 devices, nCONFIG of the FPGA can be connected to nINIT_CONF, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. The nINIT_CONF pin does not need to be connected if its functionality is not used. An internal pull-up resistor on the nINIT_CONF pin is always active in the enhanced configuration devices and the EPC2 devices, which means an external pull-up resistor is not required if nCONFIG is tied to nINIT_CONF. Since a 10-kΩ pull-up resistor to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's nINIT_CONF pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the nINIT_CONF pin an open-drain pin; the pin will only be able to drive low or tri-state. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CCINT} through a 10-kΩ pull-up resistor and the isolating diode is not needed.

Upon power-up, the APEX 20KE or APEX 20KC device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for EPC2, EPC1, and EPC1441 devices is 200 ms (maximum), and for enhanced configuration devices, the POR time can be set to either 100 ms or 2 ms, depending on its PORSEL pin setting. If the PORSEL pin is connected to GND, the POR delay is 100 ms. During this time, the configuration device drives its OE pin low. This low signal delays configuration because the OE pin is connected to the target device's nSTATUS pin. When both devices complete POR, they release their open-drain OE or nSTATUS pin, which is then pulled high by a pull-up resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX 20KE and APEX 20KC devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX 20K Programmable Logic Device Family Data Sheet* or *APEX 20KC Programmable Logic Device Family Data Sheet*.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on nCONFIG and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. The beginning of configuration can be delayed by holding the nCONFIG or nSTATUS pin low.

☞ V_{CCINT} and V_{CCIO} of the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the `nSTATUS` pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up on the `OE` pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, an external 10-kΩ pull-up resistor on the `nCS`/`CONF_DONE` line is not required. Once `nSTATUS` is released, the FPGA is ready to receive configuration data and the configuration stage begins.

When `nSTATUS` is pulled high, `OE` of the configuration device also goes high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The APEX 20KE or APEX 20KC device receives configuration data on its `DATA0` pin and the clock is received on the `DCLK` pin. Data is latched into the FPGA on the rising edge of `DCLK`.

After the FPGA has received all configuration data successfully, it releases the open-drain `CONF_DONE` pin, which is pulled high by a pull-up resistor. Since `CONF_DONE` is tied to the configuration device's `nCS` pin, the configuration device is disabled when `CONF_DONE` goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the `nCS` pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device are used, an external 10-kΩ pull-up resistor on the `nCS`/`CONF_DONE` line is not required. A low-to-high transition on `CONF_DONE` indicates configuration is complete and initialization of the device can begin.

In APEX 20KE and APEX 20KC devices, the initialization clock source is either the APEX 20KE or APEX 20KC internal oscillator (typically 10 MHz) or the optional `CLKUSR` pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX 20KE or APEX 20KC device will supply itself with enough clock cycles for proper initialization. You also have the flexibility to synchronize initialization of multiple devices by using the `CLKUSR` option. You can turn on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on `CLKUSR` will not affect the configuration process. After all configuration data is accepted and CONF_DONE goes high, APEX 20KE and APEX 20KC devices require 40 clock cycles to properly initialize.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. The enhanced configuration device and EPC2 device drives DCLK low and DATA high (EPC1 devices tri-state DATA) at the end of configuration.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the *Auto-Restart Configuration After Error* option available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box is turned on, the FPGA automatically initiates reconfiguration if an error occurs. The APEX 20KE or APEX 20KC device will release its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the FPGA has not configured successfully. Enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit was sent for CONF_DONE to reach a high state. EPC1 and EPC2 devices wait for 16 DCLK cycles. In this case, the configuration device pulls its OE pin low, which in turn drives the target device's nSTATUS pin low. If the *Auto-Restart Configuration After Error* option is set in the software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns high, the configuration device tries to reconfigure the FPGA.

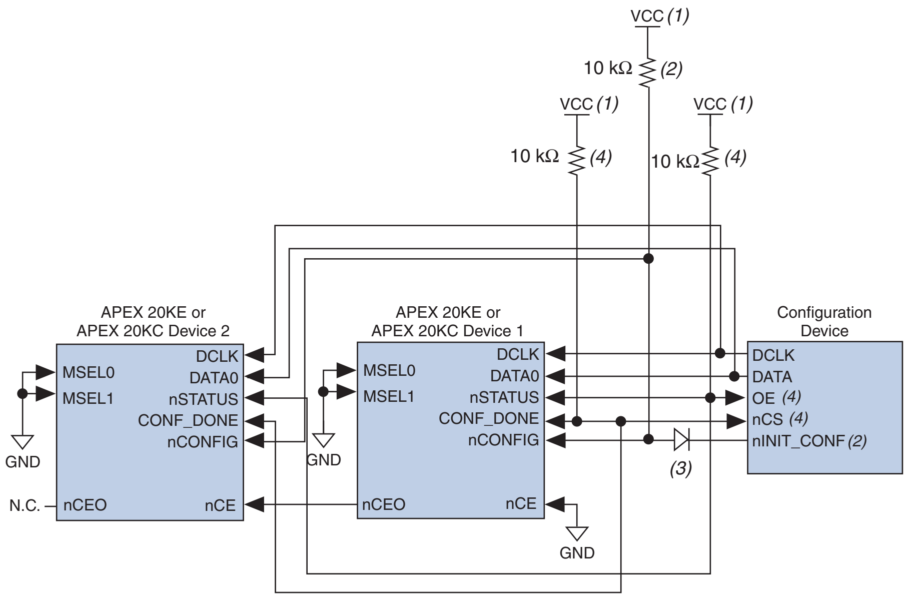
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully; therefore, your system should not pull CONF_DONE low to delay initialization. Instead, use the CLKUSR option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this will activate the configuration device since it will see its nCS pin drive low. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 7–2 shows how to configure multiple devices with a configuration device. This circuit is similar to the configuration device circuit for a single device, except the APEX 20KE or APEX 20KC devices are cascaded for multi-device configuration.

Figure 7–2. Multi-Device PS Configuration Using a Configuration Device



Notes to Figure 7–2:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} through a 10-kΩ resistor. For APEX 20KE devices, nCONFIG should be pulled up to V_{CCINT}. For APEX 20KC devices, nCONFIG should be connected to the same supply voltage as the configuration device.

(3) The nINIT_CONF pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's nINIT_CONF pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the nINIT_CONF pin an open-drain pin; the pin will only be able to drive low or tri-state. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(4) The enhanced configuration devices' and EPC2 devices' OE and nCS pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

When performing multi-device configuration, you must generate the configuration device's Programmer Object File (**.pof**) from each project's SRAM Object File (**.sof**). You can combine multiple SOFs using the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, *see Section II, Software Settings, in Volume 2*.

In multi-device PS configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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.

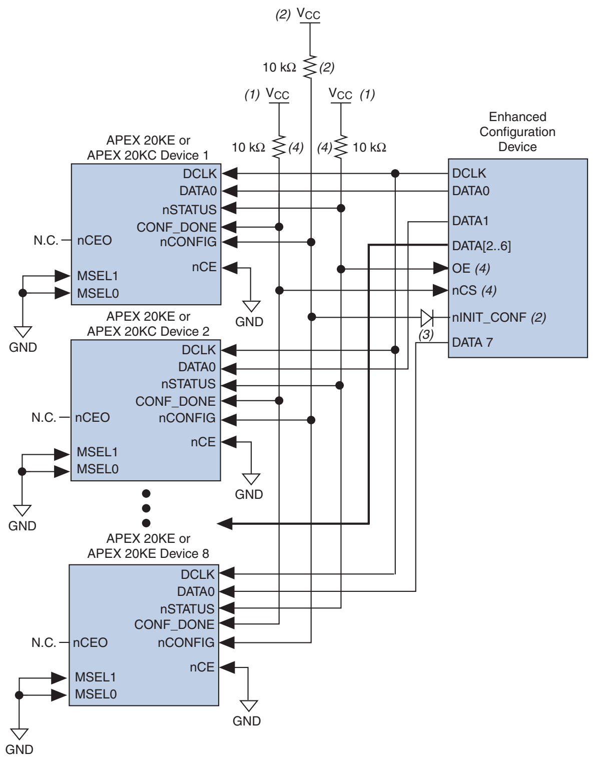
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the *Auto-Restart Configuration After Error* option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low for at least 8 µs to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

The enhanced configuration devices also support parallel configuration of up to eight devices. The n-bit (n = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure FPGAs or a chain of FPGAs. In addition, these devices do not have to be the same device family or density; they can be any combination of Altera FPGAs. An individual enhanced configuration device `DATA` line is available for each targeted FPGA. Each `DATA` line can also feed a daisy chain of FPGAs. Figure 7–3 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 7–3. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device



Notes to Figure 7–3:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} through a 10-kΩ resistor. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}. For APEX 20KC devices, `nCONFIG` should be connected to the same supply voltage as the configuration device.

(3) The `nINIT_CONF` pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's `nCONFIG` pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(4) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

The Quartus II software only allows the selection of n-bit PS configuration modes, where n must be 1, 2, 4, or 8. 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 7–3 to select the appropriate configuration mode for the fastest configuration times.

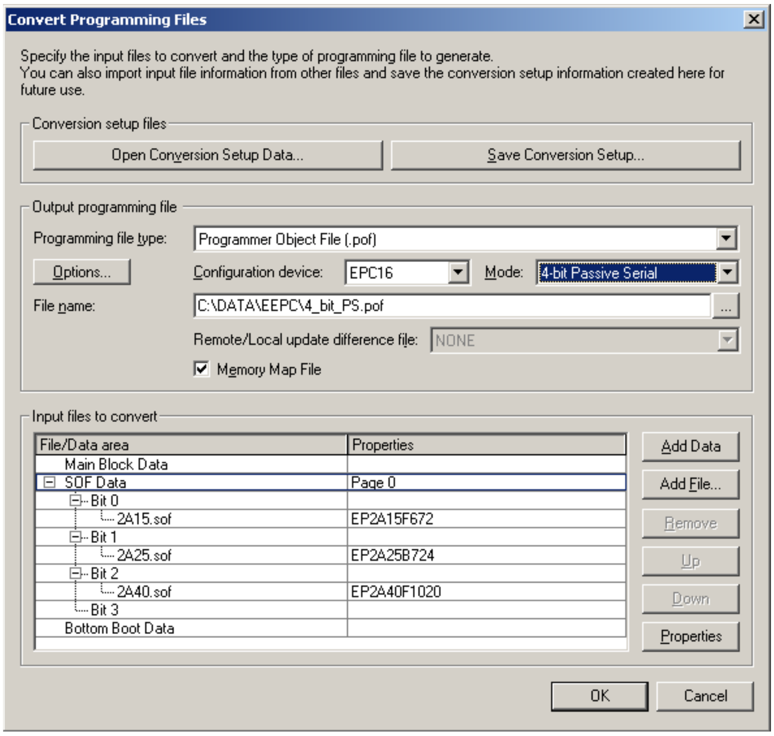
Table 7–3. Recommended Configuration Using n-Bit PS Modes

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 7–3:
(1) Assume that each `DATA` line is only configuring one device, not a daisy chain of devices.

For example, if you configure three FPGAs, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding SOF data is transmitted from the configuration device to the FPGA. 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 7–4 shows the Quartus II **Convert Programming Files** window (Tools menu) setup for this scheme.

Figure 7–4. Software Settings for Configuring Devices Using n-Bit PS Modes



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: 4-bit Passive Serial

File name: C:\DATA\EEPC\4_bit_PS.pof

Remote/Local update difference file: NONE

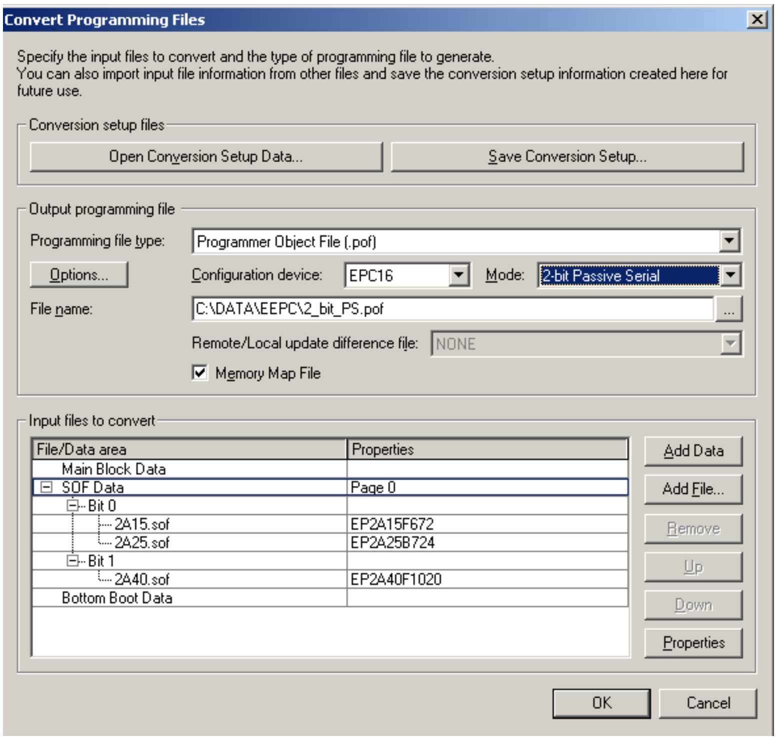
☑ Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
2A15.sof	EP2A15F672
Bit 1	
2A25.sof	EP2A25B724
Bit 2	
2A40.sof	EP2A40F1020
Bit 3	
Bottom Boot Data	

Alternatively, you can daisy chain two FPGAs 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 FPGAs with `DATA Bit0` (EP20K400E and EP20K600E devices) and the third device (the EP20K1000E device) with `DATA Bit1`. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but can increase the total system configuration time. See Figure 7–5.

Figure 7–5. Software Settings for Daisy Chaining Two FPGAs on One DATA Line



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: 2-bit Passive Serial

File name: C:\DATA\EEPC\2_bit_PS.pof

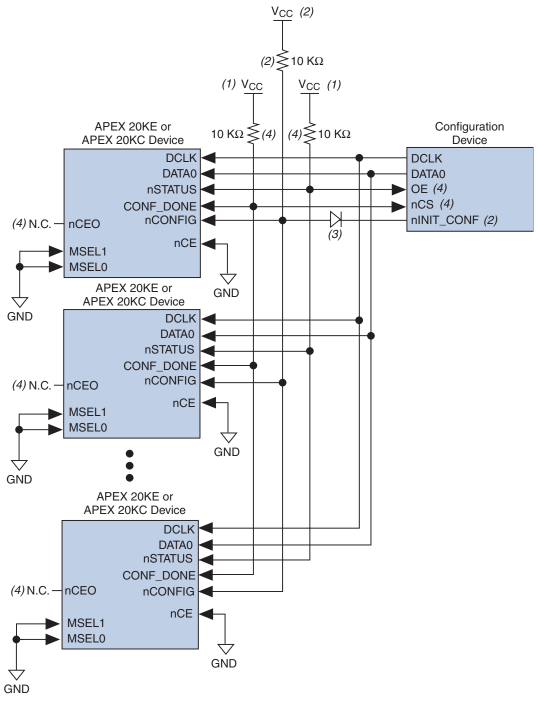
Remote/Local update difference file: NONE

☑ Memory Map File

File/Data area	Properties
Main Block Data	
⊟ SOF Data	Page 0
⊟ Bit 0	
2A15.sof	EP2A15F672
2A25.sof	EP2A25B724
⊟ Bit 1	
2A40.sof	EP2A40F1020
Bottom Boot Data	

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 7–6 shows multi-device PS configuration when the APEX 20KE and APEX 20KC devices are receiving the same configuration data.

Figure 7–6. Multiple-Device PS Configuration Using an Enhanced Configuration Device When FPGAs Receive the Same Data



Notes to Figure 7–6:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} through a 10-kΩ resistor. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}. For APEX 20KC devices, `nCONFIG` should be connected to the same supply voltage as the configuration device.

(3) The nINIT_CONF pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(4) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

(5) The `nCEO` pins of all devices are left unconnected when configuring the same configuration data into multiple devices.

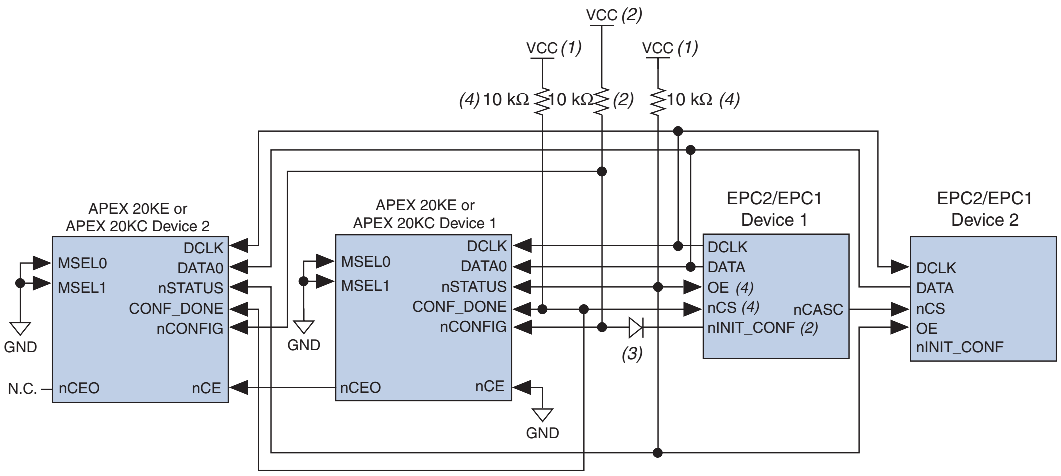
You can cascade several EPC2 or EPC1 devices to configure multiple APEX 20KE or APEX 20KC devices. The first configuration device in the chain is the master configuration device, while the subsequent devices are the slave devices. The master configuration device sends DCLK to the APEX 20KE and APEX 20KC devices and to the slave configuration devices. The first EPC device's nCS pin is connected to the CONF_DONE pins of the FPGAs, while its nCASC pin is connected to nCS of the next configuration device in the chain. The last device's nCS input comes from the previous device, while its nCASC pin is left floating. When all data from the first configuration device is sent, it drives nCASC low, which in turn drives nCS on the next configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

Enhanced configuration devices EPC4, EPC8, and EPC16 cannot be cascaded.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect CONF_DONE going high at the end of configuration, it resets the entire chain by pulling its OE pin low. This low signal drives the OE pin low on the slave configuration device(s) and drives nSTATUS low on all FPGAs, causing them to enter a reset state. This behavior is similar to the FPGA detecting an error in the configuration data.

Figure 7–7 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 7–7. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 7–7:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} through a 10-kΩ resistor. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}. For APEX 20KC devices, `nCONFIG` should be connected to the same supply voltage as the configuration device.

(3) The `nINIT_CONF` pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up resistor to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's `nCONFIG` pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(4) The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

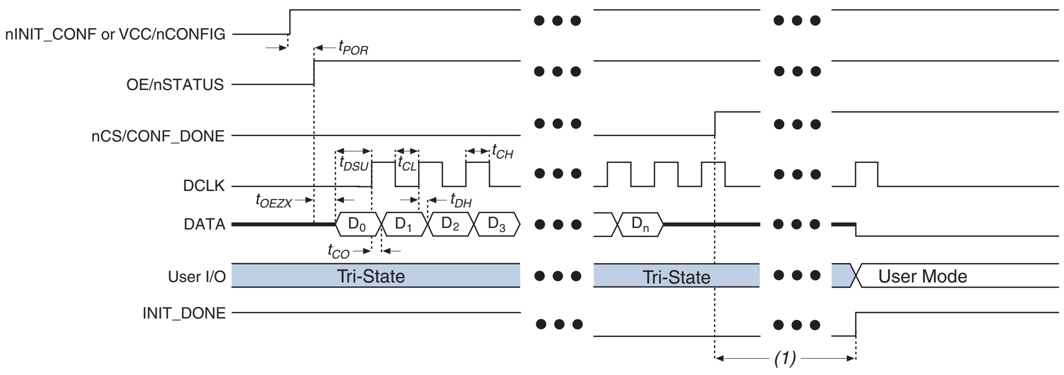
When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. An internal pull-up resistor on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices, which means an external pull-up resistor is not required if `nCONFIG` is tied to `nINIT_CONF`. Since a 10-kΩ pull-up resistor to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's `nCONFIG` pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CCINT} through a 10 kΩ resistor and the isolating diode is not needed. If multiple EPC2 devices are used to configure an APEX 20KE or APEX 20KC device(s), only the first EPC2 has its `nINIT_CONF` pin tied to the device's `nCONFIG` pin.

You can use a single configuration chain to configure APEX 20KE and APEX 20KC devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 7–8 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 7–8. APEX 20KE & APEX 20KC PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 7–8:

(1) APEX 20KE and APEX 20KC devices enter user-mode 40 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the APEX 20KE or APEX 20KC internal oscillator or the `CLKUSR` pin.

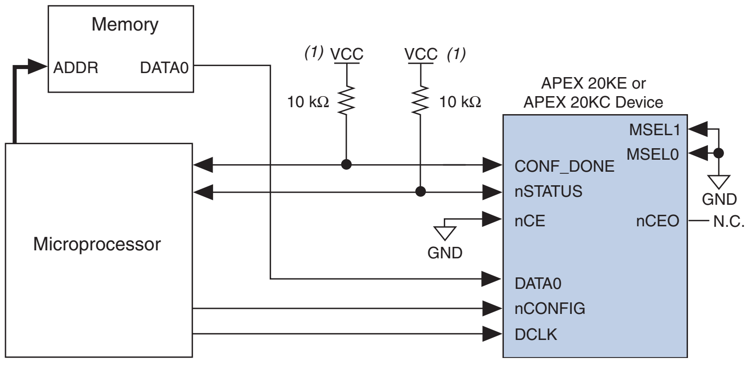
For timing information, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the Configuration Handbook.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the Configuration Handbook.

PS Configuration Using a Microprocessor

In the PS configuration scheme, an intelligent host (e.g., a microprocessor or CPLD) can transfer configuration data from a storage device (e.g., flash memory) to the target APEX 20KE and APEX 20KC devices. Configuration data can be stored in RBF, HEX, or TTF format. Figure 7–9 shows the configuration interface connections between the APEX 20KE or APEX 20KC device and a microprocessor for single device configuration.

Figure 7–9. Single Device PS Configuration Using a Microprocessor



Note to Figure 7–9:
(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device.

Upon power-up, the APEX 20KE or APEX 20KC device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX 20KE and APEX 20KC devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX 20K Programmable Logic Device Family Data Sheet* or *APEX 20KC Programmable Logic Device Family Data Sheet*.

The configuration cycle consists of three stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` is released, the FPGA is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should place the configuration data one bit at a time on the DATA0 pin. The least significant bit (LSB) of each data byte must be sent first.

The APEX 20KE or APEX 20KC device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK. Data is continuously clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In APEX 20KE and APEX 20KC devices, the initialization clock source is either the APEX 20KE or APEX 20KC internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX 20KE or APEX 20KC device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, APEX 20KE and APEX 20KC devices require 40 clock cycles to initialize properly.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design.

To ensure DCLK and DATA are not left floating at the end of configuration, the microprocessor must drive them either high or low, whichever is convenient on your board.

Handshaking signals are not used in PS configuration mode. Therefore, the configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the APEX 20KE or APEX 20KC device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

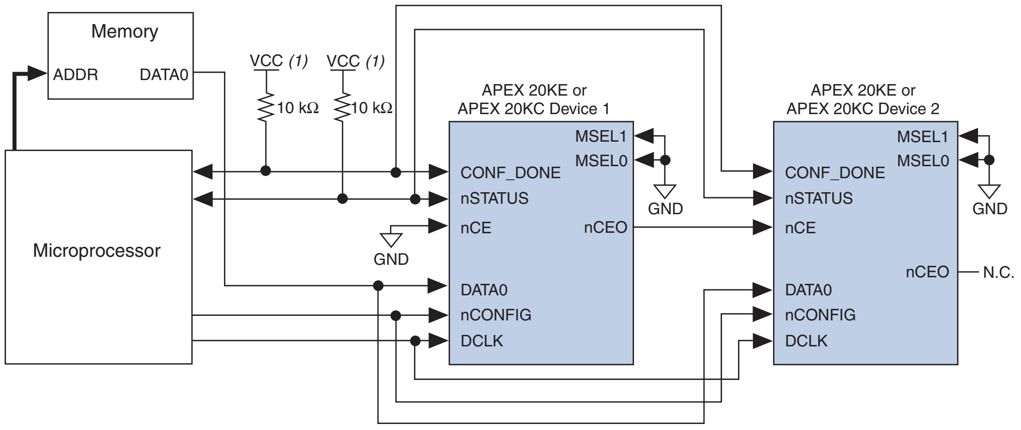
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, you can initiate a reconfiguration by transitioning the nCONFIG pin low-to-high. The nCONFIG pin must be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 7–10 shows how to configure multiple devices using a microprocessor. This circuit is similar to the PS configuration circuit for a single device, except the APEX 20KE or APEX 20KC devices are cascaded for multi-device configuration.

Figure 7–10. Multi-Device PS Configuration Using a Microprocessor



Note to Figure 7–10:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

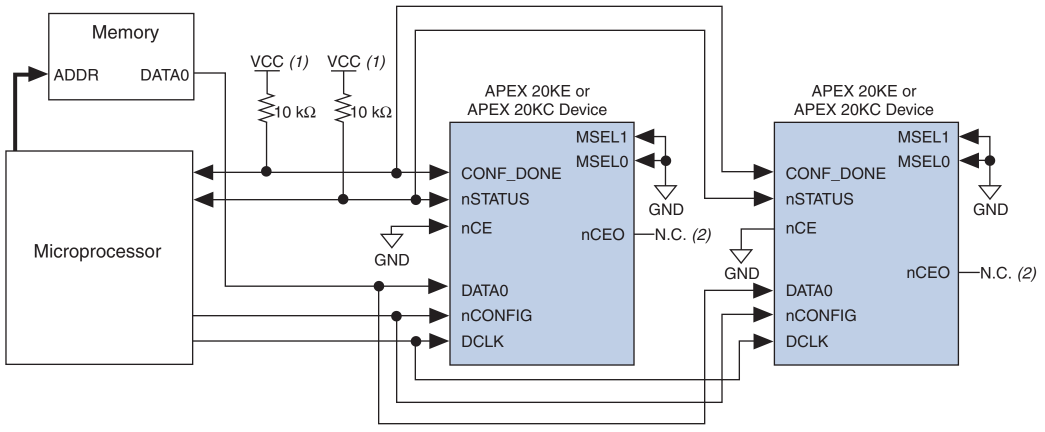
In multi-device PS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Because all device `CONF_DONE` pins are tied together, all devices initialize and enter user mode at the same time.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on `nSTATUS`, it resets the chain by pulling its `nSTATUS` pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you can have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 7–11 shows multi-device PS configuration when both APEX 20KE and APEX 20KC devices are receiving the same configuration data.

Figure 7–11. Multiple-Device PS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 7–11:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

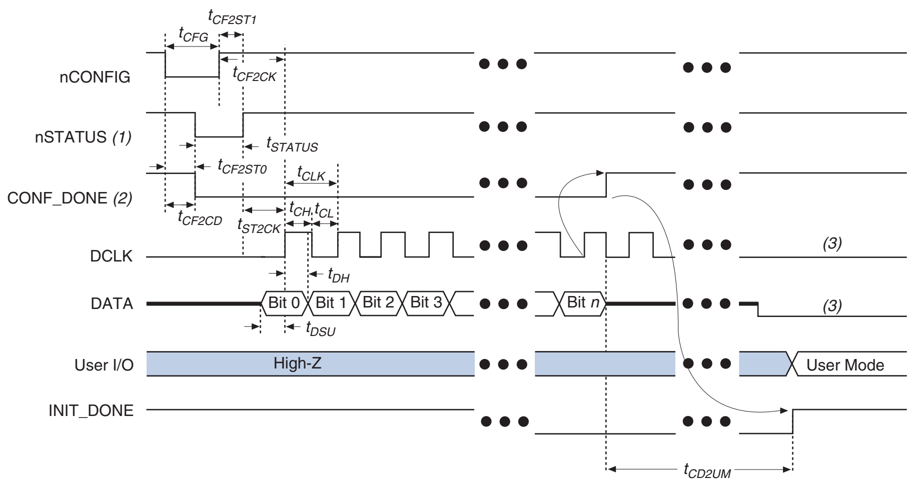
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX 20KE and APEX 20KC devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 7–12 shows the timing waveform for the PS configuration for APEX 20KE and APEX 20KC devices using a microprocessor.

Figure 7–12. APEX 20KE & APEX 20KC PS Configuration Using a Microprocessor Timing Waveform



Notes to Figure 7–12:

(1) Upon power-up, the APEX 20KE or APEX 20KC device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.
(3) `DATA0` and `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Table 7–4 defines the timing parameters for APEX 20KE and APEX 20KC devices for PS configuration.

Table 7–4. PS Timing Parameters for APEX 20KE & APEX 20KC Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	7.5		ns
t_{CL}	`DCLK` low time	7.5		ns
t_{CLK}	`DCLK` period	15		ns
f_{MAX}	`DCLK` maximum frequency		66	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 7–4:

(1) This value is applicable if users do not delay configuration by extending the `nSTATUS` low pulse width.

(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting the device. If the clock source is `CLKUSR`, multiply the clock period by 40 for APEX 20KE and APEX 20KC devices to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the Configuration Handbook.

Configuring Using the MicroBlaster Driver

The MicroBlaster^{TM} software driver allows you to configure Altera's FPGAs through the ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a RBF programming input file and is targeted for embedded passive serial configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, go to the Altera web site (http://www.altera.com).

PS Configuration Using a Download Cable

In this section, the generic term "download cable" includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable.

In PS configuration with a download cable, an intelligent host (e.g., a PC) transfers data from a storage device to the FPGA via the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Upon power-up, the APEX 20KE or APEX 20KC device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX 20KE and APEX 20KC devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX 20K Programmable Logic Device Family Data Sheet* or *APEX 20KC Programmable Logic Device Family Data Sheet*.

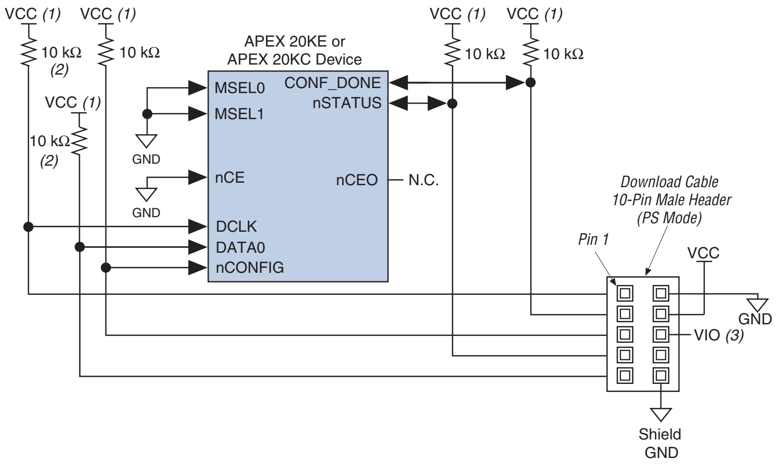
The configuration cycle consists of 3 stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. The programming hardware or download cable then places the configuration data one bit at a time on the device's `DATA0` pin. The configuration data is clocked into the target device until `CONF_DONE` goes high.

When using a download cable, setting the *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs. Additionally, the *Enable user-supplied start-up clock (CLKUSR)* option has no affect on the device initialization since this option is disabled in the SOF when programming the FPGA using the Quartus II programmer and download cable. Therefore, if you turn on the CLKUSR option, you do not need to provide a clock on CLKUSR when you are configuring the FPGA with the Quartus II programmer and a download cable. Figure 7–13 shows PS configuration for APEX 20KE and APEX 20KC devices using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cable.

Figure 7–13. PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 7–13:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (VIO pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

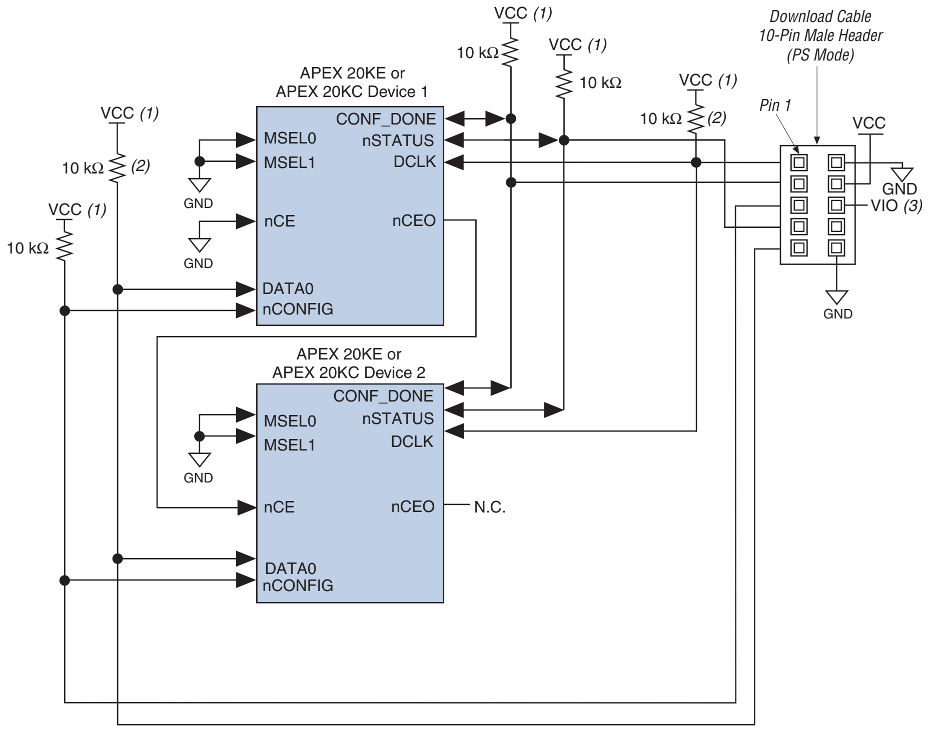
(3) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

You can use a download cable to configure multiple APEX 20KE and APEX 20KC devices by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. The first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to the `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. All other configuration pins, `nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE` are connected to every device in the chain. Because all `CONF_DONE` pins are tied together, all devices in the chain initialize and enter user mode at the same time.

In addition, because the `nSTATUS` pins are tied together, the entire chain halts configuration if any device detects an error. The *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs.

Figure 7–14 shows how to configure multiple APEX 20KE and APEX 20KC devices with a download cable.

Figure 7–14. Multi-Device PS Configuration using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 7–14:

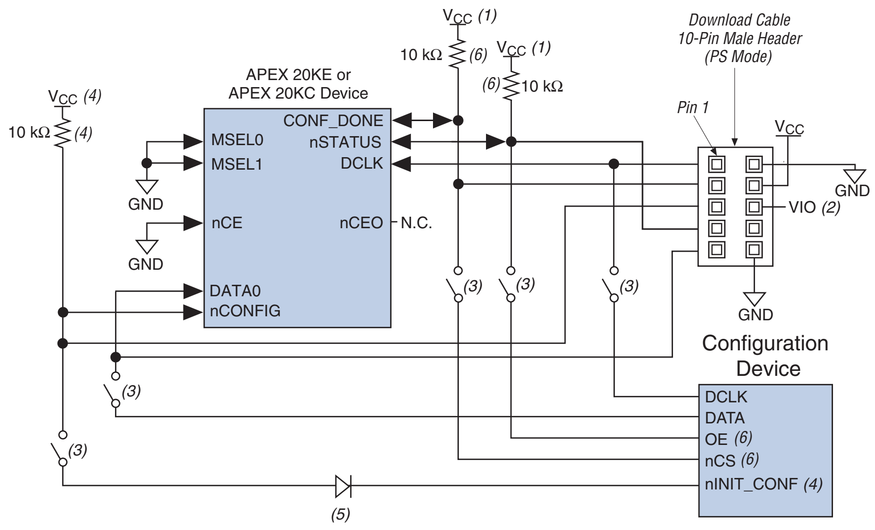
(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

If you are using a download cable to configure device(s) on a board that also has configuration devices, you should electrically isolate the configuration device from the target device(s) and cable. One way to isolate the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer chip should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) between the cable and the configuration device. The last option is to remove the configuration device from the board when configuring the FPGA with the cable. Figure 7–15 shows a combination of a configuration device and a download cable to configure an FPGA.

Figure 7–15. PS Configuration with a Download Cable & Configuration Device Circuit



Notes to Figure 7–15:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to an APEX 20KE or APEX 20KC device. Instead, 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.

(4) The nINIT_CONF pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), nCONFIG must be pulled to V_{CC} through a 10-kΩ pull-up resistor. For APEX 20KE devices, nCONFIG should be pulled up to V_{CCINT}. For APEX 20KC devices, nCONFIG should be connected to the same supply voltage as the configuration device.

(5) The nINIT_CONF pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's nINIT_CONF pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the nINIT_CONF pin an open-drain pin; the pin will only be able to drive low or tri-state. If nINIT_CONF is not used or not available (e.g., on EPC1 devices), this diode is not needed. The diode is also not needed when configuring APEX 20KC devices.

(6) The enhanced configuration devices' and EPC2 devices' OE and nCS pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, you should use external 10-kΩ pull-up resistors. If internal pull-up resistors on the enhanced configuration device 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-up resistors on configuration device* option when generating programming files.

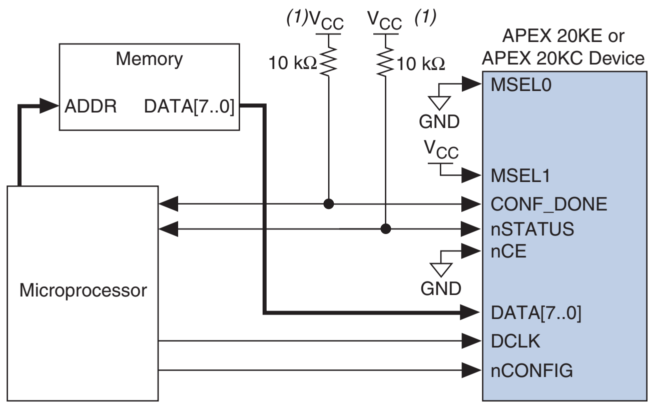
For more information on how to use the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cables, refer to the followig data sheets.

- 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

Passive Parallel Synchronous Configuration

Passive Parallel Synchronous (PPS) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target APEX 20KE or APEX 20KC device. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the serializing clock to the FPGA. The target device latches the byte-wide data on the `DATA[7..0]` pins on the rising edge of `DCLK` and then uses the next eight falling edges on `DCLK` to serialize the data internally. On the ninth rising `DCLK` edge, the next byte of configuration data is latched into the target device. Figure 7–16 shows the configuration interface connections between the FPGA and a microprocessor for single device configuration.

Figure 7–16. Single Device PPS Configuration Using a Microprocessor



Note to Figure 7–16:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

Upon power-up, the APEX 20KE or APEX 20KC device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX 20KE and APEX 20KC devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of 3 stages: reset, configuration and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should place the configuration data one byte at a time on the `DATA[7..0]` pins. New configuration data should be sent to the FPGA every eight `DCLK` cycles.

The APEX 20KE or APEX 20KC device receives configuration data on its `DATA[7..0]` pins and the clock is received on the `DCLK` pin. On the first rising `DCLK` edge, a byte of configuration data is latched into the target device; the subsequent eight falling `DCLK` edges serialize the configuration data in the device. On the ninth rising clock edge, the next byte of configuration data is latched and serialized into the target device.

Data is clocked into the target device until `CONF_DONE` goes high. After the FPGA has received all configuration data successfully, it releases the open-drain `CONF_DONE` pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on `CONF_DONE` indicates configuration is complete and initialization of the device can begin.

In APEX 20KE and APEX 20KC devices, the initialization process is synchronous and can be clocked by its internal oscillator (typically 10 MHz) or by the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX 20KE or APEX 20KC device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, APEX 20KE and APEX 20KC devices require 40 clock cycles to initialize properly.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 10-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DCLK and DATA0 are not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. The DATA[7..1] pins are available as user I/O pins after configuration. When the PPS scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

The configuration clock (DCLK) speed must be below the specified frequency, as listed in Table 7–5, to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time. An optional status pin (RDYnBSY) on the FPGA indicates when it is busy serializing configuration data and when it is ready to accept the next data byte. The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration on Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

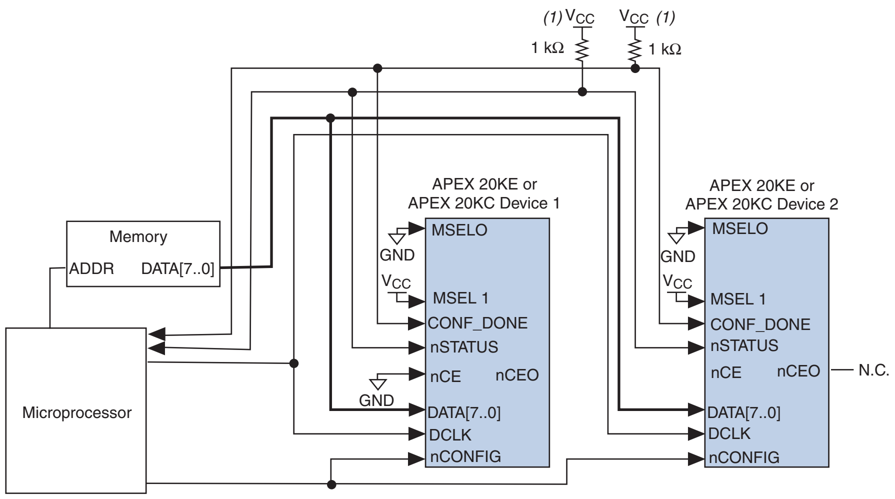
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 8 µs for APEX 20KE and APEX 20KC devices. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 7–17 shows how to configure multiple APEX 20KE and APEX 20KC devices using a microprocessor. This circuit is similar to the PPS configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 7–17. Multi-Device PPS Configuration Using a Microprocessor



Note to Figure 7–17:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Altera recommends keeping the configuration data valid on the `DATA[7..0]` bus for the 8 serializing clock cycles. The configuration data should be held valid on the `DATA` bus for the complete byte period because the `nCEO` of the first (and preceding) device can go low during the serializing `DCLK` cycles. Once the `nCEO` of the first (and preceding) device goes low, the second (and next) device becomes active and will begin trying to accept configuration data. If the configuration data is not valid on the first `DCLK` edge after `nCEO` goes low, then the second device will see incorrect configuration data and will never begin accepting configuration data. This situation will only arise if you are sharing the `DATA[7..0]` bus with other system data such that the configuration data is only valid for a portion of the byte period.

If your system requires to bus-share the DATA[7..0] line, you can work-around this by ensuring that the second (or next) device sees correct configuration data on the first rising edge of DCLK after the nCEO signal goes low. This can be achieved by delaying the nCEO signal by using external registers or by presenting the next byte of configuration data after the nCEO transition.

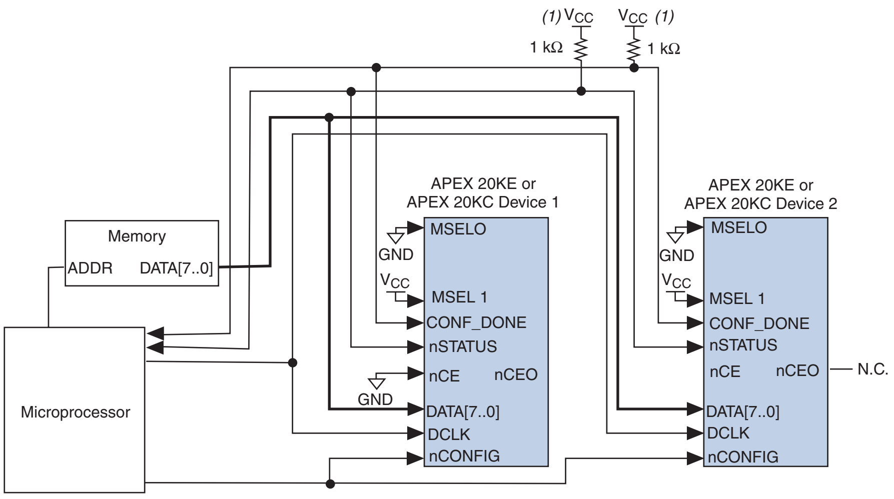
All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration on Frame Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 7–18 shows multi-device PPS configuration when both devices are receiving the same configuration data.

Figure 7–18. Multiple-Device PPS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 7–18:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

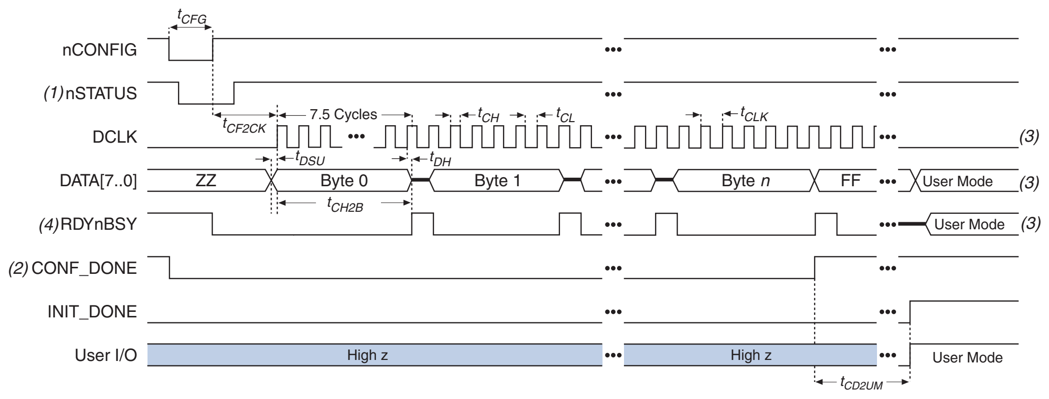
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX 20KE or APEX 20KC devices with other Altera devices that support PPS configuration, such as Mercury™, ACEX® 1K, or FLEX® 10K devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 7–19 shows the timing waveform for the PPS configuration scheme using a microprocessor.

Figure 7–19. APEX 20KE & APEX 20KC PPS Configuration Timing Waveform



Notes to Figure 7–19:

(1) Upon power-up, the APEX 20KE and APEX 20KC devices holds nSTATUS low for approximately 5 µs after VCC reaches its minimum requirement.

(2) Upon power-up, before and during configuration, CONF_DONE is low.

(3) DATA0 and DCLK should not be left floating after configuration. It should be driven high or low, whichever is more convenient. DATA[7..1] and RDYnBSY are available as user I/Os after configuration and the state of theses pins depends on the design programmed into the device.

(4) The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.

Table 7–5 defines the timing parameters for APEX 20KE and APEX 20KC devices for PPS configuration.

Table 7–5. PPS Timing Parameters for APEX 20KE & APEX 20KC Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	8		µs
t_{STATUS}	nSTATUS low pulse width	10	40 (1)	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		1 (1)	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	40		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH2B}	First rising DCLK to first rising RDYnBSY (2)	0.75 (3)		µs

Table 7–5. PPS Timing Parameters for APEX 20KE & APEX 20KC Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CH}	DCLK high time	15		ns
t_{CL}	DCLK low time	15		ns
t_{CLK}	DCLK period	30		ns
f_{MAX}	DCLK frequency		33.3	MHz
t_{CD2UM}	CONF_DONE high to user mode *(4)*	2	8	µs

Notes to Table 7–5:

(1) This value is obtainable if users do not delay configuration by extending the nCONFIG or nSTATUS low pulse width.

(2) The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.

(3) This parameter depends on the DCLK frequency. The RDYnBSY signal goes high 7.5 clock cycles after the rising edge of DCLK. This value was calculated with a DCLK frequency of 10 MHz.

(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 40 to obtain this value.

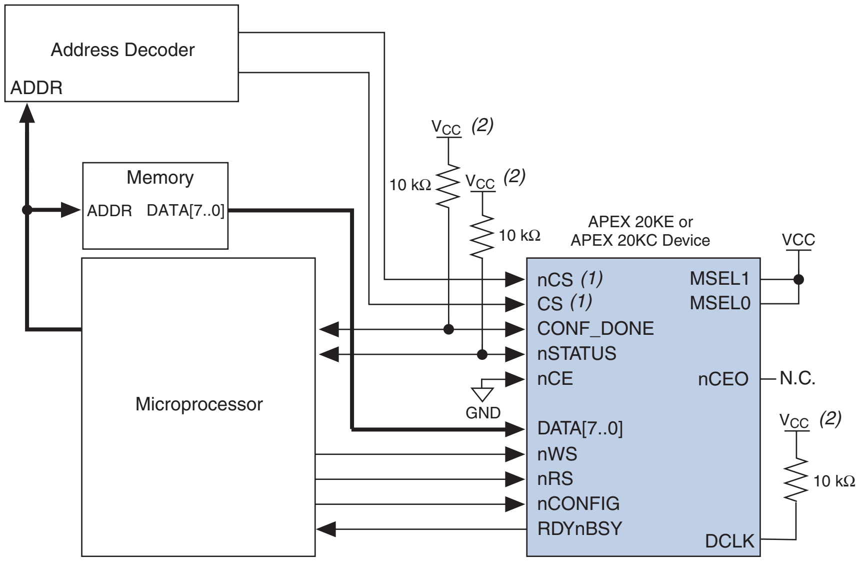
Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume, 2* of the Configuration Handbook.

Passive Parallel Asynchronous Configuration

Passive Parallel Asynchronous (PPA) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target APEX 20KE or APEX 20KC device. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the accompanying strobe signals to the FPGA. When using PPA, you should pull the DCLK pin high through a 10-kΩ pull-up resistor to prevent unused configuration input pins from floating.

Figure 7–20 shows the configuration interface connections between the FPGA and a microprocessor for single device PPA configuration. The microprocessor or an optional address decoder can control the device's chip select pins, nCS and CS. The address decoder allows the microprocessor to select the APEX 20KE or APEX 20KC device by accessing a particular address, which simplifies the configuration process. The nCS and CS pins must be held active during configuration and initialization.

Figure 7–20. Single Device PPA Configuration Using a Microprocessor Note (1)



Notes to Figure 7–20:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

During PPA configuration, it is only required to use either the nCS or CS pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, nCS can be tied to ground while CS is toggled to control configuration. The device's nCS or CS pins can be toggled during PPA configuration if the design meets the specifications set for t_{CSSU}, t_{WSP}, and t_{CSH} listed in Table 7–6.

Upon power-up, the APEX 20KE or APEX 20KC device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds nSTATUS low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. APEX 20KE and APEX 20KC devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the *APEX 20K Programmable Logic Device Family Data Sheet* or *APEX 20KC Programmable Logic Device Family Data Sheet*.

The configuration cycle consists of 3 stages: reset, configuration and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the nCONFIG pin.

VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 10-kΩ pull-up resistor. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should then assert the target device's nCS pin low and/or CS pin high. Next, the microprocessor places an 8-bit configuration word (one byte) on the target device's DATA[7..0] pins and pulses the nWS pin low.

On the rising edge of nWS, the target device latches in a byte of configuration data and drives its RDYnBSY signal low, which indicates it is processing the byte of configuration data. The microprocessor can then perform other system functions while the APEX 20KE or APEX 20KC device is processing the byte of configuration data.

During the time RDYnBSY is low, the APEX 20KE or APEX 20KC device internally processes the configuration data using its internal oscillator (typically 10 MHz). When the device is ready for the next byte of configuration data, it will drive RDYnBSY high. If the microprocessor senses a high signal when it polls RDYnBSY, the microprocessor sends the next byte of configuration data to the FPGA.

Alternatively, the nRS signal can be strobed low, causing the RDYnBSY signal to appear on DATA7. Because RDYnBSY does not need to be monitored, this pin doesn't need to be connected to the microprocessor. Data should not be driven onto the data bus while nRS is low because it will cause contention on the DATA7 pin. If the nRS pin is not used to monitor configuration, it should be tied high.

To simplify configuration and save an I/O port, the microprocessor can wait for the total time of t_{BUSY}(max) + t_{RDY2WS} + t_{W2SB} before sending the next data byte. In this set-up, nRS should be tied high and RDYnBSY does not need to be connected to the microprocessor. The t_{BUSY}, t_{RDY2WS} and t_{W2SB} timing specifications are listed in Table 7–6.

Next, the microprocessor checks `nSTATUS` and `CONF_DONE`. If `nSTATUS` is not low and `CONF_DONE` is not high, the microprocessor sends the next data byte. However, if `nSTATUS` is not low and all the configuration data has been received, the device is ready for initialization. After the FPGA has received all configuration data successfully, it releases the open-drain `CONF_DONE` pin, which is pulled high by an external 10-kΩ pull-up resistor. A low-to-high transition on `CONF_DONE` indicates configuration is complete and initialization of the device can begin.

In APEX 20KE and APEX 20KC devices, the initialization clock source is either the APEX 20KE or APEX 20KC internal oscillator (typically 10 MHz) or the optional `CLKUSR` pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the APEX 20KE or APEX 20KC device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device.

You also have the flexibility to synchronize initialization of multiple devices by using the `CLKUSR` option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on `CLKUSR` will not affect the configuration process. After all configuration data has been accepted and `CONF_DONE` goes high, APEX 20KE and APEX 20KC devices require 40 clock cycles to initialize properly.

An optional `INIT_DONE` pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the `INIT_DONE` pin is used it will be high due to an external 10-kΩ pull-up when `nCONFIG` is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DATA0 is not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. After configuration, the nCS, CS, nRS, nWS, RDYnBSY, and DATA[7..1] pins can be used as user I/O pins. When the PPA scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

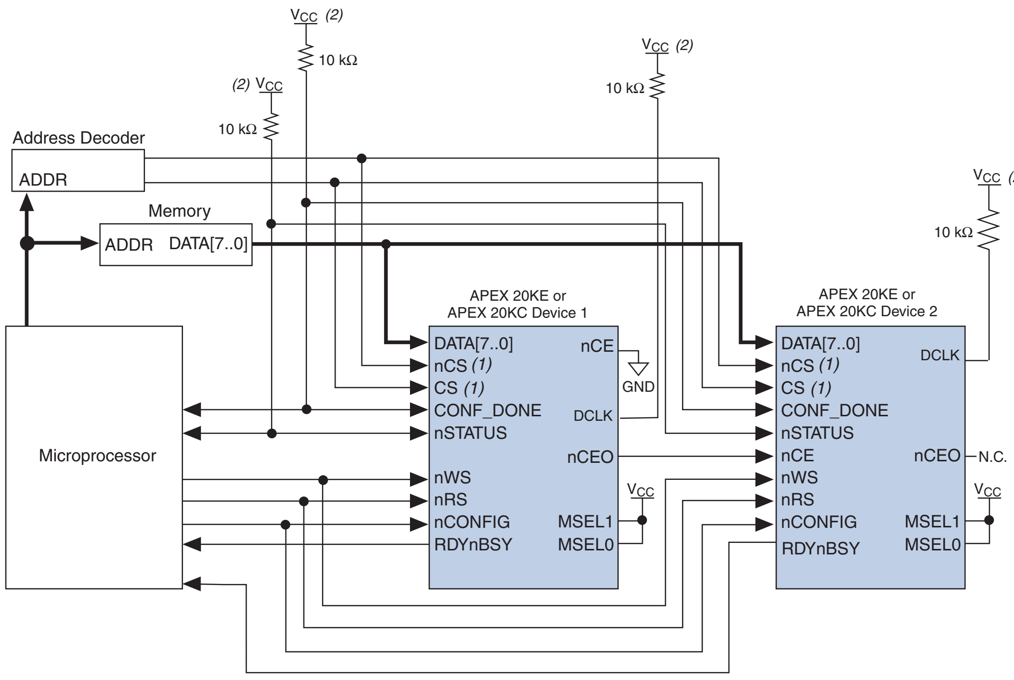
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE has not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the nCONFIG pin low-to-high. The nCONFIG pin should be low for at least 8 µs. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 7–21 shows how to configure multiple APEX 20KE and APEX 20KC devices using a microprocessor. This circuit is similar to the PPA configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 7–21. Multi-Device PPA Configuration Using a Microprocessor



Notes to Figure 7–21:

(1) If not used, the `CS` pin can be connected to V_{CC} directly. If not used, the `nCS` pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPA configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Each device's RDYnBSY pin can have a separate input to the microprocessor. Alternatively, if the microprocessor is pin limited, all the RDYnBSY pins can feed an AND gate and the output of the AND gate can feed the microprocessor. For example, if you have 2 devices in a PPA configuration chain, the second device's RDYnBSY pin will be high during the time that the first device is being configured. When the first device has been successfully configured, it will driven nCEO low to activate the next device in the chain and drive its RDYnBSY pin high. Therefore, since RDYnBSY signal is driven high before configuration and after configuration before entering user-mode, the device being configured will govern the output of the AND gate.

The nRS signal can be used in multi-device PPA chain since the APEX 20KE or APEX 20KC device will tri-state its DATA[7..0] pins before configuration and after configuration before entering user-mode to avoid contention. Therefore, only the device that is currently being configured will respond to the nRS strobe by asserting DATA7.

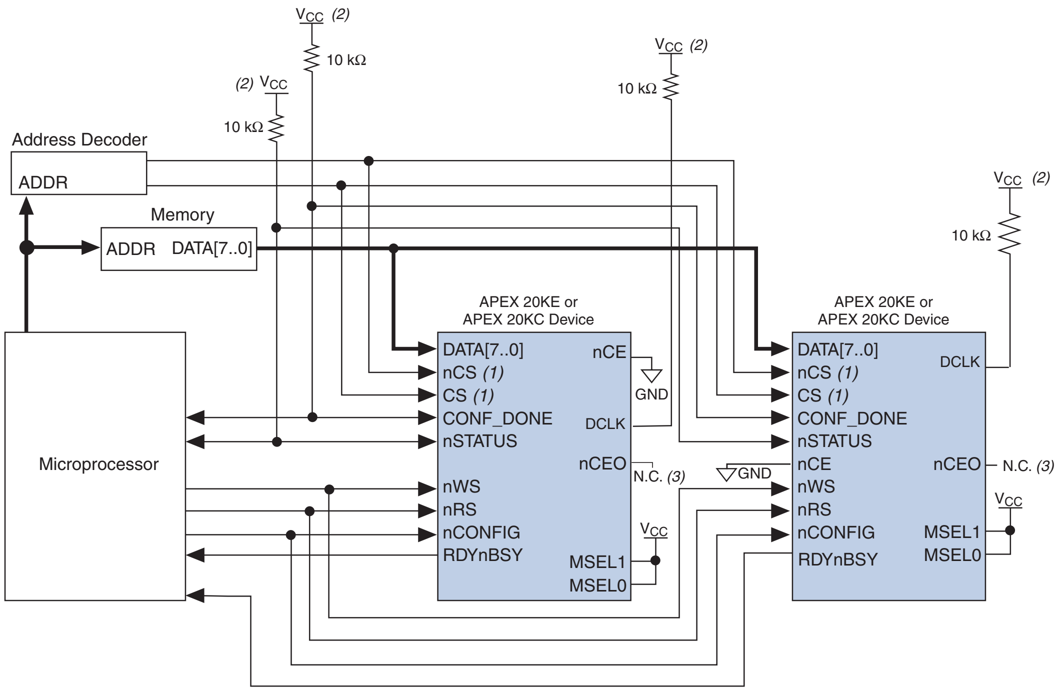
All other configuration pins (nCONFIG, nSTATUS, DATA[7..0], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition (with a low pulse of at least 8 µs) on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DATA[7..1]`, `nCS`, `CS`, `nWS`, `nRS` and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 7–22 shows multi-device PPA configuration when both devices are receiving the same configuration data.

Figure 7–22. Multiple-Device PPA Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 7–22:

(1) If not used, the `CS` pin can be connected to V_{CC} directly. If not used, the `nCS` pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

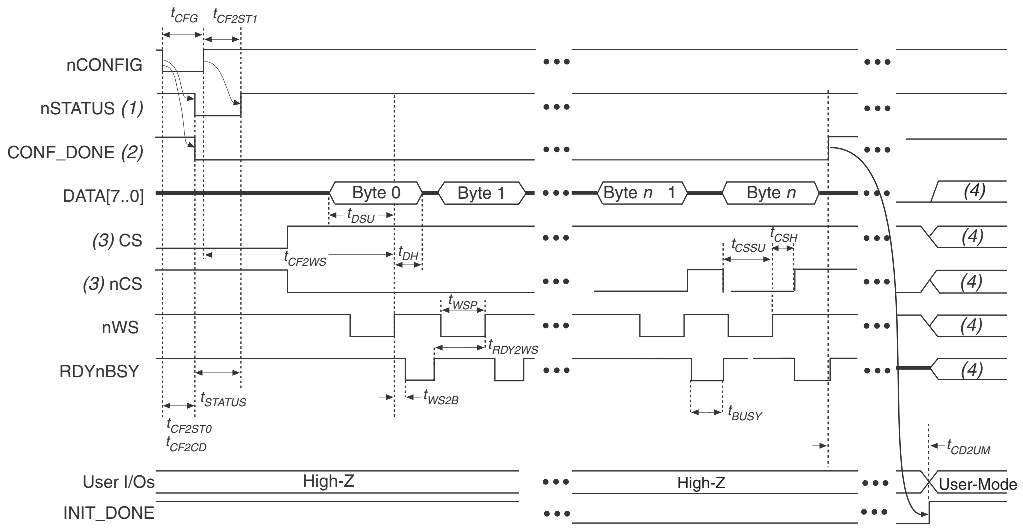
(3) The `nCEO` pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure APEX 20KE and APEX 20KC devices with other Altera devices that support PPA configuration, such as Stratix®, Mercury, APEX II, ACEX 1K, and FLEX 10KE devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 7–23 shows the timing waveform for the PPA configuration scheme using a microprocessor.

Figure 7–23. APEX 20KE & APEX 20KC PPA Configuration Timing Waveform



Notes to Figure 7–23:

(1) Upon power-up, the APEX 20KE or APEX 20KC device holds `nSTATUS` low for not more than 5 µs after V_{CCINT} reaches its minimum requirement.

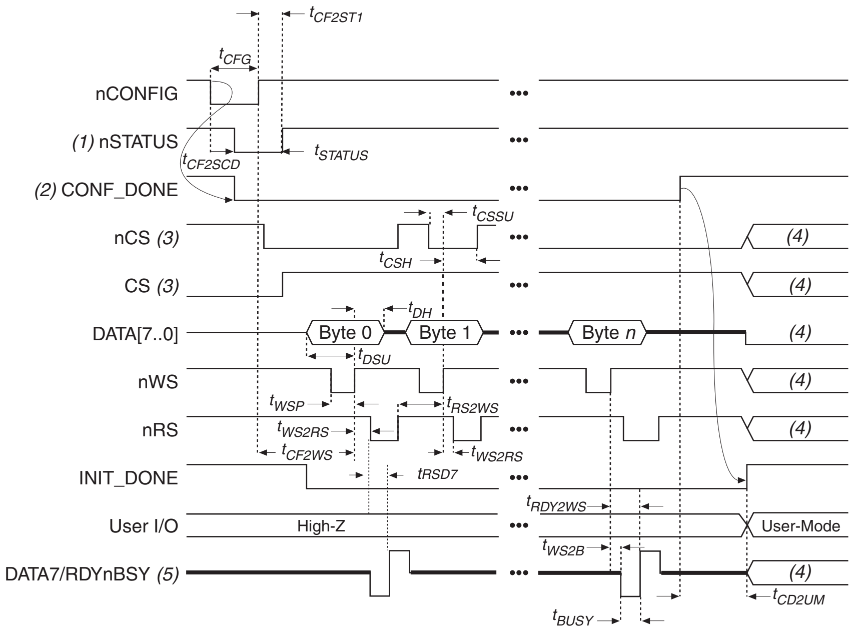
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS` and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

Figure 7–24 shows the timing waveform for the PPA configuration scheme when using a strobed `nRS` and `nWS` signal.

Figure 7–24. APEX 20KE & APEX 20KC PPA Configuration Timing Waveform Using nRS & nWS



Notes to Figure 7–24:

(1) Upon power-up, the APEX 20KE or APEX 20KC device holds `nSTATUS` low for not more than 5 µs after V_{CCINT} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS`, and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

(5) `DATA7` is a bidirectional pin. It is an input for configuration data input, but it is an output to show the status of `RDYnBSY`.

Table 7–6 defines the timing parameters for APEX 20KE or APEX 20KC devices for PPA configuration.

Table 7–6. PPA Timing Parameters for APEX 20KE & APEX 20KC Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 (1)	µs

Table 7–6. PPA Timing Parameters for APEX 20KE & APEX 20KC Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.1	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 7–6:

(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.

(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 for APEX 20KE and APEX 20KC devices to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2 in* the Configuration Handbook.

JTAG Configuration

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The JTAG circuitry can also be used to shift configuration data into the device.

For more information on JTAG boundary-scan testing, see *Application Note 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

A device operating in JTAG mode uses four required pins, TDI, TDO, TMS, and TCK, and one optional pin, TRST. All user I/O pins are tri-stated during JTAG configuration. APEX 20KE and APEX 20KC devices are designed such that JTAG instructions have precedence over any device configuration modes. This means that JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of APEX 20KE and APEX 20KC FPGAs during PS configuration, PS configuration will be terminated and JTAG configuration will begin.

Table 7–7 explains each JTAG pin's function.

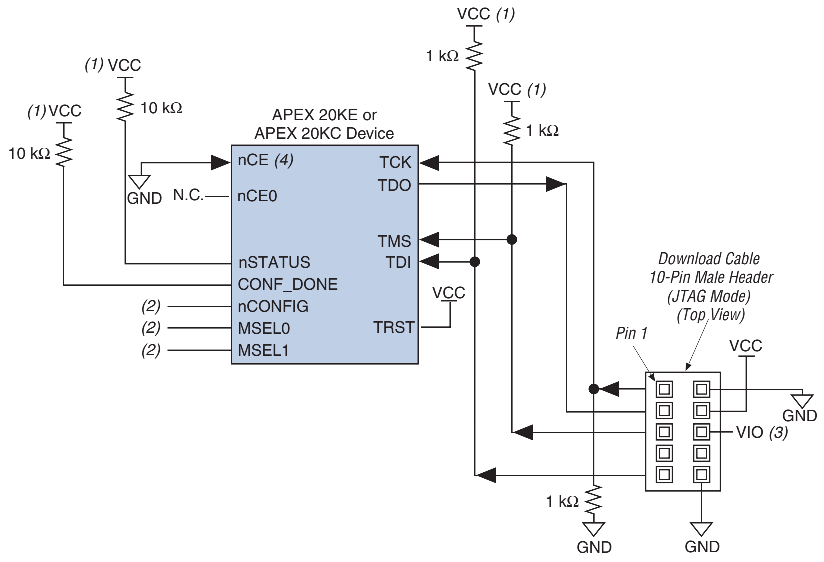
Table 7–7. JTAG Pin Descriptions

Pin	Description	Function
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The TRST pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

If V_{CCIO} of the bank where the JTAG pins reside, are tied to 3.3-V, both the I/O pins and JTAG TDO port will drive at 3.3-V levels.

During JTAG configuration, data can be downloaded to the device on the PCB through the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV header. Configuring devices through a cable is similar to programming devices in-system, except the TRST pin should be connected to V_{CC}. This ensures that the TAP controller is not reset. Figure 7–25. shows JTAG configuration of a single APEX 20KE or APEX 20KC device.

Figure 7–25. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 7–25:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (VIO pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The nCONFIG, MSEL0, and MSEL1 pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect nCONFIG to V_{CC}, and MSEL0 and MSEL1 to ground.

(3) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV cable, this pin is a no connect. In the USB Blaster and ByteBlaster II cable, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

(4) nCE must be connected to GND or driven low for successful JTAG configuration.

To configure a single device in a JTAG chain, the programming software places all other devices in BYPASS mode. In BYPASS mode, devices pass programming data from the TDI pin to the TDO pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the TDO pin one clock cycle later.

APEX 20KE and APEX 20KC devices have dedicated JTAG pins that always function as JTAG pins. JTAG testing can be performed on APEX 20KE and APEX 20KC devices both before and after configuration, but not during configuration. The chip-wide reset (`DEV_CLRn`) and chip-wide output enable (`DEV_OE`) pins on APEX 20KE and APEX 20KC devices do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of APEX 20KE and APEX 20KC devices, the dedicated configuration pins should be considered. Table 7–8 shows how these pins should be connected during JTAG configuration.

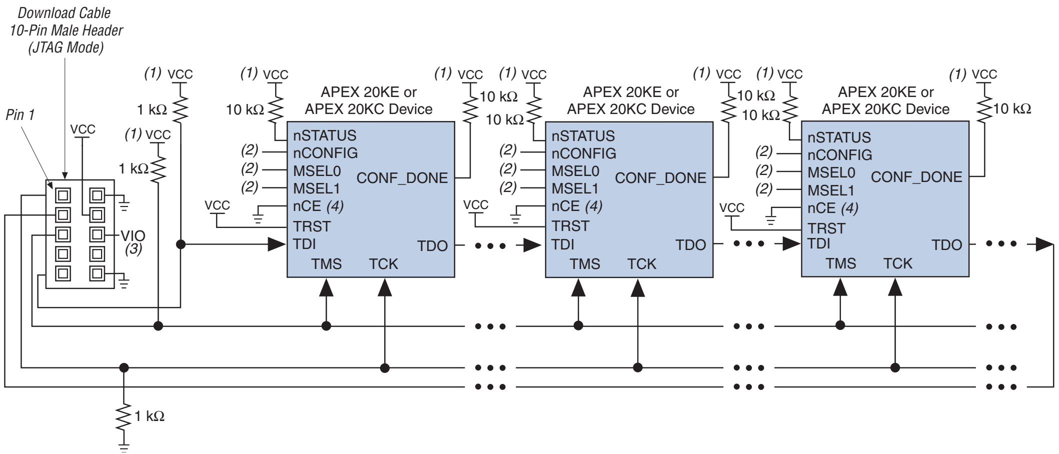
Table 7–8. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all APEX 20KE and APEX 20KC devices in the chain, `nCE` should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, PPS or PPA configuration chains, the `nCE` pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all APEX 20KE and APEX 20KC devices in the chain, `nCEO` can be left floating or connected to the `nCE` of the next device. See nCE description above.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie both pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling high via a resistor, or driven by some control circuitry.
nSTATUS	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `nSTATUS` pin should be pulled up to V_{CC} individually. `nSTATUS` pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 10-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `CONF_DONE` pin should be pulled up to V_{CC} individually. `CONF_DONE` going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.
DATA0	Should not be left floating. Drive low or high, whichever is more convenient on your board.

When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. When four or more devices are connected in a JTAG chain, Altera recommends buffering the `TCK`, `TDI`, and `TMS` pins with an on-board buffer.

JTAG-chain device programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 7–26 shows multi-device JTAG configuration.

Figure 7–26. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 7–26:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to GND or driven low during JTAG configuration. In multi-device PS, PPS, and PPA configuration chains, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device will drive `nCE` of the next device low when it has successfully been JTAG configured.

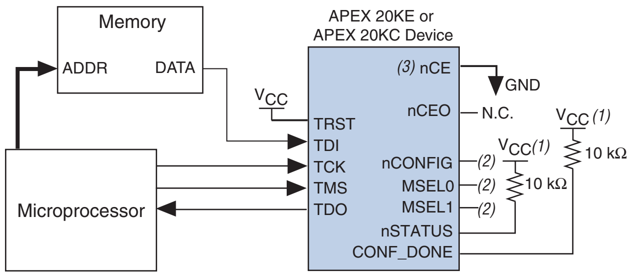
Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

For more information about configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, the software checks the state of `CONF_DONE` through the JTAG port. If `CONF_DONE` is not high, the Quartus II software indicates that configuration has failed. If `CONF_DONE` is high, the software indicates that configuration was successful. When Quartus II generates a Jam file for a multi-device chain, it contains instructions so that all the devices in the chain will be initialized at the same time.

Figure 7–27 shows JTAG configuration of an APEX 20KE or APEX 20KC FPGA with a microprocessor.

Figure 7–27. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 7–27:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

(2) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

(3) `nCE` must be connected to GND or driven low for successful JTAG configuration.

Jam STAPL

Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, see *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the jam player, visit the Altera web site at:

www.altera.com/support/software/download/programming/jam/jam-index.jsp

Configuring APEX 20KE & APEX 20KC FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including APEX 20KE and APEX 20KC FPGAs, through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in RBF format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on other platforms.

For more information on the JRunner software driver, see the *JRunner Software Driver: An Embedded Solution to the JTAG Configuration White Paper* and the source files.

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the APEX 20KE or APEX 20KC device. Table 7–9 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 7–9. Dedicated Configuration Pins on the APEX 20KE & APEX 20KC Device (Part 1 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
MSEL0 MSEL1	N/A	All	Input	Two-bit configuration input that sets the APEX 20KE or APEX 20KC device configuration scheme. See Table 7–3 for the appropriate connections. These pins must remain at a valid state during power-up, before `nCONFIG` is pulled low to initiate a reconfiguration and during configuration.
nCONFIG	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode will cause the FPGA to lose its configuration data, enter a reset state, tri-state all I/O pins, and returning this pin to a logic high level will initiate a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, `nCONFIG` can be tied directly to V_{CC} or to the configuration device's `nINIT_CONF` pin. For succesful configuration of APEX 20KE devices, `nCONFIG` must be tied to V_{CCINT} through a 10-kΩ pull-up resistor.

Table 7–9. Dedicated Configuration Pins on the APEX 20KE & APEX 20KC Device (Part 2 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nSTATUS	N/A	All	Bidirectional open-drain	The FPGA drives `nSTATUS` low immediately after power-up and releases it within 5 µs. (When using a configuration device, the configuration device holds `nSTATUS` low for up to 200 ms.) Status output. If an error occurs during configuration, `nSTATUS` is pulled low by the target device. Status input. If an external source drives the `nSTATUS` pin low during configuration or initialization, the target device enters an error state. Driving `nSTATUS` low after configuration and initialization does not affect the configured device. If a configuration device is used, driving `nSTATUS` low will cause the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on `nSTATUS` in user-mode, the FPGA will not reconfigure. To initiate a reconfiguration, `nCONFIG` must be pulled low. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. For succesful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, `use` an external 10-kΩ pull-up resistor. If internal pull-up resistors on the enhanced configuration devices are used, external 10-kΩ pull-up resistors should not be used on these pins.
CONF_DONE	N/A	All	Bidirectional open-drain	Status output. The target FPGA drives the `CONF_DONE` pin low before and during configuration. Once all configuration data is received without error and the initialization cycle starts, the target device releases `CONF_DONE`. Status input. After all data is received and `CONF_DONE` goes high, the target device initializes and enters user mode. Driving `CONF_DONE` low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. For successful configuration of APEX 20KE and APEX 20KC devices using EPC2 devices, use an external 10-kΩ pull-up resistor. If internal pull-up resistors on the enhanced configuration devices are used, external 10-kΩ pull-up resistors should not be used on these pins.

Table 7–9. Dedicated Configuration Pins on the APEX 20KE & APEX 20KC Device (Part 3 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCE	N/A	All	Input	Active-low chip enable. The `nCE` pin activates the device with a low signal to allow configuration. The `nCE` pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, `nCE` of the first device is tied low while its `nCEO` pin is connected to `nCE` of the next device in the chain. The `nCE` pin must also be held low for successful JTAG programming of the FPGA.
nCEO	N/A	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's `nCE` pin. The `nCEO` of the last device in the chain is left floating.
DCLK	N/A	Synchronous configuration schemes (PS and PPS)	Input	Clock input used to clock data from an external source into the target device. Data is latched into the FPGA on the rising edge of `DCLK`. In PPA mode, `DCLK` should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, `DCLK` will be driven low after configuration is done. In schemes that use a control host, `DCLK` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA0	N/A	All	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the `DATA0` pin. After configuration, EPC1 and EPC1441 devices tri-state this pin, while enhanced and EPC2 devices drive this pin high. In schemes that use a control host, `DATA0` should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA[7..1]	I/O	Parallel configuration schemes (PPS and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on `DATA[7..0]`. In serial configuration schemes, they function as user I/O pins during configuration, which means they are tri-stated. After PPA or PPS configuration, `DATA[7..1]` are available as a user I/O pins and the state of these pin depends on the Dual-Purpose Pin settings.

Table 7–9. Dedicated Configuration Pins on the APEX 20KE & APEX 20KC Device (Part 4 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
DATA7	I/O	PPA	Bidirectional	In the PPA configuration scheme, the `DATA7` pin presents the `RDYnBSY` signal after the `nRS` signal has been strobed low. In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `DATA7` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nWS	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the `DATA[7..0]` pins. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nWS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nRS	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the `RDYnBSY` signal on the `DATA7` pin. If the `nRS` pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nRS` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.

Table 7–9. Dedicated Configuration Pins on the APEX 20KE & APEX 20KC Device (Part 5 of 5)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
RDYnBSY	I/O	PPS and PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPS and PPA configuration schemes, this pin will drive out high after power-up, before configuration and after configuration before entering user-mode. In non-PPS and non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPS and PPA configuration, `RDYnBSY` is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nCS/CS	I/O	PPA	Input	Chip-select inputs. A low on `nCS` and a high on `CS` select the target device for configuration. The `nCS` and `CS` pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the `nCS` or `CS` pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, `nCS` can be tied to ground while `CS` is toggled to control configuration. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, `nCS` and `CS` are available as a user I/O pins and the state of these pins depends on the dual-purpose pin settings.

Table 7–10 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration they function as user I/O pins, which means they are tri-stated with weak pull-up resistors.

Table 7–10. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
CLKUSR	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software
INIT_DONE	N/A if option is on. I/O if option is off.	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 10-kΩ 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 and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the *Enable INIT_DONE output* option in the Quartus II software.
DEV_OE	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the *Enable device-wide output enable (DEV_OE)* option in the Quartus II software.
DEV_CLRn	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the *Enable device-wide reset (DEV_CLRn)* option in the Quartus II software.

JTAG pins must be kept stable before and during configuration. JTAG pin stability prevents accidental loading of JTAG instructions. Table 7–11 describes the dedicated JTAG pins.

Table 7–11. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
TDI	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine.Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

APEX 20KE Power Sequencing

The following guidelines explain how to manage device power sequencing for APEX 20KE devices. These guidelines apply to all configuration schemes.

☞ Altera has enhanced the APEX II and APEX 20KC devices, so you do not need to follow these guidelines for those devices. A system designed for an APEX 20KE device can successfully configure an APEX II or APEX 20KC device.

The APEX 20KE logic array and I/O pins can operate on different power supplies. V_{CCINT} powers the logic array, and each I/O bank has a separate V_{CCIO} supply.

These guidelines will allow you to configure APEX 20KE devices upon power-up as well as recover from power brown-out conditions. Specifically, V_{CCINT} can decrease to any voltage, and the device will successfully reconfigure when power is restored. During the brown-out condition, APEX 20KE devices may lose configuration if V_{CCINT} falls below the minimum V_{CCINT} device specification. If V_{CCINT} drops below the specified operating range, the APEX 20KE device resets and drives nSTATUS low. When V_{CCINT} is in the specified operating range, nSTATUS will be released and configuration will begin.

☞ Stratix, Stratix GX, Cyclone, APEX II, APEX 20KC, Mercury, ACEX 1K, FLEX 10K, and FLEX 6000 devices' power supplies (V_{CCINT} and V_{CCIO}) can be powered up in either order.

When configuring APEX 20KE devices in the same configuration chain as other Altera FPGAs, you must follow these guidelines.

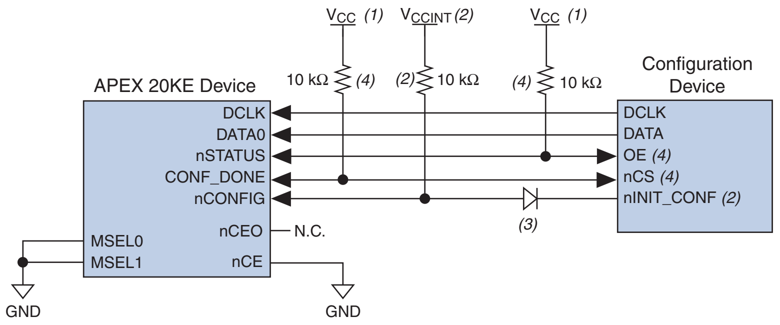
Power Sequencing Considerations

If the APEX 20KE device is configured by an external host, such as a microprocessor, ensure that the configuration controller holds nCONFIG low during power-up and then drives nCONFIG high to begin configuration when all power supplies are stable. This applies for all possible power-up sequences. External pull-ups used on nCONFIG, nSTATUS, and CONF_DONE should be 10 kΩ.

To ensure recovery from brown-out conditions and successful configuration between APEX 20KE devices and configuration devices in all possible power-up sequences, pull nCONFIG up to V_{CCINT} through a 10-kΩ resistor and hold nCONFIG low for the entire time that the two supplies are powering up to their specified operating ranges. All other external pull-up resistors used on nSTATUS, and CONF_DONE should also be 10 kΩ.

When using enhanced configuration devices or EPC2 device, nCONFIG of the FPGA can be connected to nINIT_CONF, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. Figure 7–28 shows the board connections used to support the initiate configuration JTAG instruction with the nINIT_CONF pin.

Figure 7–28. Configuring an APEX 20KE Device Using a Configuration Device



Notes to Figure 7–28:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CCINT} through a 10-kΩ resistor.

(3) The `nINIT_CONF` pin has an internal pull-up resistor to 3.3 V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's nCONFIG pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), this diode is not needed.

(4) The enhanced configuration devices' and EPC2 devices' OE and `nCS` pins have internal programmable pull-up resistors. For successful configuration of APEX 20KE devices, you should use external 10-kΩ pull-up resistors. 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 `nINIT_CONF` pin is an open-drain output and has an internal pull-up resistor to 3.3-V that is always active. Since a 10-kΩ pull-up to V_{CCINT} is required to successfully configure APEX 20KE devices, you need to isolate the 1.8-V V_{CCINT} from the configuration device's 3.3-V supply. To isolate the 1.8-V and 3.3-V power supplies, add a diode between the APEX 20KE device's `nCONFIG` pin and the configuration device's `nINIT_CONF` pin. Select a diode with a threshold voltage (V_T) less than or equal to 0.7 V. The diode will make the `nINIT_CONF` pin an open-drain pin; the pin will only be able to drive low or tri-state.

If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CCINT} through a 10-kΩ resistor and the isolating diode is not needed.

Use these guidelines to ensure a successful power-up and configuration:

- V_{CCINT} is powered before V_{CCIO}—These guidelines should be followed for applications where the sequence of the power supplies is unknown. It is highly recommended that you follow these guidelines to ensure successful configuration of your APEX 20KE device.
- V_{CCINT} is powered after V_{CCIO}—These guidelines should be followed if V_{CCIO} is powered before V_{CCINT}.

V_{CCINT} is Powered Before V_{CCIO}

If V_{CCINT} is powered before V_{CCIO}, the `nCONFIG` signal must be held low for the entire time that the two supplies are powering up to their specified operating ranges. The recommendations in this section should also be followed for applications where the sequence of the power supplies is unknown (e.g., hot-socketing applications). These guidelines should be followed for all configuration schemes using a configuration device or an external host, such as a microprocessor.

For more details on APEX 20KE and configuration device voltage supply specifications, refer to the *APEX 20K Programmable Logic Device Family Data Sheet* and the *Configuration Devices for Altera FPGAs Data Sheet* or *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet*, respectively.

Use one of the following methods to hold `nCONFIG` low:

- Use a power-monitoring circuit on the board. This circuit will drive `nCONFIG` low when V_{CCINT} and V_{CCIO} are out of range, and then release `nCONFIG`. `nCONFIG` can then be externally pulled up when V_{CCINT} returns to a normal operating level. National Semiconductor offers a small power-on reset circuit (part number LP3470) for a power monitor.
- Many voltage regulators offer a power-good signal that can be used to hold `nCONFIG` low during power-up. If there are multiple regulators, use the power-good signal on the regulator that powers up last. If the power sequence is unknown, the power-good signals can be `AND`ed in a discrete device.
- A microcontroller or intelligent host can externally drive `nCONFIG` low while both the V_{CCINT} and V_{CCIO} supplies are being powered.

V_{CCINT} is Powered After V_{CCIO}

If the APEX 20KE device is configured by an external host, such as a microprocessor, ensure that the configuration controller holds `nCONFIG` low during power-up and then drives `nCONFIG` high when all power supplies are stable to begin of configuration.

If the APEX 20KE device is configured using a configuration device and V_{CCINT} is powered up during or after the configuration device has exited POR (released `nSTATUS/OE` signal), `nCONFIG` must be tied to V_{CCINT} through a 10-kΩ resistor. The configuration device will exit POR 200 ms (maximum) after power-up.

When using an enhanced configuration device to configure any Altera FPGAs, including APEX 20KE devices, the V_{CCINT} of the FPGA must be powered before the enhanced configuration device exits POR (signaled by the low-to-high transition on the `OE/nSTATUS` signal). 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 FPGA is not powered up after the enhanced configuration device exits POR, the `CONF_DONE` signal will be high since the pull-up resistor is pulling this signal high. When the enhanced configuration device exits POR, `OE` is released and pulled high by a pull-up resistor. If the enhanced configuration device sees its `nCS/CONF_DONE` signal also high, the configuration device will go into an idle state because it sees the FPGA is already configured. `DATA` and `DCLK` will not toggle and the enhanced configuration device will only exit this idle state 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 its power-up before the enhanced configuration device exits POR. Alternatively the `nSTATUS/OE` line can be held low by an open-drain signal until after the V_{CCINT} power supply is stable.

For more information about power sequencing of enhanced configuration devices, see the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* in the Configuration Handbook.

8. Configuring Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K Devices

CF51006-2.0

Introduction

Mercury™, APEX™ 20K (2.5 V), ACEX® 1K, and FLEX® 10K devices can be configured using one of four configuration schemes. All configuration schemes use either a microprocessor or configuration device.

This section covers how to configure Mercury, APEX 20K (2.5 V), ACEX 1K and FLEX 10K devices. APEX 20K (non-E and non-C) devices use a 2.5-V voltage supply for V_{CCINT}, while APEX 20KE and APEX 20KC devices use a 1.8-V voltage supply for V_{CCINT}. If your target FPGA is an APEX 20K device which uses a 1.8-V V_{CCINT}. For more information, see *Configuring APEX 20KE & APEX 20KC Devices* in the Configuration Handbook.

Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices can be configured using the passive serial (PS), passive parallel synchronous (PPS), passive parallel asynchronous (PPA), and Joint Test Action Group (JTAG) configuration schemes. The configuration scheme used is selected by driving the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device `MSEL1` and `MSEL0` pins either high or low as shown in Table 8–1. If your application only requires a single configuration mode, the `MSEL` pins can be connected to V_{CC} (V_{CCIO} of the I/O bank where the `MSEL` pin resides) or to ground. If your application requires more than one configuration mode, you can switch the `MSEL` pins after the FPGA is configured successfully. Toggling these pins during user-mode does not affect the device operation; however, the `MSEL` pins must be valid before a reconfiguration is initiated.

Table 8–1. Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K Configuration Schemes

MSEL1	MSEL0	Configuration Scheme
0	0	PS
1	0	PPS
1	1	PPA
(1)	*(1)*	JTAG Based *(2)*

Notes to Table 8–1:

(1) Do not leave the `MSEL` pins floating; connect them to a low- or high-logic level. These pins support the non-JTAG configuration scheme used in production. If only JTAG configuration is used, you should connect the `MSEL` pins to ground.

(2) JTAG-based configuration takes precedence over other configuration schemes, which means `MSEL` pin settings are ignored.

Tables 8–2 through 8–5 show the approximate configuration file sizes for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices.

Table 8–2. Mercury Raw Binary File (.rbf) Sizes

Device	Data Size (Bits)	Data Size (Bytes)
EP1M120	1,303,120	162,890
EP1M350	4,394,032	549,254

Table 8–3. APEX 20K (2.5 V) Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EP20K100	993,360	124,170
EP20K200	1,950,800	243,850
EP20K400	3,880,720	485,090

Table 8–4. ACEX 1K Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EP1K10	159,160	19,895
EP1K30	473,720	59,215
EP1K50	784,184	98,023
EP1K100	1,335,720	166,965

Table 8–5. FLEX 10K Raw Binary File (.rbf) Sizes

Devices	Data Size (Bits)	Data Size (Bytes)
EPF10K30E	473,720	59,215
EPF10K50E	784,184	98,023
EPF10K50S	784,184	98,023
EPF10K100B	1,200,000	150,000
EPF10K100E	1,335,720	166,965
EPF10K130E	1,838,360	229,795
EPF10K200E	2,756,296	344,537
EPF10K200S	2,756,296	344,537
EPF10K10A	120,000	15,000
EPF10K30A	406,000	50,750
EPF10K50V	621,000	77,625
EPF10K100A	1,200,000	150,000
EPF10K130V	1,600,000	200,000
EPF10K250A	3,300,000	412,500
EPF10K10	118,000	14,750
EPF10K20	231,000	28,875
EPF10K30	376,000	47,000
EPF10K40	498,000	62,250
EPF10K50	621,000	77,625
EPF10K70	892,000	111,500
EPF10K100	1,200,000	150,000

Use the data in Tables 8–2 through 8–5 only to estimate the file size before design compilation. Different configuration file formats, such as a Hexidecimal (**.hex**) or Tabular Text File (**.ttf**) format, will have different file sizes. However, for any specific version of the Quartus® II or MAX+PLUS® II software, all designs targeted for the same device will have the same configuration file size.

The following chapter describes in detail how to configure Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using the supported configuration schemes. The last section describes the device configuration pins available. In this chapter, the generic term device(s) or FPGA(s) will include all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices.

For more information on setting device configuration options or creating configuration files, *see Section II, Software Settings, in Volume 2*.

Passive Serial Configuration

You can perform Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K PS configuration using an Altera configuration device, an intelligent host (e.g., a microprocessor or Altera® MAX® device), or a download cable.

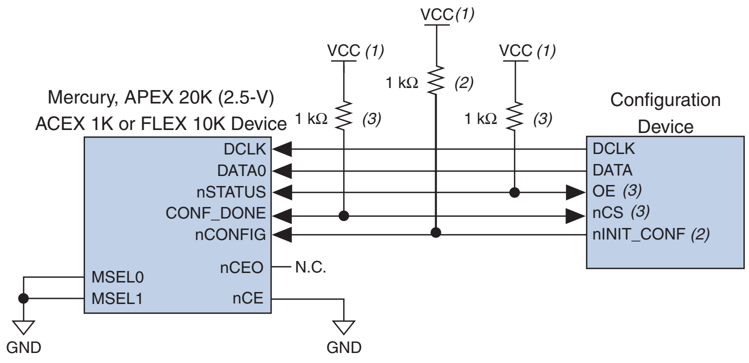
PS Configuration Using a Configuration Device

You can use an Altera configuration device, such as an enhanced configuration device, EPC2, or EPC1 device, to configure Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a serial configuration bitstream. Configuration data is stored in the configuration device. Figure 8–1 shows the configuration interface connections between Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K devices and a configuration device.

The figures in this chapter only show the configuration-related pins and the configuration pin connections between the configuration device and the FPGA.

For more information on the configuration device and flash interface pins (e.g., `PGM[2..0]`, `EXCLK`, `PORSEL`, `A[20..0]`, and `DQ[15..0]`), see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* in the Configuration Handbook.

Figure 8–1. Single Device PS Configuration Using a Configuration Device



Notes to Figure 8–1:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

The value of the internal pull-up resistors on the enhanced configuration devices and EPC2 devices can be found in the Operating Conditions table of the *Enhanced Configuration Devices (EPC4, EPC8, & EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the Configuration Handbook.

When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the `INIT_CONF` JTAG instruction to initiate FPGA configuration. 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up on the `nINIT_CONF` pin is always active in enhanced configuration devices and EPC2 devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. The configuration device also goes through a POR delay to allow the power supply to stabilize. The POR time for EPC2, EPC1, and EPC1441 devices is 200 ms (maximum), and for enhanced configuration devices, the POR time can be set to either 100 ms or 2 ms, depending on its `PORSEL` pin setting. If the `PORSEL` pin is connected to GND, the POR delay is 100 ms. During this time, the configuration device drives its `OE` pin low. This low signal delays configuration because the `OE` pin is connected to the target device's `nSTATUS` pin. When both devices complete POR, they release their open-drain `OE` or `nSTATUS` pin, which is then pulled high by a pull-up resistor. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

When the power supplies have reached the appropriate operating voltages, the target FPGA senses the low-to-high transition on `nCONFIG` and initiates the configuration cycle. The configuration cycle consists of three stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. The beginning of configuration can be delayed by holding the `nCONFIG` or `nSTATUS` pin low.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the `nSTATUS` pin, which is pulled high by a pull-up resistor. Enhanced configuration and EPC2 devices have an optional internal pull-up on the `OE` pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up resistor is not used, an external 1-kΩ pull-up resistor on the `OE/nSTATUS` line is required. Once `nSTATUS` is released, the FPGA is ready to receive configuration data and the configuration stage begins.

When `nSTATUS` is pulled high, `OE` of the configuration device also goes high and the configuration device clocks data out serially to the FPGA using its internal oscillator. The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device receives configuration data on its `DATA0` pin and the clock is received on the `DCLK` pin. Data is latched into the FPGA on the rising edge of `DCLK`.

After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by a pull-up resistor. Since CONF_DONE is tied to the configuration device's nCS pin, the configuration device is disabled when CONF_DONE goes high. Enhanced configuration and EPC2 devices have an optional internal pull-up resistor on the nCS pin. This option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If this internal pull-up is not used, an external 1-kΩ pull-up resistor on the nCS/CONF_DONE line is required. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device will allow enough clock cycles for proper initialization.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, an external host 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.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. You can turn on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used, it will be high due to an external 1-kΩ pull-up resistor when nCONFIG is low and during the beginning of configuration. Once the option bit to enable INIT_DONE is programmed into the device (during the first frame of configuration data), the INIT_DONE pin goes low. When initialization is complete, the INIT_DONE pin is released and pulled high. This low-to-high transition signals that the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. The enhanced configuration device and EPC2 device drive DCLK low and DATA high (EPC1 devices tri-state DATA) at the end of configuration.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. Since the nSTATUS pin is tied to OE, the configuration device will also be reset. If the *Auto-Restart Configuration After Error* option available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box is turned on, the FPGA automatically initiates reconfiguration if an error occurs. The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When the nSTATUS pin is released and pulled high by a pull-up resistor, the configuration device reconfigures the chain. If this option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

In addition, if the configuration device sends all of its data and then detects that CONF_DONE has not gone high, it recognizes that the FPGA has not configured successfully. Enhanced configuration devices wait for 64 DCLK cycles after the last configuration bit was sent for CONF_DONE to reach a high state. EPC1 and EPC2 devices wait for 16 DCLK cycles. In this case, the configuration device pulls its OE pin low, which in turn drives the target device's nSTATUS pin low. If the *Auto-Restart Configuration After Error* option is set in the software, the target device resets and then releases its nSTATUS pin after a reset time-out period (maximum of 40 µs). When nSTATUS returns high, the configuration device tries to reconfigure the FPGA.

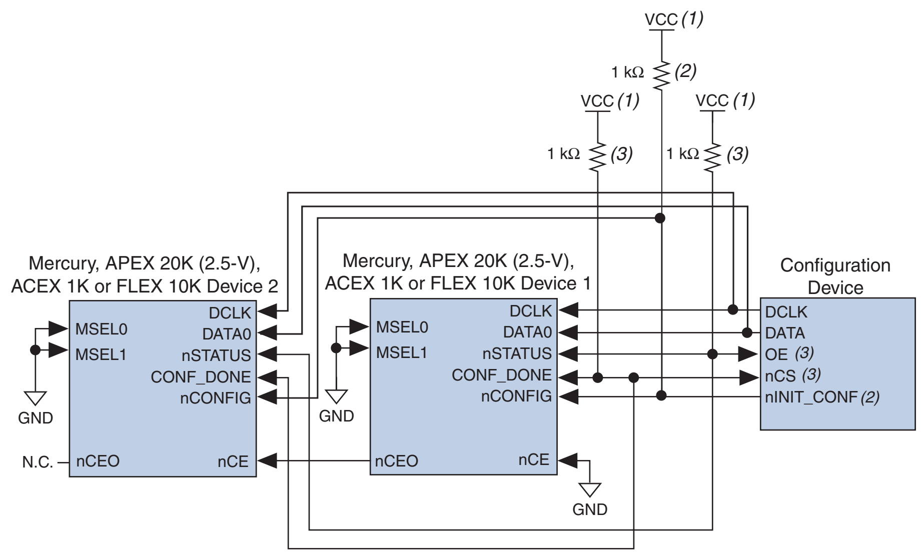
When CONF_DONE is sensed low after configuration, the configuration device recognizes that the target device has not configured successfully; therefore, your system should not pull CONF_DONE low to delay initialization. Instead, use the CLKUSR option to synchronize the initialization of multiple devices that are not in the same configuration chain. Devices in the same configuration chain will initialize together if their CONF_DONE pins are tied together.

If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by pulling the nCONFIG pin low. When nCONFIG is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Since CONF_DONE is pulled low, this will activate the configuration device since it will see its nCS pin drive low. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 8–2 shows how to configure multiple devices with a configuration device. This circuit is similar to the configuration device circuit for a single device, except Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are cascaded for multi-device configuration.

Figure 8–2. Multi-Device PS Configuration Using a Configuration Device



Notes to Figure 8–2*:*

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

When performing multi-device configuration, you must generate the configuration device's Programmer Object File (**.pof**) from each project's SRAM Object File (**.sof**). You can combine multiple SOFs using the Quartus II software.

For more information on how to create configuration files for multi-device configuration chains, *see Section II, Software Settings, in Volume 2*.

In multi-device PS configuration, the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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.

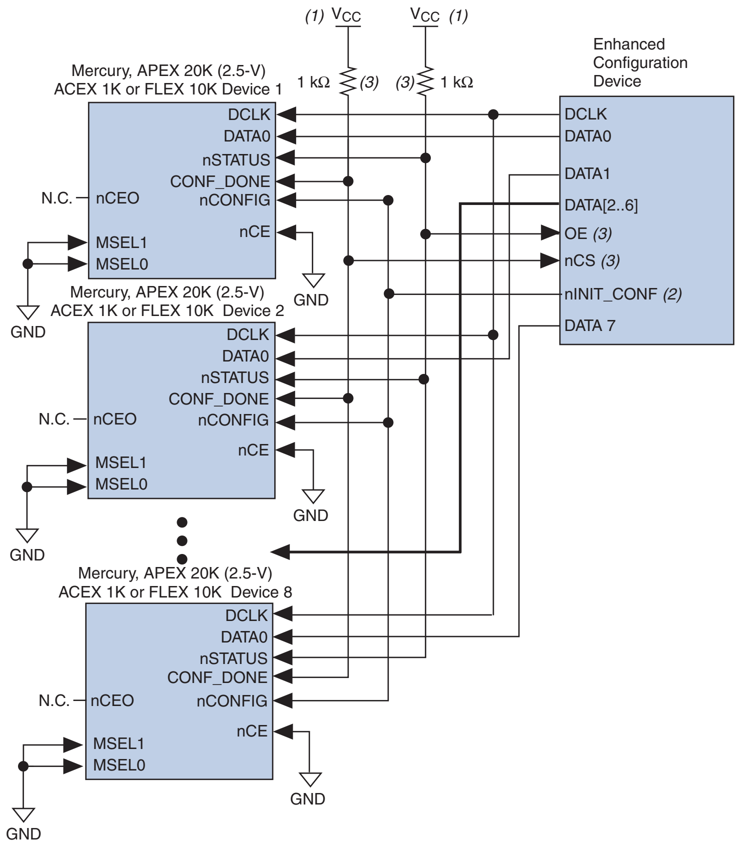
When configuring multiple devices, configuration does not begin until all devices release their OE or nSTATUS pins. Similarly, since all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This low signal drives the OE pin low on the configuration device and drives nSTATUS low on all FPGAs, which causes them to enter a reset state. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the devices will automatically initiate reconfiguration if an error occurs. The FPGAs will release their nSTATUS pins after a reset time-out period (maximum of 40 µs). When all the nSTATUS pins are released and pulled high, the configuration device tries to reconfigure the chain. If the *Auto-Restart Configuration After Error* option is turned off, the external system must monitor nSTATUS for errors and then pulse nCONFIG low to restart configuration. The external system can pulse nCONFIG if nCONFIG is under system control rather than tied to V_{CC}.

Enhanced configuration devices also support parallel configuration of up to eight devices. The n-bit (n = 1, 2, 4, or 8) PS configuration mode allows enhanced configuration devices to concurrently configure FPGAs or a chain of FPGAs. In addition, these devices do not have to be the same device family or density; they can be any combination of Altera FPGAs. An individual enhanced configuration device DATA line is available for each targeted FPGA. Each DATA line can also feed a daisy chain of FPGAs. Figure 8–3 shows how to concurrently configure multiple devices using an enhanced configuration device.

Figure 8–3. Concurrent PS Configuration of Multiple Devices Using an Enhanced Configuration Device


(1) VCC
VCC (1)
1 kΩ
(3)
(3)
1 kΩ
Enhanced Configuration Device
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 1
DCLK
DATA0
nSTATUS
CONF_DONE
nCONFIG
nCE
N.C.
nCEO
MSEL1
MSEL0
GND
GND
DCLK
DATA0
DATA1
DATA[2..6]
OE (3)
nCS (3)
nINIT_CONF (2)
DATA 7
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 2
Mercury, APEX 20K (2.5-V) ACEX 1K or FLEX 10K Device 8

Notes to Figure 8–3:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

The Quartus II software only allows the selection of n-bit PS configuration modes, where n must be 1, 2, 4, or 8. 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 8–6 to select the appropriate configuration mode for the fastest configuration times.

Table 8–6. Recommended Configuration Using n-Bit PS Modes

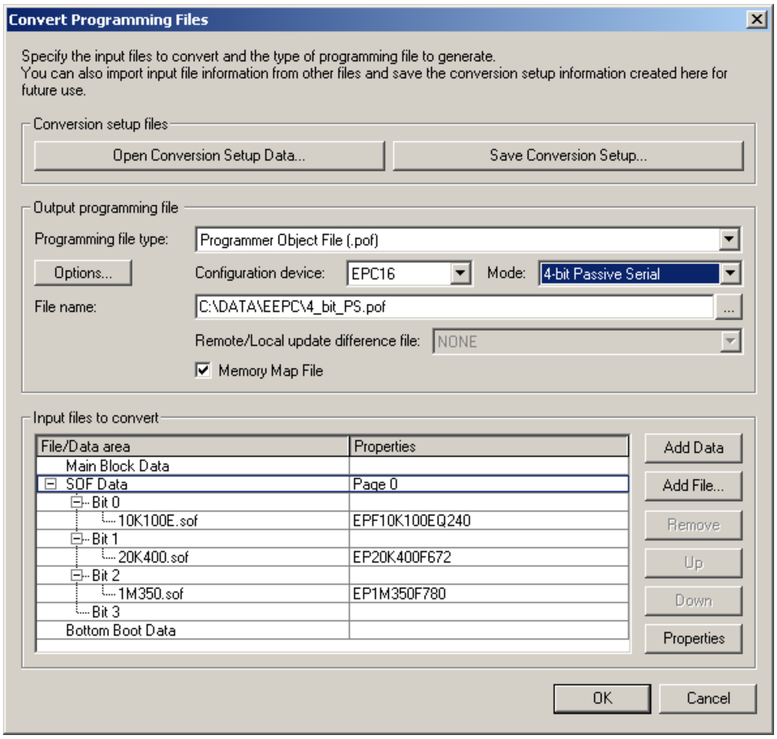
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 8–6:

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

For example, if you configure three FPGAs, you would use the 4-bit PS mode. For the `DATA0`, `DATA1`, and `DATA2` lines, the corresponding SOF data is transmitted from the configuration device to the FPGA. 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 8–4 shows the Quartus II **Convert Programming Files** window (Tools menu) setup for this scheme.

Figure 8–4. Software Settings for Configuring Devices Using n-Bit PS Modes



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: 4-bit Passive Serial

File name: C:\DATA\EEPC\4_bit_PS.pof

Remote/Local update difference file: NONE

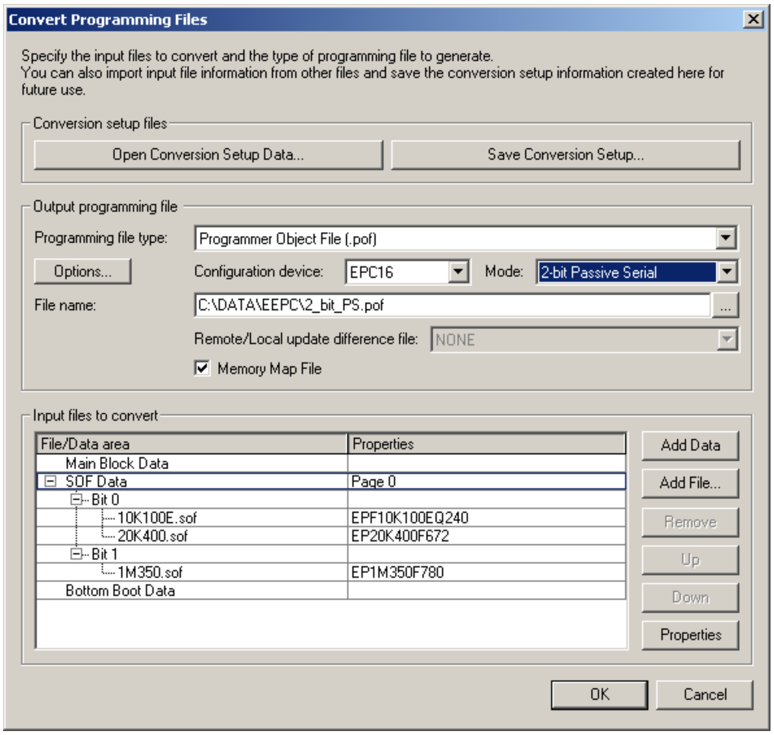
[✓] Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
10K100E.sof	EPF10K100EQ240
Bit 1	
20K400.sof	EP20K400F672
Bit 2	
1M350.sof	EP1M350F780
Bit 3	
Bottom Boot Data	

Alternatively, you can daisy chain two FPGAs 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 FPGAs with `DATA Bit0` (EPF10K100E and EP20K400 devices) and the third device (the EP1M350 device) with `DATA Bit1`. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but can increase the total system configuration time. See Figure 8–5.

Figure 8–5. Software Settings for Daisy Chaining Two FPGAs on 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: C:\DATA\EEPC\2_bit_PS.pof

Remote/Local update difference file: NONE

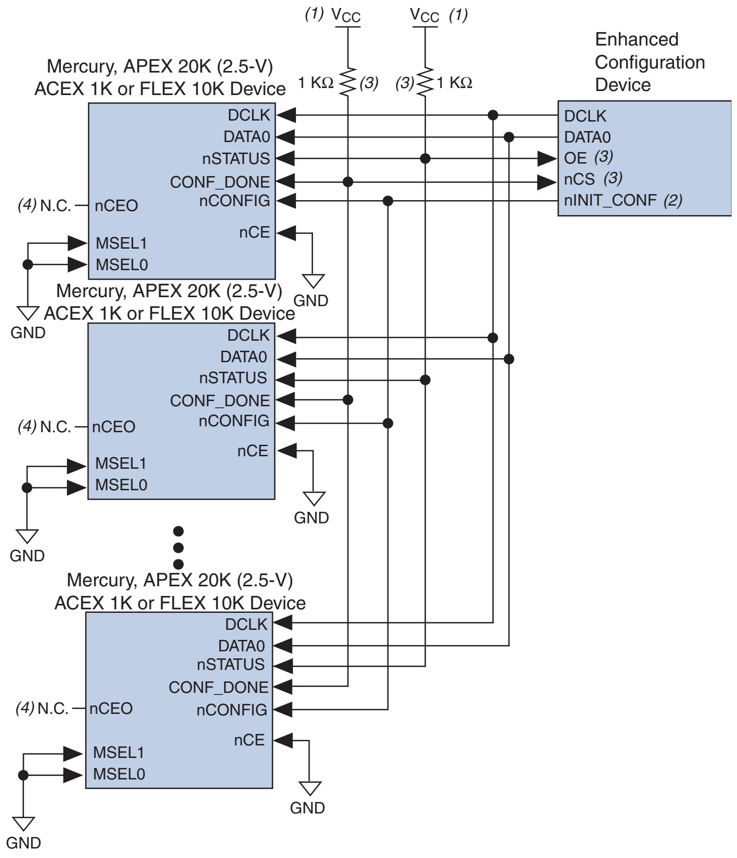
☑ Memory Map File

Input files to convert

File/Data area	Properties
Main Block Data	
SOF Data	Page 0
Bit 0	
10K100E.sof	EPF10K100EQ240
20K400.sof	EP20K400F672
Bit 1	
1M350.sof	EP1M350F780
Bottom Boot Data	

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–6 shows multi-device PS configuration when the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are receiving the same configuration data.

Figure 8–6. Multiple-Device PS Configuration Using an Enhanced Configuration Device When FPGAs Receive the Same Data



Notes to Figure 8–6:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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) The `nCEO` pins of all devices are left unconnected when configuring the same configuration data into multiple devices.

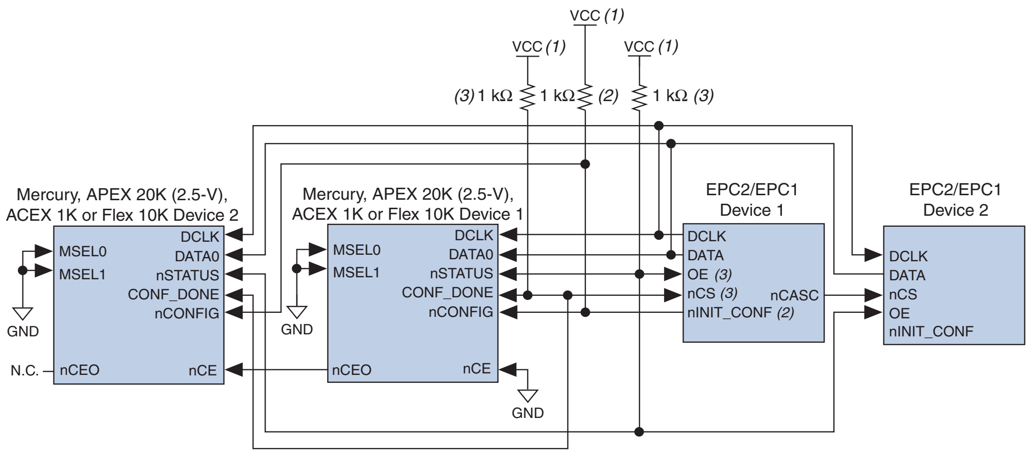
You can cascade several EPC2 or EPC1 devices to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. The first configuration device in the chain is the master configuration device, while the subsequent devices are the slave devices. The master configuration device sends `DCLK` to the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices and to the slave configuration devices. The first EPC device's `nCS` pin is connected to the `CONF_DONE` pins of the FPGAs, while its `nCASC` pin is connected to `nCS` of the next configuration device in the chain. The last device's `nCS` input comes from the previous device, while its `nCASC` pin is left floating. When all data from the first configuration device is sent, it drives `nCASC` low, which in turn drives `nCS` on the next configuration device. Because a configuration device requires less than one clock cycle to activate a subsequent configuration device, the data stream is uninterrupted.

Enhanced configuration devices EPC4, EPC8, and EPC16 cannot be cascaded.

Since all `nSTATUS` and `CONF_DONE` pins are tied together, if any device detects an error, the master configuration device stops configuration for the entire chain and the entire chain must be reconfigured. For example, if the master configuration device does not detect `CONF_DONE` going high at the end of configuration, it resets the entire chain by pulling its `OE` pin low. This low signal drives the `OE` pin low on the slave configuration device(s) and drives `nSTATUS` low on all FPGAs, causing them to enter a reset state. This behavior is similar to the FPGA detecting an error in the configuration data.

Figure 8–7 shows how to configure multiple devices using cascaded EPC2 or EPC1 devices.

Figure 8–7. Multi-Device PS Configuration Using Cascaded EPC2 or EPC1 Devices



Notes to Figure 8–7:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) has an internal pull-up resistor that is always active, meaning 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(3) The enhanced configuration devices' and EPC2 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.

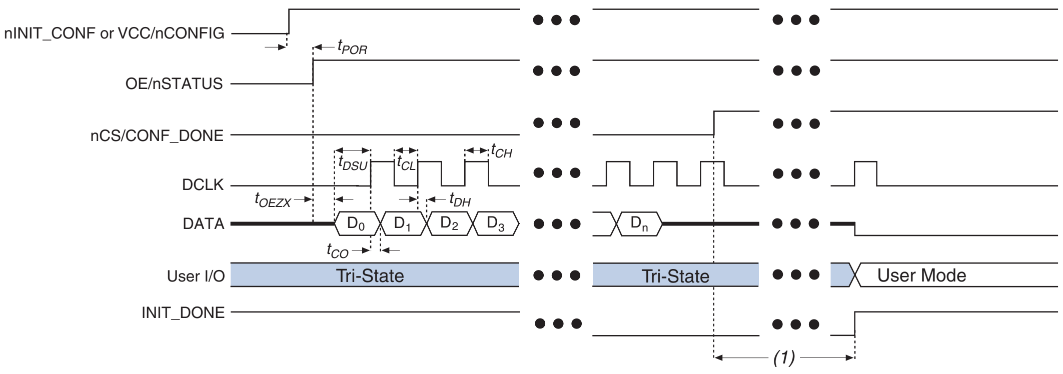
When using enhanced configuration devices or EPC2 devices, `nCONFIG` of the FPGA can be connected to `nINIT_CONF`, which allows the INIT_CONF JTAG instruction to initiate FPGA configuration. The `nINIT_CONF` pin does not need to be connected if its functionality is not used. If `nINIT_CONF` is not used or not available (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor. An internal pull-up on the `nINIT_CONF` pin is always active in the enhanced configuration devices and the EPC2 devices, which means an external pull-up is not required if `nCONFIG` is tied to `nINIT_CONF`. If multiple EPC2 devices are used to configure a Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K device(s), only the first EPC2 has its `nINIT_CONF` pin tied to the device's `nCONFIG` pin.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 8–8 shows the timing waveform for the PS configuration scheme using a configuration device.

Figure 8–8. PS Configuration Using a Configuration Device Timing Waveform



Note to Figure 8–8:

(1) Mercury devices enter user-mode 136 clock cycles after `CONF_DONE` goes high. APEX 20K devices enter user-mode 40 clock cycles after `CONF_DONE` goes high. The initialization clock can come from the Mercury or APEX 20K internal oscillator or the `CLKUSR` pin. ACEX 1K and FLEX 10K devices enter user-mode 10 clock cycles after CONF_DONE goes high. The initialization clock can come from `DCLK` or `CLKUSR`.

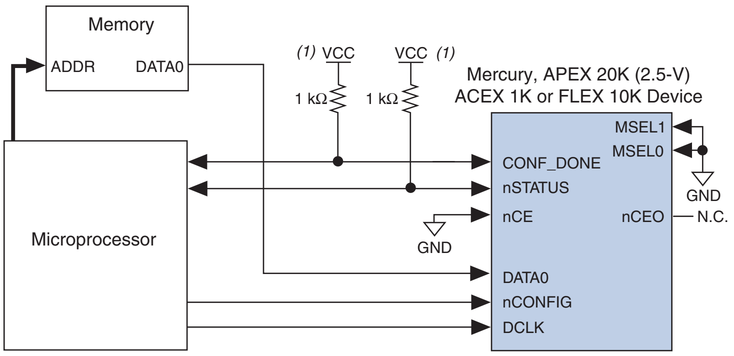
For timing information, refer to the Configuration Handbook Section *Altera Configuration Devices*, the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* or the *Configuration Devices for SRAM-based LUT Devices Data Sheet* in the Configuration Handbook.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, Volume 2* of the Configuration Handbook.

PS Configuration Using a Microprocessor

In the PS configuration scheme, an intelligent host (e.g., a microprocessor or CPLD) can transfer configuration data from a storage device (e.g., flash memory) to the target Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. Configuration data can be stored in RBF, HEX, or TTF format. Figure 8–9 shows the configuration interface connections between the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device and a microprocessor for single device configuration.

Figure 8–9. Single Device PS Configuration Using a Microprocessor



Note to Figure 8–9:

(1) Connect the pull-up resistor to a supply that provides an acceptable input signal for the device.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of three stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

☞ VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released, the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one bit at a time on the DATA0 pin. The least significant bit (LSB) of each data byte must be sent first.

The Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device receives configuration data on its DATA0 pin and the clock is received on the DCLK pin. Data is latched into the FPGA on the rising edge of DCLK. Data is continuously clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device allows enough clock cycles for proper initialization.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, the clock on DCLK is the clock source for initialization. Programming files generated by the Quartus II or MAX+PLUS II software already have these initialization clock cycles included in the file. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. The *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. To ensure DCLK and DATA are not left floating at the end of configuration, the microprocessor must drive them either high or low, whichever is convenient on your board.

Handshaking signals are not used in PS configuration mode. Therefore, the configuration clock (DCLK) speed must be below the specified frequency to ensure correct configuration. No maximum DCLK period exists, which means you can pause configuration by halting DCLK for an indefinite amount of time.

If an error occurs during configuration, the FPGA drives its nSTATUS pin low, resetting itself internally. The low signal on the nSTATUS pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option (available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box) is turned on, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device releases nSTATUS after a reset time-out period (maximum of 40 µs). After nSTATUS is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

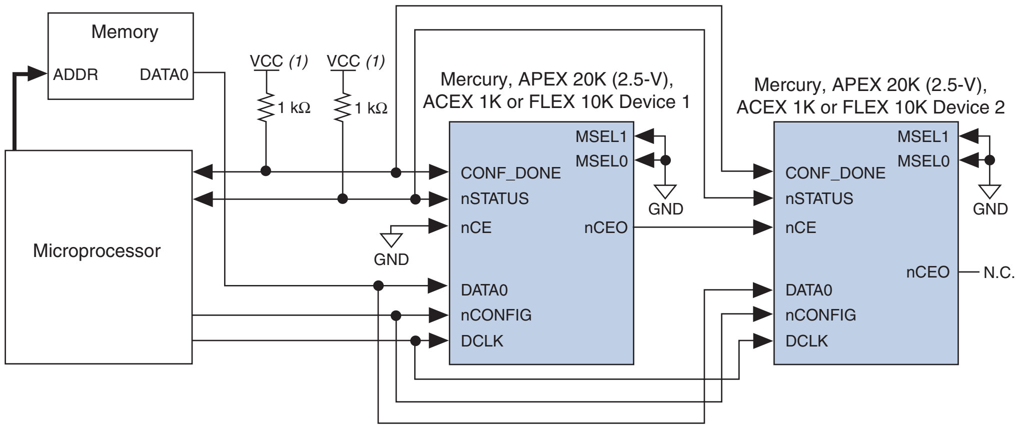
The microprocessor can also monitor the CONF_DONE and INIT_DONE pins to ensure successful configuration. The CONF_DONE pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but CONF_DONE or INIT_DONE have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional CLKUSR pin is being used and nCONFIG is pulled low to restart configuration during device initialization, you need to ensure that CLKUSR continues toggling during the time nSTATUS is low (maximum of 40 µs).

When the FPGA is in user-mode, you can initiate a reconfiguration by transitioning the `nCONFIG` pin low-to-high. The `nCONFIG` pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices and 2 µs for ACEX 1K and Flex 10K devices. When `nCONFIG` is pulled low, the FPGA also pulls `nSTATUS` and `CONF_DONE` low and all I/O pins are tri-stated. Once `nCONFIG` returns to a logic high state and `nSTATUS` is released by the FPGA, reconfiguration begins.

Figure 8–10 shows how to configure multiple devices using a microprocessor. This circuit is similar to the PS configuration circuit for a single device, except Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are cascaded for multi-device configuration.

Figure 8–10. Multi-Device PS Configuration Using a Microprocessor



Note to Figure 8–10:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

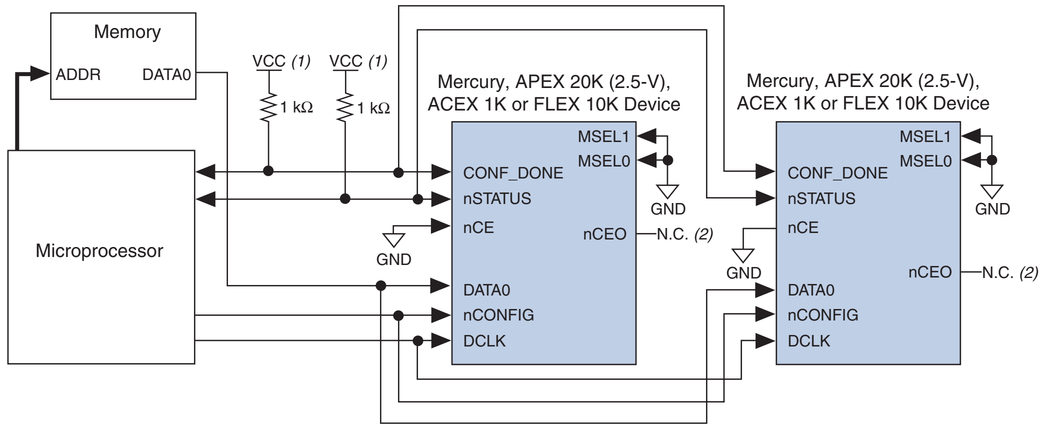
In multi-device PS configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

In your system, you can have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA0, and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they can 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–11 shows multi-device PS configuration when Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are receiving the same configuration data.

Figure 8–11. Multiple-Device PS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–11:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

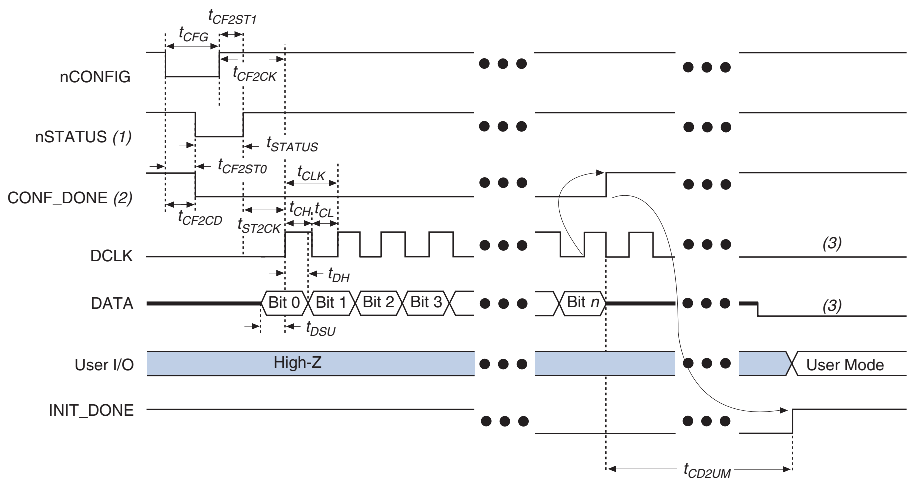
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices with other Altera devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device CONF_DONE and nSTATUS pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 8–12 shows the timing waveform for the PS configuration for Mercury, APEX 20K, ACEX 1K, and FLEX 10KE devices when using a microprocessor.

Figure 8–12. PS Configuration Using a Microprocessor Timing Waveform



Notes to Figure 8–12:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) `DATA0` and `DCLK` should not be left floating after configuration. It should be driven high or low, whichever is more convenient.

Tables 8–7 through 8–10 defines the timing parameters for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices for PS configuration.

Table 8–7. PS Timing Parameters for Mercury Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	21		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 (2)	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 (2)	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	45		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns

Table 8–7. PS Timing Parameters for Mercury Devices (Part 2 of 2) *Note (1)*

Symbol	Parameter	Min	Max	Units
t_{CH}	`DCLK` high time	10		ns
t_{CL}	`DCLK` low time	10		ns
t_{CLK}	`DCLK` period	20		ns
f_{MAX}	`DCLK` frequency		50	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(3)*	6	28	µs

Notes to Table 8–7:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–8. PS Timing Parameters for APEX 20K Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` maximum frequency		33.3	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 8–8:
(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 to obtain this value.

Table 8–9. PS Timing Parameters for ACEX & FLEX 10KE Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		4 *(1)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	5		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	15		ns
t_{CL}	DCLK low time	15		ns
t_{CLK}	DCLK period	30		ns
f_{MAX}	DCLK maximum frequency		33.3	MHz
t_{CD2UM}	CONF_DONE high to user mode *(2)*	0.6	2	µs

Notes to Table 8–9:

(1) This value is obtainable if users do not delay configuration by extending the nSTATUS low pulse width.

(2) The minimum and maximum numbers apply only if DCLK is chosen as the clock source for starting up the device. If the clock source is CLKUSR, multiply the clock period by 10 to obtain this value.

Table 8–10. PS Timing Parameters for FLEX 10K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		4 *(1)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	5		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	30		ns
t_{CL}	DCLK low time	30		ns
t_{CLK}	DCLK period	60		ns

Table 8–10. PS Timing Parameters for FLEX 10K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
f_{MAX}	`DCLK` maximum frequency		16.7	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	0.6	2	µs

Notes to Table 8–10:
(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.
(2) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, in Volume 2* of the Configuration Handbook.

Configuring Using the MicroBlaster Driver

The MicroBlaster™ software driver allows you to configure Altera's FPGAs through the ByteBlasterMV cable in PS mode. The MicroBlaster software driver supports a RBF programming input file and is targeted for embedded passive serial configuration. The source code is developed for the Windows NT operating system, although you can customize it to run on other operating systems. For more information on the MicroBlaster software driver, go to the Altera web site (http://www.altera.com).

PS Configuration Using a Download Cable

In this section, the generic term "download cable" includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable.

In PS configuration with a download cable, an intelligent host (e.g., a PC) transfers data from a storage device to the FPGA via the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cable.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a POR for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

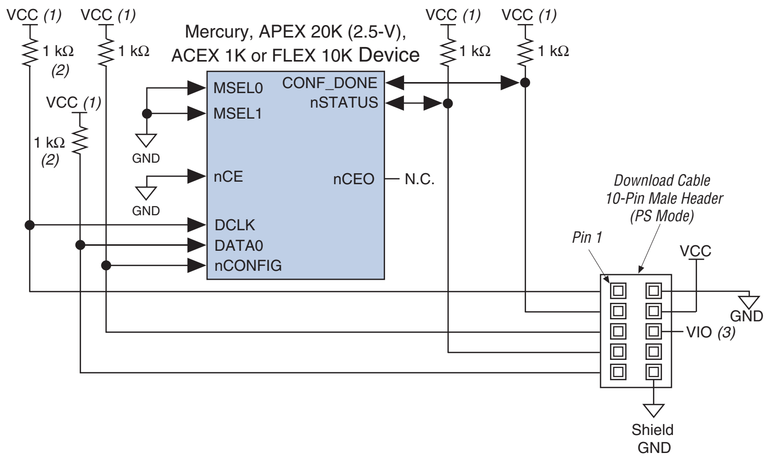
The configuration cycle consists of 3 stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration in this scheme, the download cable generates a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. The programming hardware or download cable then places the configuration data one bit at a time on the device's `DATA0` pin. The configuration data is clocked into the target device until `CONF_DONE` goes high.

When using a download cable, setting the *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs. Additionally, the *Enable user-supplied start-up clock (CLKUSR)* option has no affect on the device initialization since this option is disabled in the SOF when programming the FPGA using the Quartus II programmer and download cable. Therefore, if you turn on the *CLKUSR* option, you do not need to provide a clock on `CLKUSR` when you are configuring the FPGA with the Quartus II programmer and a download cable. Figure 8–13 shows PS configuration for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a USB Blaster, MasterBlaster, ByteBlaster II or ByteBlasterMV cable.

Figure 8–13. PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 8–13:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on `DATA0` and `DCLK` are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that `DATA0` and `DCLK` are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on `DATA0` and `DCLK` are not needed.

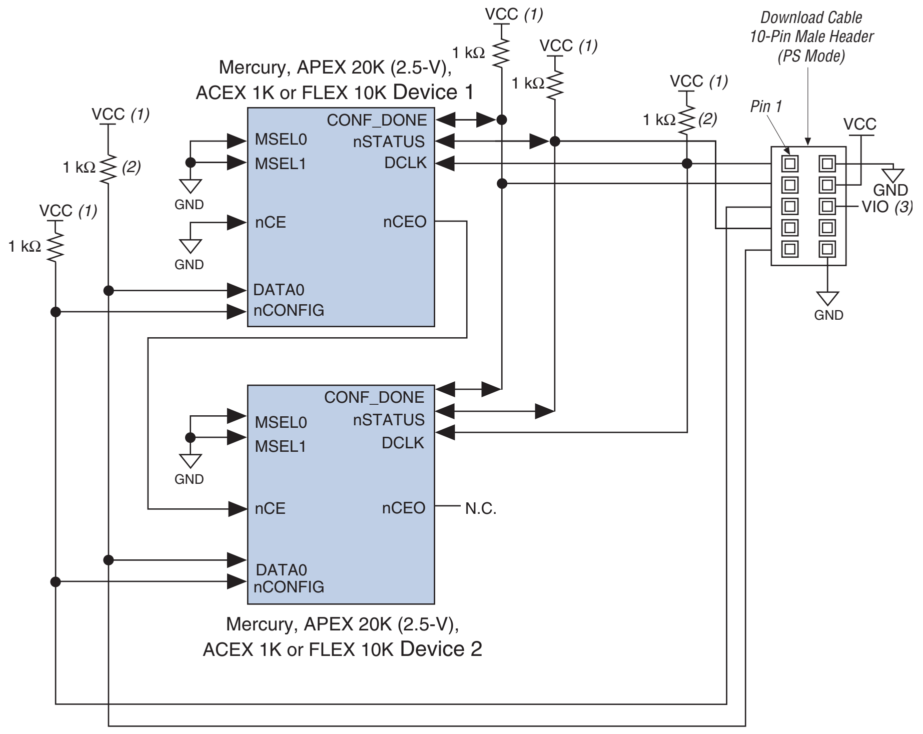
(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

You can use a download cable to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices by connecting each device's `nCEO` pin to the subsequent device's `nCE` pin. The first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to the `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. All other configuration pins, `nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE` are connected to every device in the chain. Because all `CONF_DONE` pins are tied together, all devices in the chain initialize and enter user mode at the same time.

In addition, because the `nSTATUS` pins are tied together, the entire chain halts configuration if any device detects an error. The *Auto-Restart Configuration After Error* option does not affect the configuration cycle because you must manually restart configuration in the Quartus II software when an error occurs.

Figure 8–14 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices with a download cable.

Figure 8–14. Multi-Device PS Configuration Using a USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV Cable



Notes to Figure 8–14:

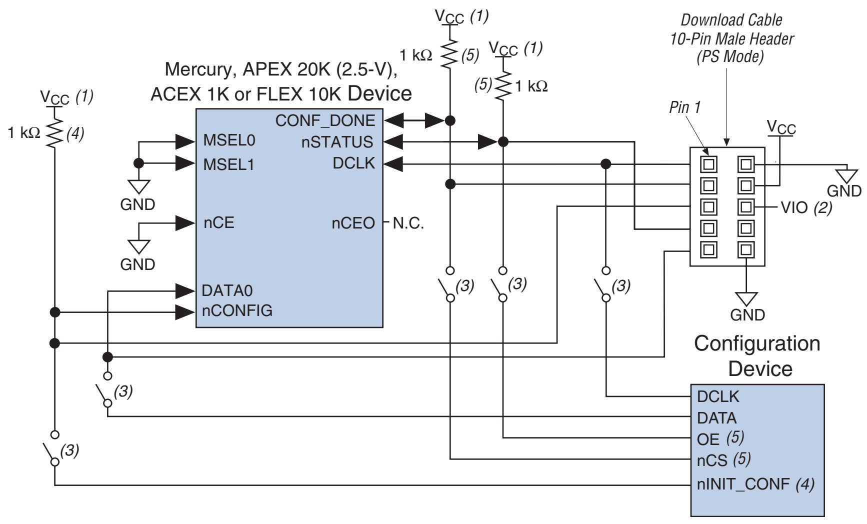
(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (VIO pin), ByteBlaster II or ByteBlasterMV cable.

(2) The pull-up resistors on DATA0 and DCLK are only needed if the download cable is the only configuration scheme used on your board. This is to ensure that DATA0 and DCLK are not left floating after configuration. For example, if you are also using a configuration device, the pull-up resistors on DATA0 and DCLK are not needed.

(3) Pin 6 of the header is a VIO reference voltage for the MasterBlaster output driver. VIO should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to nCE when it is used for Active Serial programming, otherwise it is a no connect.

If you are using a download cable to configure device(s) on a board that also has configuration devices, you should electrically isolate the configuration device from the target device(s) and cable. One way to isolate the configuration device is to add logic, such as a multiplexer, that can select between the configuration device and the cable. The multiplexer chip should allow bidirectional transfers on the `nSTATUS` and `CONF_DONE` signals. Another option is to add switches to the five common signals (`nCONFIG`, `nSTATUS`, `DCLK`, `DATA0`, and `CONF_DONE`) between the cable and the configuration device. The last option is to remove the configuration device from the board when configuring the FPGA with the cable. Figure 8–15 shows a combination of a configuration device and a download cable to configure an FPGA.

Figure 8–15. PS Configuration with a Download Cable & Configuration Device Circuit



Notes to Figure 8–15:

(1) The pull-up resistor should be connected to the same supply voltage as the configuration device.

(2) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the MasterBlaster Serial/USB Communications Cable Data Sheet for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(3) You should not attempt configuration with a download cable while a configuration device is connected to a Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K device. Instead, 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.

(4) The `nINIT_CONF` pin (available on enhanced configuration devices and EPC2 devices only) 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 (e.g., on EPC1 devices), `nCONFIG` must be pulled to V_{CC} either directly or through a resistor.

(5) The enhanced configuration devices' and EPC2 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-up resistors on configuration device* option when generating programming files.

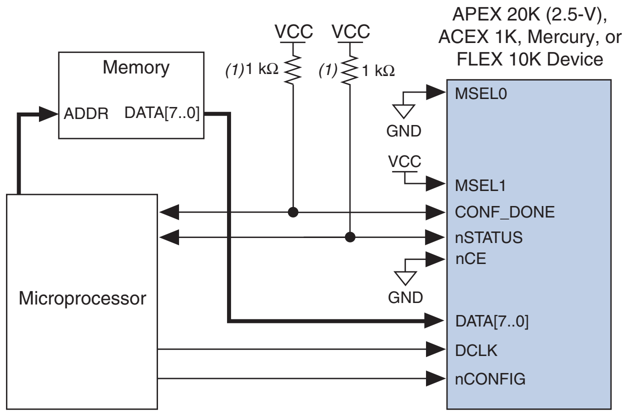
For more information on how to use the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV cables, refer to the following data sheets.

- 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

Passive Parallel Synchronous Configuration

Passive parallel synchronous (PPS) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the serializing clock to the FPGA. The target device latches the byte-wide data on the `DATA[7..0]` pins on the rising edge of `DCLK` and then uses the next eight falling edges on `DCLK` to serialize the data internally. On the ninth rising `DCLK` edge, the next byte of configuration data is latched into the target device. Figure 8–16 shows the configuration interface connections between the FPGA and a microprocessor for single device configuration.

Figure 8–16. Single Device PPS Configuration Using a Microprocessor



Note to Figure 8–16:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of 3 stages: reset, configuration, and initialization. While nCONFIG or nSTATUS are low, the device is in reset. To initiate configuration in this scheme, the microprocessor must generate a low-to-high transition on the nCONFIG pin.

☞ VCCINT and VCCIO pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When nCONFIG goes high, the device comes out of reset and releases the open-drain nSTATUS pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once nSTATUS is released the FPGA is ready to receive configuration data and the configuration stage begins. When nSTATUS is pulled high, the microprocessor should place the configuration data one byte at a time on the DATA[7..0] pins. New configuration data should be sent to the FPGA every eight DCLK cycles.

The Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K device receives configuration data on its DATA[7..0] pins and the clock is received on the DCLK pin. On the first rising DCLK edge, a byte of configuration data is latched into the target device; the subsequent eight falling DCLK edges serialize the configuration data in the device. On the ninth rising clock edge, the next byte of configuration data is latched and serialized into the target device.

Data is clocked into the target device until CONF_DONE goes high. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K devices, the initialization process is synchronous and can be clocked by its internal oscillator (typically 10 MHz) or by the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K device will take care to provide itself with enough clock cycles for proper initialization. Therefore, if the internal oscillator is the initialization clock source, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

In ACEX 1K and FLEX 10K devices, initialization can be clocked by the clock input on the DCLK pin or by the optional CLKUSR pin. By default, the clock on DCLK is the clock source for initialization. The configuration files created by the Quartus II and the MAX+PLUS II software incorporate the extra bits for proper device initialization. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device. Driving DCLK to the device after configuration is complete does not affect device operation.

You also have the flexibility to synchronize initialization of multiple devices by using the CLKUSR option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly. APEX 20K devices require 40 clock cycles, while ACEX 1K and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-ups and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure DCLK and DATA0 are not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. The DATA[7..1] pins are available as user I/O pins after configuration. When the PPS scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

The configuration clock (`DCLK`) speed must be below the specified frequency, as listed in Tables 8–12 through 8–14, to ensure correct configuration. No maximum `DCLK` period exists, which means you can pause configuration by halting `DCLK` for an indefinite amount of time. An optional status pin (`RDYnBSY`) on the FPGA indicates when it is busy serializing configuration data and when it is ready to accept the next data byte. The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

If an error occurs during configuration, the FPGA drives its `nSTATUS` pin low, resetting itself internally. The low signal on the `nSTATUS` pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration on Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases `nSTATUS` after a reset time-out period (maximum of 40 µs). After `nSTATUS` is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse `nCONFIG` low. If this option is turned off, the microprocessor must generate a low-to-high transition on `nCONFIG` to restart the configuration process.

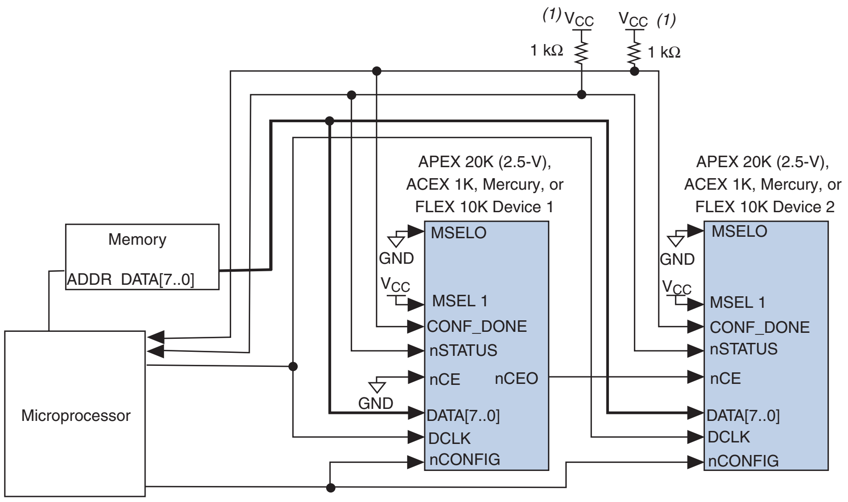
The microprocessor can also monitor the `CONF_DONE` and `INIT_DONE` pins to ensure successful configuration. The `CONF_DONE` pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but `CONF_DONE` or `INIT_DONE` have not gone high, the microprocessor must reconfigure the target device.

☞ If the optional `CLKUSR` pin is being used and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the `nCONFIG` pin low-to-high. The `nCONFIG` pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices and 2 µs for ACEX 1K and FLEX 10K devices. When `nCONFIG` is pulled low, the FPGA also pulls `nSTATUS` and `CONF_DONE` low and all I/O pins are tri-stated. Once `nCONFIG` returns to a logic high state and `nSTATUS` is released by the FPGA, reconfiguration begins.

Figure 8–17 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10K devices using a microprocessor. This circuit is similar to the PPS configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 8–17. Multi-Device PPS Configuration Using a Microprocessor



Note to Figure 8–17:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPS configuration the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Altera recommends keeping the configuration data valid on the `DATA[7..0]` bus for the 8 serializing clock cycles. The configuration data should be held valid on the `DATA` bus for the complete byte period because the `nCEO` of the first (and preceding) device can go low during the serializing `DCLK` cycles. Once the `nCEO` of the first (and preceding) device goes low, the second (and next) device becomes active and will begin trying to accept configuration data. If the configuration data is not valid on the first `DCLK` edge after `nCEO` goes low, then the second device will see incorrect configuration data and will never begin accepting configuration data. This situation will only arise if you are sharing the `DATA[7..0]` bus with other system data such that the configuration data is only valid for a portion of the byte period.

If your system requires to bus-share the DATA[7..0] line, you can work-around this by ensuring that the second (or next) device sees correct configuration data on the first rising edge of DCLK after the nCEO signal goes low. This can be achieved by delaying the nCEO signal by using external registers or by presenting the next byte of configuration data after the nCEO transition.

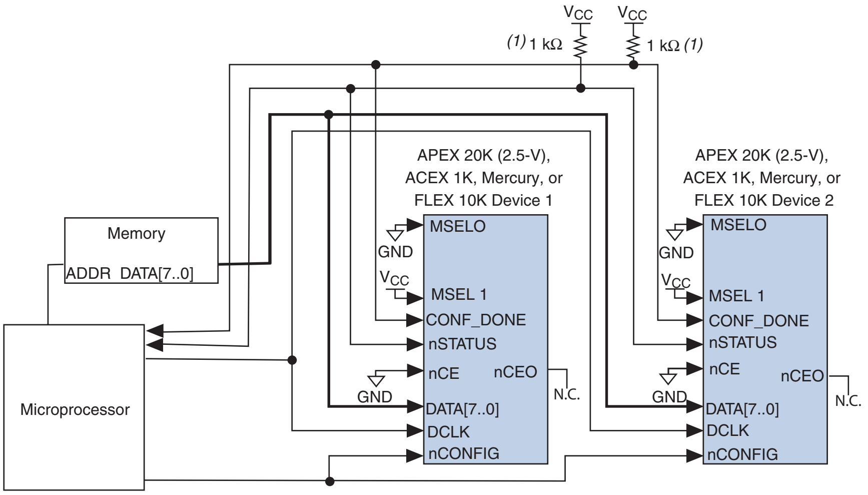
All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DCLK and DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration on Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device nCE inputs are tied to GND, while nCEO pins are left floating. All other configuration pins (nCONFIG, nSTATUS, DCLK, DATA[7..0], and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they 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. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–18 shows multi-device PPS configuration when both devices are receiving the same configuration data.

Figure 8–18. Multiple-Device PPS Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–18:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

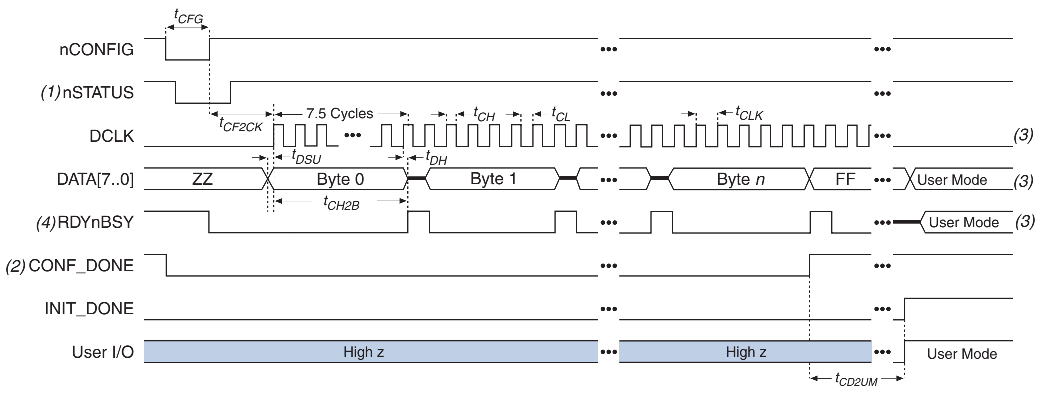
(2) The nCEO pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K (2.5 V), ACEX 1K or FLEX 10KE devices with other Altera devices that support PPS configuration, such as APEX 20KE devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, *see Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 8–19 shows the timing waveform for the PPS configuration scheme using a microprocessor.

Figure 8–19. Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10KE PPS Configuration Timing Waveform



Notes to Figure 8–19:

(1) Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE device holds nSTATUS low for approximately 5 µs after V_{CC} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, CONF_DONE is low.

(3) DATA0 and DCLK should not be left floating after configuration. It should be driven high or low, whichever is more convenient. DATA[7..1] and RDYnBSY are available as user I/O pins after configuration and the state of theses pins depends on the design programmed into the device.

(4) The RDYnBSY pin is not required in the PPS mode. Configuration data can be sent every 8 DCLK cycles without monitoring this status pin.

Tables 8–12 through 8–14 define the timing parameters for Mercury, APEX 20K, ACEX 1K, and FLEX 10K devices for PPS configuration.

Table 8–11. PPS Timing Parameters for Mercury Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	nCONFIG low to CONF_DONE low		200	ns
t_{CF2ST0}	nCONFIG low to nSTATUS low		200	ns
t_{CFG}	nCONFIG low pulse width	21		µs
t_{STATUS}	nSTATUS low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	nCONFIG high to nSTATUS high		1 *(1)*	µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	45		µs
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		µs
t_{DSU}	Data setup time before rising edge on DCLK	10		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH2B}	First rising DCLK to first rising RDYnBSY *(2)*	0.75 *(3)*		µs

Table 8–11. PPS Timing Parameters for Mercury Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CH}	`DCLK` high time	10		ns
t_{CL}	`DCLK` low time	10		ns
t_{CLK}	`DCLK` period	20		ns
f_{MAX}	`DCLK` frequency		50	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	6	28	µs

Notes to Table 8–11:

(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.
(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.
(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–12. PPS Timing Parameters for APEX 20K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	40		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` frequency		33.3	MHz

Table 8–12. PPS Timing Parameters for APEX 20K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	2	8	µs

Notes to Tables 8–12:

(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.

(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.

(4) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 to obtain this value.

Table 8–13. PPS Timing Parameters for ACEX 1K & FLEX 10KE Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs
t_{CH}	`DCLK` high time	15		ns
t_{CL}	`DCLK` low time	15		ns
t_{CLK}	`DCLK` period	30		ns
f_{MAX}	`DCLK` frequency		33.3	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	0.6	2	µs

Notes to Table 8–13:

(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.

(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.

(4) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Table 8–14. PPS Timing Parameters for FLEX 10K Devices

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{ST2CK}	`nSTATUS` high to first rising edge on `DCLK`	1		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	10		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH2B}	First rising `DCLK` to first rising `RDYnBSY` *(2)*	0.75 *(3)*		µs
t_{CH}	`DCLK` high time	30		ns
t_{CL}	`DCLK` low time	30		ns
t_{CLK}	`DCLK` period	60		ns
f_{MAX}	`DCLK` frequency		16.7	MHz
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	0.6	2	µs

Notes to Table 8–14:

(1) This value is obtainable if users do not delay configuration by extending the `nSTATUS` low pulse width.

(2) The `RDYnBSY` pin is not required in the PPS mode. Configuration data can be sent every 8 `DCLK` cycles without monitoring this status pin.

(3) This parameter depends on the `DCLK` frequency. The `RDYnBSY` signal goes high 7.5 clock cycles after the rising edge of `DCLK`. This value was calculated with a `DCLK` frequency of 10 MHz.

(4) The minimum and maximum numbers apply only if `DCLK` is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

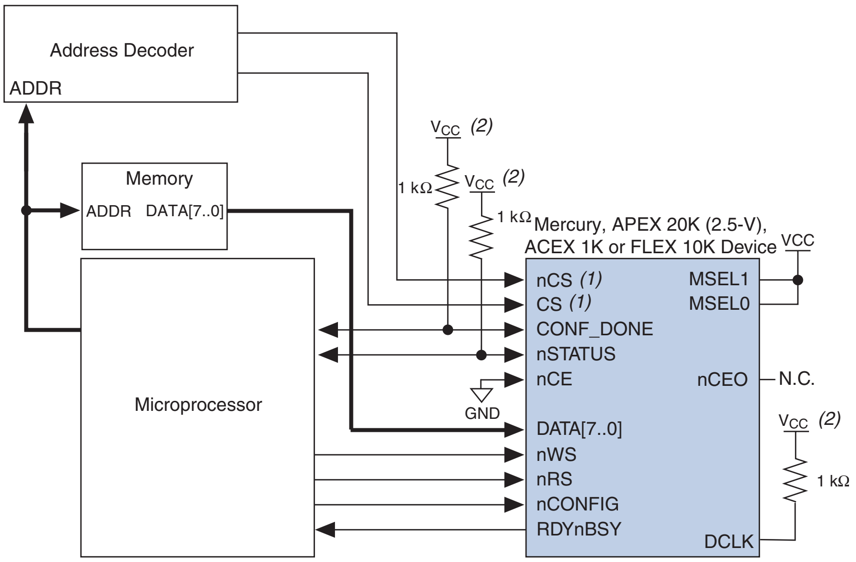
Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, in Volume 2* of the Configuration Handbook.

Passive Parallel Asynchronous Configuration

Passive Parallel Asynchronous (PPA) configuration uses an intelligent host, such as a microprocessor, to transfer configuration data from a storage device, such as flash memory, to the target Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices. Configuration data can be stored in TTF, RBF, or HEX format. The host system outputs byte-wide data and the accompanying strobe signals to the FPGA. When using PPA, you should pull the `DCLK` pin high through a 1-kΩ pull-up resistor to prevent unused configuration input pins from floating.

Figure 8–20 shows the configuration interface connections between the FPGA and a microprocessor for single device PPA configuration. The microprocessor or an optional address decoder can control the device's chip select pins, `nCS` and `CS`. The address decoder allows the microprocessor to select the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device by accessing a particular address, which simplifies the configuration process. The `nCS` and `CS` pins must be held active during configuration and initialization.

Figure 8–20. Single Device PPA Configuration Using a Microprocessor *Note (1)*



Notes to Figure 8–20:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the `nCS` pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for the device.

During PPA configuration, it is only required to use either the `nCS` or `CS` pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, `nCS` can be tied to ground while `CS` is toggled to control configuration. The device's `nCS` or `CS` pins can be toggled during PPA configuration if the design meets the specifications set for t_{CSSU}, t_{WSP}, and t_{CSH} listed in Tables 8–15 through 8–17.

Upon power-up, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device goes through a Power-On Reset (POR) for approximately 5 µs. During POR, the device resets and holds `nSTATUS` low, and tri-states all user I/O pins. Once the FPGA successfully exits POR, all user I/O pins are tri-stated. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10KE devices have weak pull-up resistors on the user I/O pins which are on before and during configuration.

The value of the weak pull-up resistors on the I/O pins that are on before and during configuration can be found in the Operating Conditions table of the appropriate device family data sheet.

The configuration cycle consists of 3 stages: reset, configuration, and initialization. While `nCONFIG` or `nSTATUS` are low, the device is in reset. To initiate configuration, the microprocessor must generate a low-to-high transition on the `nCONFIG` pin.

`VCCINT` and `VCCIO` pins on the banks where the configuration and JTAG pins reside need to be fully powered to the appropriate voltage levels in order to begin the configuration process.

When `nCONFIG` goes high, the device comes out of reset and releases the open-drain `nSTATUS` pin, which is then pulled high by an external 1-kΩ pull-up resistor. Once `nSTATUS` is released the FPGA is ready to receive configuration data and the configuration stage begins. When `nSTATUS` is pulled high, the microprocessor should then assert the target device's `nCS` pin low and/or `CS` pin high. Next, the microprocessor places an 8-bit configuration word (one byte) on the target device's `DATA[7..0]` pins and pulses the `nWS` pin low.

On the rising edge of `nWS`, the target device latches in a byte of configuration data and drives its `RDYnBSY` signal low, which indicates it is processing the byte of configuration data. The microprocessor can then perform other system functions while the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device is processing the byte of configuration data.

During the time `RDYnBSY` is low, the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device internally processes the configuration data using its internal oscillator (typically 10 MHz). When the device is ready for the next byte of configuration data, it will drive `RDYnBSY` high. If the microprocessor senses a high signal when it polls `RDYnBSY`, the microprocessor sends the next byte of configuration data to the FPGA.

Alternatively, the `nRS` signal can be strobed low, causing the `RDYnBSY` signal to appear on `DATA7`. Because `RDYnBSY` does not need to be monitored, this pin doesn't need to be connected to the microprocessor. Data should not be driven onto the data bus while `nRS` is low because it will cause contention on the `DATA7` pin. If the `nRS` pin is not used to monitor configuration, it should be tied high.

To simplify configuration and save an I/O port, the microprocessor can wait for the total time of t_{BUSY}(max) + t_{RDY2WS} + t_{W2SB} before sending the next data byte. In this set-up, nRS should be tied high and RDYnBSY does not need to be connected to the microprocessor. The t_{BUSY}, t_{RDY2WS} and t_{W2SB} timing specifications are listed in Tables 8–15 through 8–17.

Next, the microprocessor checks nSTATUS and CONF_DONE. If nSTATUS is not low and CONF_DONE is not high, the microprocessor sends the next data byte. However, if nSTATUS is not low and all the configuration data has been received, the device is ready for initialization. After the FPGA has received all configuration data successfully, it releases the open-drain CONF_DONE pin, which is pulled high by an external 1-kΩ pull-up resistor. A low-to-high transition on CONF_DONE indicates configuration is complete and initialization of the device can begin.

In Mercury and APEX 20K (2.5 V) devices, the initialization clock source is either the FPGA's internal oscillator (typically 10 MHz) or the optional CLKUSR pin. By default, the internal oscillator is the clock source for initialization. If the internal oscillator is used, the Mercury or APEX 20K (2.5 V) device allows enough clock cycles for proper initialization. Therefore, sending the entire configuration file to the device is sufficient to configure and initialize the device.

In ACEX 1K and FLEX 10K devices, the initialization clock source is either an external host (e.g. a configuration device or microprocessor) or the optional CLKUSR pin. By default, in PPA, the device uses its internal oscillator (typically 10MHz) to clock the initialization cycle. The ACEX 1K or FLEX 10K device will take care to provide itself with enough clock cycles for proper initialization.

You also have the flexibility to synchronize initialization of multiple devices by using the *CLKUSR* option. The *Enable user-supplied start-up clock (CLKUSR)* option can be turned on in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. Supplying a clock on CLKUSR will not affect the configuration process. After all configuration data has been accepted and CONF_DONE goes high, Mercury devices require 136 clock cycles to initialize properly, APEX 20K (2.5 V) devices require 40 clock cycles, ACEX and FLEX 10K devices require 10 clock cycles.

An optional INIT_DONE pin is available, which signals the end of initialization and the start of user-mode with a low-to-high transition. This *Enable INIT_DONE output* option is available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box. If the INIT_DONE pin is used it will be high due to an external 1-kΩ pull-up when nCONFIG is low and during the beginning of configuration. 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. The microprocessor must be able to detect this low-to-high transition which signals the FPGA has entered user mode. In user-mode, the user I/O pins will no longer have weak pull-up resistors and will function as assigned in your design. When initialization is complete, the FPGA enters user mode.

To ensure `DATA0` is not left floating at the end of configuration, the microprocessor must take care to drive them either high or low, whichever is convenient on your board. After configuration, the `nCS`, `CS`, `nRS`, `nWS`, `RDYnBSY`, and `DATA[7..1]` pins can be used as user I/O pins. When the PPA scheme is chosen in the Quartus II software, as a default these I/O pins are tri-stated in user mode and should be driven by the microprocessor. To change this default option in the Quartus II software, select the **Dual-Purpose Pins** tab of the **Device & Pin Options** dialog box.

If an error occurs during configuration, the FPGA drives its `nSTATUS` pin low, resetting itself internally. The low signal on the `nSTATUS` pin also alerts the microprocessor that there is an error. If the *Auto-Restart Configuration After Error* option-available in the Quartus II software from the **General** tab of the **Device & Pin Options** dialog box-is turned on, the FPGA releases `nSTATUS` after a reset time-out period (maximum of 40 µs). After `nSTATUS` is released and pulled high by a pull-up resistor, the microprocessor can try to reconfigure the target device without needing to pulse `nCONFIG` low. If this option is turned off, the microprocessor must generate a low-to-high transition on `nCONFIG` to restart the configuration process.

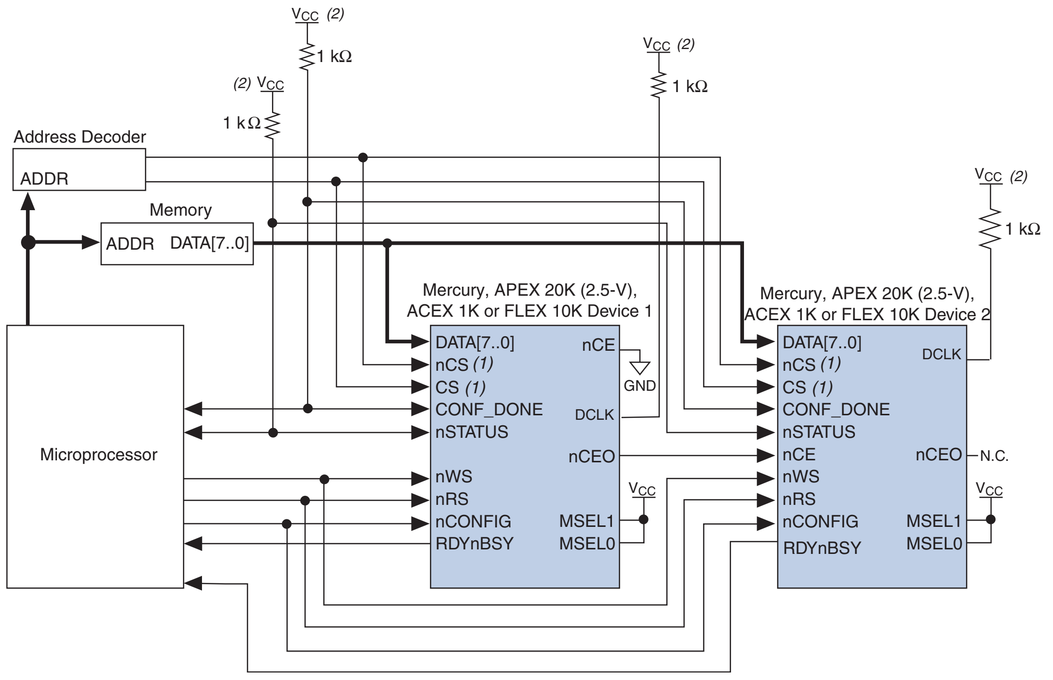
The microprocessor can also monitor the `CONF_DONE` and `INIT_DONE` pins to ensure successful configuration. The `CONF_DONE` pin must be monitored by the microprocessor to detect errors and determine when programming completes. If the microprocessor sends all configuration data but `CONF_DONE` or `INIT_DONE` has not gone high, the microprocessor must reconfigure the target device.

☞ If the optional `CLKUSR` pin is being used and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 µs).

When the FPGA is in user-mode, a reconfiguration can be initiated by transitioning the `nCONFIG` pin low-to-high. The **`nCONFIG`** pin should be low for at least 21 µs for Mercury devices, 8 µs for APEX 20K devices, and 2 µs for ACEX 1K and FLEX 10K devices. When `nCONFIG` is pulled low, the FPGA also pulls nSTATUS and CONF_DONE low and all I/O pins are tri-stated. Once nCONFIG returns to a logic high state and nSTATUS is released by the FPGA, reconfiguration begins.

Figure 8–21 shows how to configure multiple Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices using a microprocessor. This circuit is similar to the PPA configuration circuit for a single device, except the devices are cascaded for multi-device configuration.

Figure 8–21. Multi-Device PPA Configuration Using a Microprocessor



Notes to Figure 8–21:

(1) If not used, the CS pin can be connected to V_{CC} directly. If not used, the nCS pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

In multi-device PPA configuration the first device's nCE pin is connected to GND while its nCEO pin is connected to nCE of the next device in the chain. The last device's nCE input comes from the previous device, while its nCEO pin is left floating. After the first device completes configuration in a multi-device configuration chain, its nCEO pin drives low to activate the second device's nCE pin, which prompts the second device to begin configuration. The second device in the chain begins configuration within one clock cycle; therefore, the transfer of data destinations is transparent to the microprocessor.

Each device's RDYnBSY pin can have a separate input to the microprocessor. Alternatively, if the microprocessor is pin limited, all the RDYnBSY pins can feed an AND gate and the output of the AND gate can feed the microprocessor. For example, if you have 2 devices in a PPA configuration chain, the second device's RDYnBSY pin will be high during the time that the first device is being configured. When the first device has been successfully configured, it will driven nCEO low to activate the next device in the chain and drive its RDYnBSY pin high. Therefore, since RDYnBSY signal is driven high before configuration and after configuration before entering user-mode, the device being configured will govern the output of the AND gate.

The nRS signal can be used in multi-device PPA chain since the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device will tri-state its DATA[7..0] pins before configuration and after configuration before entering user-mode to avoid contention. Therefore, only the device that is currently being configured will respond to the nRS strobe by asserting DATA7.

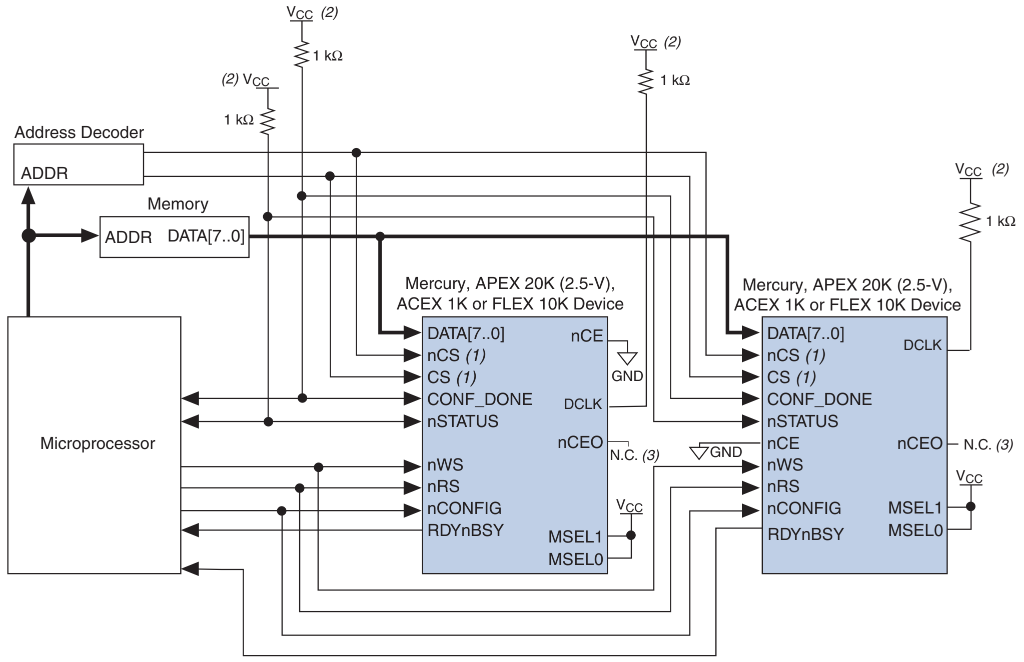
All other configuration pins (nCONFIG, nSTATUS, DATA[7..0], nCS, CS, nWS, nRS and CONF_DONE) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the DATA lines are buffered for every fourth device. Because all device CONF_DONE pins are tied together, all devices initialize and enter user mode at the same time.

Since all nSTATUS and CONF_DONE pins are tied together, if any device detects an error, configuration stops for the entire chain and the entire chain must be reconfigured. For example, if the first FPGA flags an error on nSTATUS, it resets the chain by pulling its nSTATUS pin low. This behavior is similar to a single FPGA detecting an error.

If the *Auto-Restart Configuration After Error* option is turned on, the FPGAs release their nSTATUS pins after a reset time-out period (maximum of 40 µs). After all nSTATUS pins are released and pulled high, the microprocessor can try to reconfigure the chain without needing to pulse nCONFIG low. If this option is turned off, the microprocessor must generate a low-to-high transition on nCONFIG to restart the configuration process.

In your system, you may have multiple devices that contain the same configuration data. To support this configuration scheme, all device `nCE` inputs are tied to GND, while `nCEO` pins are left floating. All other configuration pins (`nCONFIG`, `nSTATUS`, `DATA[7..1]`, `nCS`, `CS`, `nWS`, `nRS` and `CONF_DONE`) are connected to every device in the chain. You should pay special attention to the configuration signals because they may require buffering to ensure signal integrity and prevent clock skew problems. Specifically, ensure that the `DATA` lines are buffered for every fourth device. Devices must be the same density and package. All devices will start and complete configuration at the same time. Figure 8–22 shows multi-device PPA configuration when both devices are receiving the same configuration data.

Figure 8–22. Multiple-Device PPA Configuration Using a Microprocessor When Both FPGAs Receive the Same Data



Notes to Figure 8–22:

(1) If not used, the `CS` pin can be connected to V_{CC} directly. If not used, the `nCS` pin can be connected to GND directly.

(2) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

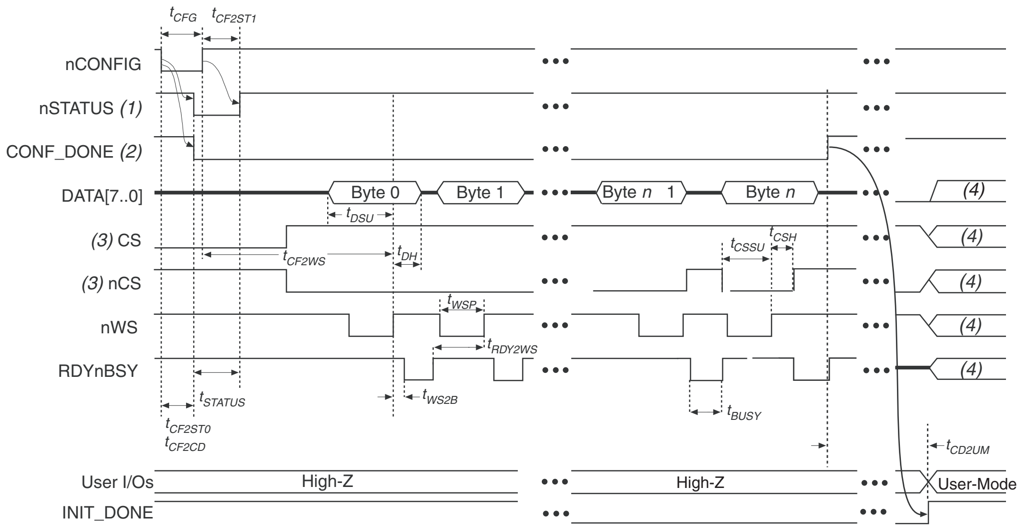
(3) The `nCEO` pins of both devices are left unconnected when configuring the same configuration data into multiple devices.

You can use a single configuration chain to configure Mercury, APEX 20K, ACEX, and FLEX 10KE devices with other Altera devices that support PPA configuration, such as Stratix® or APEX II devices. To ensure that all devices in the chain complete configuration at the same time or that an error flagged by one device initiates reconfiguration in all devices, all of the device `CONF_DONE` and `nSTATUS` pins must be tied together.

For more information on configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

Figure 8–23 shows the timing waveform for the PPA configuration scheme using a microprocessor.

Figure 8–23. PPA Configuration Timing Waveform



Notes to Figure 8–23:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

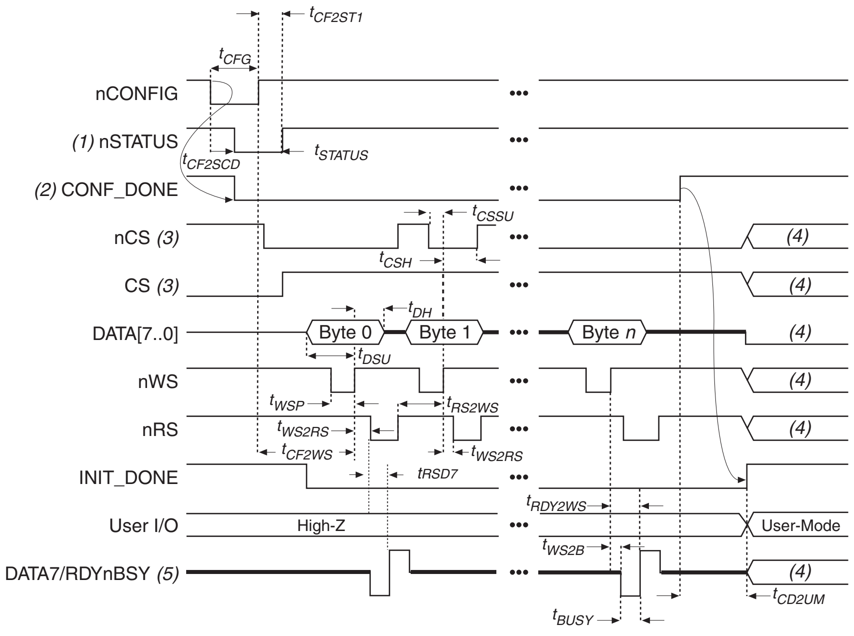
(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS` and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

Figure 8–24 shows the timing waveform for the PPA configuration scheme when using a strobed `nRS` and `nWS` signal.

Figure 8–24. PPA Configuration Timing Waveform Using nRS & nWS



Notes to Figure 8–24:

(1) Upon power-up, the Mercury, APEX 20K, ACEX 1K, or FLEX 10K device holds `nSTATUS` low for not more than 5 µs after V_{CC} reaches its minimum requirement.

(2) Upon power-up, before and during configuration, `CONF_DONE` is low.

(3) The user can toggle `nCS` or `CS` during configuration if the design meets the specification for t_{CSSU}, t_{WSP}, and t_{CSH}.

(4) `DATA0` should not be left floating after configuration. It should be driven high or low, whichever is more convenient. `DATA[7..1]`, `CS`, `nCS`, `nWS`, `nRS`, and `RDYnBSY` are available as user I/O pins after configuration and the state of theses pins depends on the dual-purpose pin settings.

(5) `DATA7` is a bidirectional pin. It is an input for configuration data input, but it is an output to show the status of `RDYnBSY`.

Tables 8–15 through 8–17 define the timing parameters for Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices for PPA configuration.

Table 8–15. PPA Timing Parameters for Mercury Devices (Part 1 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	21		µs

Table 8–15. PPA Timing Parameters for Mercury Devices (Part 2 of 2) Note (1)

Symbol	Parameter	Min	Max	Units
t_{STATUS}	`nSTATUS` low pulse width	10	40 (2)	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 (2)	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.4	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode (3)	6	28	µs

Notes to Table 8–15:
(1) This information is preliminary.
(2) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(3) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 136 to obtain this value.

Table 8–16. PPA Timing Parameters for APEX 20K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	8		µs
t_{STATUS}	`nSTATUS` low pulse width	10	40 (1)	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		1 (1)	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	10		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	0		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	40		µs
t_{DSU}	Data setup time before rising edge on `nWS`	10		ns

Table 8–16. PPA Timing Parameters for APEX 20K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.1	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(2)*	2	8	µs

Notes to Table 8–16:

(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.

(2) The minimum and maximum numbers apply only if the internal oscillator is chosen as the clock source for starting up the device. If the clock source is `CLKUSR`, multiply the clock period by 40 to obtain this value.

Table 8–17. PPA Timing Parameters for ACEX 1K & FLEX 10K Devices (Part 1 of 2)

Symbol	Parameter	Min	Max	Units
t_{CF2CD}	`nCONFIG` low to `CONF_DONE` low		200	ns
t_{CF2ST0}	`nCONFIG` low to `nSTATUS` low		200	ns
t_{CFG}	`nCONFIG` low pulse width	2		µs
t_{STATUS}	`nSTATUS` low pulse width	1	10 *(1)*	µs
t_{CF2ST1}	`nCONFIG` high to `nSTATUS` high		4 *(1)*	µs
t_{CSSU}	Chip select setup time before rising edge on `nWS`	20		ns
t_{CSH}	Chip select hold time after rising edge on `nWS`	10 *(2)* 15 *(3)*		ns
t_{CF2WS}	`nCONFIG` high to first rising edge on `nWS`	5		µs
t_{DSU}	Data setup time before rising edge on `nWS`	20		ns
t_{DH}	Data hold time after rising edge on `nWS`	0		ns
t_{WSP}	`nWS` low pulse width	200		ns
t_{WS2B}	`nWS` rising edge to `RDYnBSY` low		50	ns
t_{BUSY}	`RDYnBSY` low pulse width	0.4	1.6	µs
t_{RDY2WS}	`RDYnBSY` rising edge to `nWS` rising edge	50		ns
t_{WS2RS}	`nWS` rising edge to `nRS` falling edge	200		ns

Table 8–17. PPA Timing Parameters for ACEX 1K & FLEX 10K Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
t_{RS2WS}	`nRS` rising edge to `nWS` rising edge	200		ns
t_{RSD7}	`nRS` falling edge to `DATA7` valid with `RDYnBSY` signal		50	ns
t_{CD2UM}	`CONF_DONE` high to user mode *(4)*	0.6	2	µs

Notes to Table 8–17:
(1) This value is obtainable if users do not delay configuration by extending the `nCONFIG` or `nSTATUS` low pulse width.
(2) This parameter value applies to EPF10K10, EPF10K20, EPF10K40, EPF10K50, all FLEX 10KA, and FLEX 10KE devices.
(3) This parameter value applies to only EPF10K70 and EPF10K100 devices.
(4) If the clock source is `CLKUSR`, multiply the clock period by 10 to obtain this value.

Device configuration options and how to create configuration files are discussed further in *Section II, Software Settings, in Volume 2* of the Configuration Handbook.

JTAG Configuration

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The JTAG circuitry can also be used to shift configuration data into the device.

For more information on JTAG boundary-scan testing, see *Application Note 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

A device operating in JTAG mode uses four required pins, `TDI`, `TDO`, `TMS`, and `TCK`, and one optional pin, `TRST`. All user I/O pins are tri-stated during JTAG configuration. Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices are designed such that JTAG instructions have precedence over any device configuration modes. This means that JTAG configuration can take place without waiting for other configuration modes to complete. For example, if you attempt JTAG configuration of Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K FPGAs during PS configuration, PS configuration will be terminated and JTAG configuration will begin.

Table 8–18 explains each JTAG pin's function.

Table 8–18. JTAG Pin Descriptions

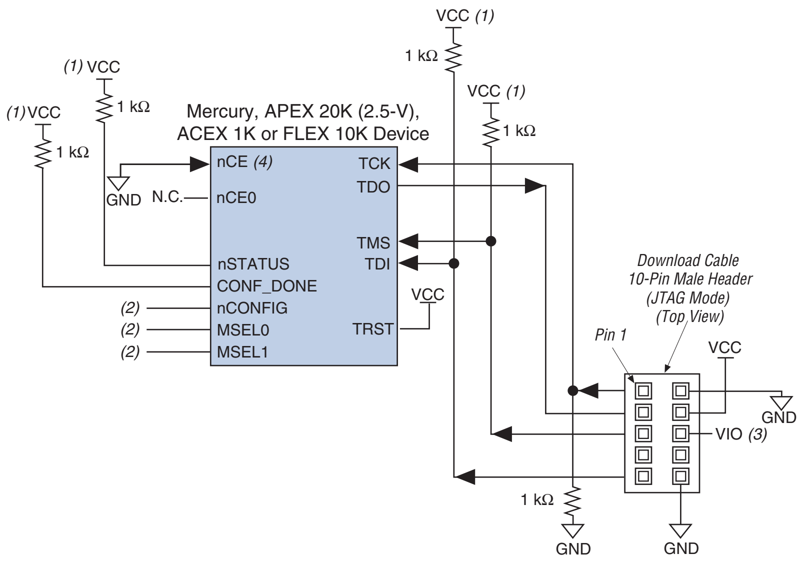
Pin	Description	Function
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of `TCK`. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of `TCK`. Therefore, `TMS` must be set up before the rising edge of `TCK`. `TMS` is evaluated on the rising edge of `TCK`. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST *(1)*	Test reset input (optional)	Active-low input to asynchronously reset the boundary-scan circuit. The `TRST` pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

Note to Table 8–18:
(1) FLEX 10K devices in 144-pin thin quad flat pack (TQFP) packages do not have a `TRST` pin. Therefore, the `TRST` pin can be ignored when using these devices.

If V_{CCIO} of the bank where the JTAG pins reside, are tied to 3.3-V, both the I/O pins and JTAG `TDO` port will drive at 3.3-V levels.

During JTAG configuration, data can be downloaded to the device on the PCB through the USB Blaster, MasterBlaster, ByteBlaster II, or ByteBlasterMV header. Configuring devices through a cable is similar to programming devices in-system, except the `TRST` pin should be connected to V_{CC}. This ensures that the TAP controller is not reset. Figure 8–25. shows JTAG configuration of a single Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device.

Figure 8–25. JTAG Configuration of a Single Device Using a Download Cable



Notes to Figure 8–25:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II, or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

To configure a single device in a JTAG chain, the programming software places all other devices in BYPASS mode. In BYPASS mode, devices pass programming data from the `TDI` pin to the `TDO` pin through a single bypass register without being affected internally. This scheme enables the programming software to program or verify the target device. Configuration data driven into the device appears on the `TDO` pin one clock cycle later.

Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices have dedicated JTAG pins that always function as JTAG pins. JTAG testing can be performed on Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices both before and after configuration, but not during configuration. The chip-wide reset (`DEV_CLRn`) and chip-wide output enable (`DEV_OE`) pins on Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices do not affect JTAG boundary-scan or programming operations. Toggling these pins does not affect JTAG operations (other than the usual boundary-scan operation).

When designing a board for JTAG configuration of Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices, the dedicated configuration pins should be considered. Table 8–19 shows how these pins should be connected during JTAG configuration.

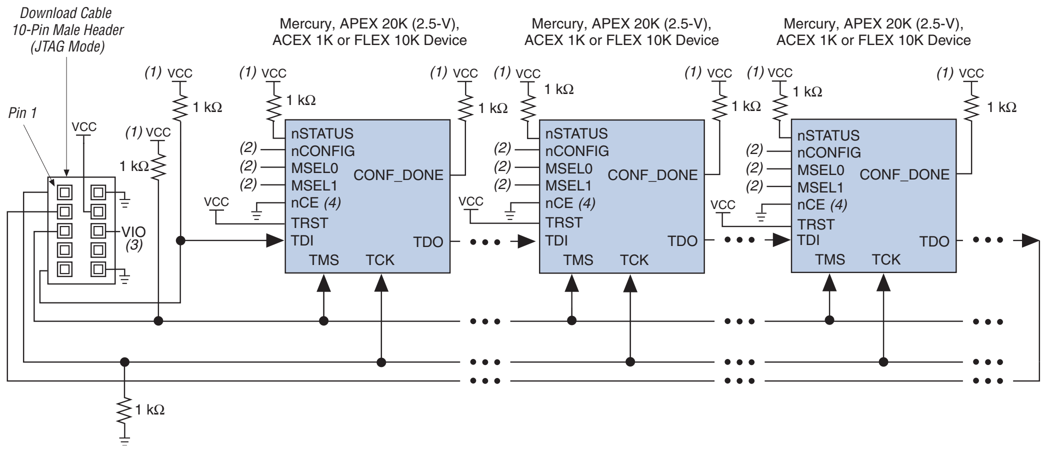
Table 8–19. Dedicated Configuration Pin Connections During JTAG Configuration

Signal	Description
nCE	On all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices in the chain, `nCE` should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, PPS or PPA configuration chains, the `nCE` pins should be connected to GND during JTAG configuration or JTAG configured in the same order as the configuration chain.
nCEO	On all Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K devices in the chain, `nCEO` can be left floating or connected to the `nCE` of the next device. See `nCE` description above.
MSEL	These pins must not be left floating. These pins support whichever non-JTAG configuration is used in production. If only JTAG configuration is used, you should tie both pins to ground.
nCONFIG	Driven high by connecting to V_{CC}, pulling high via a resistor, or driven by some control circuitry.
nSTATUS	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `nSTATUS` pin should be pulled up to V_{CC} individually. `nSTATUS` pulling low in the middle of JTAG configuration indicates that an error has occurred.
CONF_DONE	Pull to V_{CC} via a 1-kΩ resistor. When configuring multiple devices in the same JTAG chain, each `CONF_DONE` pin should be pulled up to V_{CC} individually. `CONF_DONE` going high at the end of JTAG configuration indicates successful configuration.
DCLK	Should not be left floating. Drive low or high, whichever is more convenient on your board.
DATA0	Should not be left floating. Drive low or high, whichever is more convenient on your board.

When programming a JTAG device chain, one JTAG-compatible header is connected to several devices. The number of devices in the JTAG chain is limited only by the drive capability of the download cable. When four or more devices are connected in a JTAG chain, Altera recommends buffering the `TCK`, `TDI`, and `TMS` pins with an on-board buffer.

JTAG-chain device programming is ideal when the system contains multiple devices, or when testing your system using JTAG BST circuitry. Figure 8–26 shows multi-device JTAG configuration.

Figure 8–26. JTAG Configuration of Multiple Devices Using a Download Cable



Notes to Figure 8–26:

(1) The pull-up resistor should be connected to the same supply voltage as the USB Blaster, MasterBlaster (`VIO` pin), ByteBlaster II or ByteBlasterMV cable.

(2) The `nCONFIG`, `MSEL0`, and `MSEL1` pins should be connected to support a non-JTAG configuration scheme. If only JTAG configuration is used, connect `nCONFIG` to V_{CC}, and `MSEL0` and `MSEL1` to ground.

(3) Pin 6 of the header is a `VIO` reference voltage for the MasterBlaster output driver. `VIO` should match the device's V_{CCIO}. Refer to the *MasterBlaster Serial/USB Communications Cable Data Sheet* for this value. In the ByteBlasterMV, this pin is a no connect. In the USB Blaster and ByteBlaster II, this pin is connected to `nCE` when it is used for Active Serial programming, otherwise it is a no connect.

(4) `nCE` must be connected to GND or driven low for successful JTAG configuration.

The `nCE` pin must be connected to GND or driven low during JTAG configuration. In multi-device PS, PPS, and PPA configuration chains, the first device's `nCE` pin is connected to GND while its `nCEO` pin is connected to `nCE` of the next device in the chain. The last device's `nCE` input comes from the previous device, while its `nCEO` pin is left floating. After the first device completes configuration in a multi-device configuration chain, its `nCEO` pin drives low to activate the second device's `nCE` pin, which prompts the second device to begin configuration. Therefore, if these devices are also in a JTAG chain, you should make sure the `nCE` pins are connected to GND during JTAG configuration or that the devices are JTAG configured in the same order as the configuration chain. As long as the devices are JTAG configured in the same order as the multi-device configuration chain, the `nCEO` of the previous device will drive `nCE` of the next device low when it has successfully been JTAG configured.

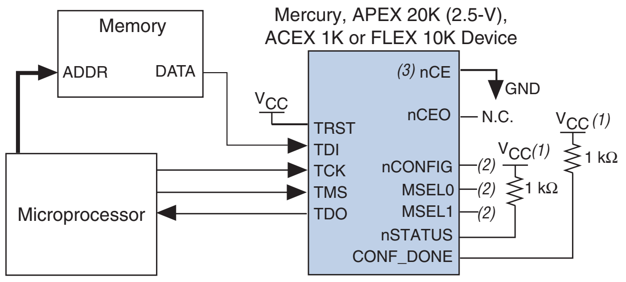
Other Altera devices that have JTAG support can be placed in the same JTAG chain for device programming and configuration.

For more information about configuring multiple Altera devices in the same configuration chain, see *Configuring Mixed Altera FPGA Chains* in the Configuration Handbook.

The Quartus II software verifies successful JTAG configuration upon completion. At the end of configuration, the software checks the state of `CONF_DONE` through the JTAG port. If `CONF_DONE` is not high, the Quartus II software indicates that configuration has failed. If `CONF_DONE` is high, the software indicates that configuration was successful. When Quartus II generates a JAM file for a multi-device chain, it contains instructions so that all the devices in the chain will be initialized at the same time.

Figure 8–27 shows JTAG configuration of a Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K FPGA with a microprocessor.

Figure 8–27. JTAG Configuration of a Single Device Using a Microprocessor



Notes to Figure 8–27:

(1) The pull-up resistor should be connected to a supply that provides an acceptable input signal for all devices in the chain.

(2) Connect the `nCONFIG`, `MSEL1`, and `MSEL0` pins to support a non-JTAG configuration scheme. If your design only uses JTAG configuration, connect the `nCONFIG` pin to V_{CC} and the `MSEL1` and `MSEL0` pins to ground.

(3) `nCE` must be connected to GND or driven low for successful JTAG configuration.

Jam STAPL

Jam STAPL, JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam STAPL is a freely licensed open standard.

The Jam Player provides an interface for manipulating the IEEE Std. 1149.1 JTAG TAP state machine.

For more information on JTAG and Jam STAPL in embedded environments, see *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*. To download the jam player, visit the Altera web site at:

www.altera.com/support/software/download/programming/jam/jam-index.jsp

Configuring Mercury, APEX 20K (2.5 V), ACEX 1K & FLEX 10K FPGAs with JRunner

JRunner is a software driver that allows you to configure Altera FPGAs, including Mercury, APEX 20K (2.5 V), ACEX 1K, and FLEX 10K FPGAs, through the ByteBlaster II or ByteBlasterMV cables in JTAG mode. The programming input file supported is in RBF format. JRunner also requires a Chain Description File (**.cdf**) generated by the Quartus II software. JRunner is targeted for embedded JTAG configuration. The source code has been developed for the Windows NT operating system (OS). You can customize the code to make it run on other platforms.

For more information on the JRunner software driver, see the *JRunner Software Driver: An Embedded Solution to the JTAG Configuration White Paper* and the source files.

Device Configuration Pins

The following tables describe the connections and functionality of all the configuration related pins on the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device. Table 8–20 describes the dedicated configuration pins, which are required to be connected properly on your board for successful configuration. Some of these pins may not be required for your configuration schemes.

Table 8–20. Dedicated Configuration Pins (Part 1 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`MSEL0` `MSEL1`	N/A	All	Input	Two-bit configuration input that sets the Mercury, APEX 20K (2.5 V), ACEX 1K, or FLEX 10K device configuration scheme. See Table 8–6 for the appropriate connections. These pins must remain at a valid state during power-up, before `nCONFIG` is pulled low to initiate a reconfiguration and during configuration.
`VCCSEL`	N/A	All	Input	This pin is only available in Mercury devices. Dedicated input that ensures the configuration related I/O banks have powered up to the appropriate 1.8-V or 2.5-V/3.3-V voltage levels before starting configuration. A logic high (1.5 V, 1.8 V, 2.5 V, 3.3 V) means 2.5 V/3.3 V, and a logic low means 1.8 V.
`nCONFIG`	N/A	All	Input	Configuration control input. Pulling this pin low during user-mode will cause the FPGA to lose its configuration data, enter a reset state, tri-state all I/O pins, and returning this pin to a logic high level will initiate a reconfiguration. If your configuration scheme uses an enhanced configuration device or EPC2 device, `nCONFIG` can be tied directly to V_{CC} or to the configuration device's `nINIT_CONF` pin.

Table 8–20. Dedicated Configuration Pins (Part 2 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
`nSTATUS`	N/A	All	Bidirectional open-drain	The FPGA drives `nSTATUS` low immediately after power-up and releases it within 5 µs. (When using a configuration device, the configuration device holds `nSTATUS` low for up to 200 ms.) Status output. If an error occurs during configuration, `nSTATUS` is pulled low by the target device. Status input. If an external source drives the `nSTATUS` pin low during configuration or initialization, the target device enters an error state. Driving `nSTATUS` low after configuration and initialization does not affect the configured device. If a configuration device is used, driving `nSTATUS` low will cause the configuration device to attempt to configure the FPGA, but since the FPGA ignores transitions on `nSTATUS` in user-mode, the FPGA will not reconfigure. To initiate a reconfiguration, `nCONFIG` must be pulled low. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.
`CONF_DONE`	N/A	All	Bidirectional open-drain	Status output. The target FPGA drives the `CONF_DONE` pin low before and during configuration. Once all configuration data is received without error and the initialization cycle starts, the target device releases `CONF_DONE`. Status input. After all data is received and `CONF_DONE` goes high, the target device initializes and enters user mode. Driving `CONF_DONE` low after configuration and initialization does not affect the configured device. The enhanced configuration devices' and EPC2 devices' `OE` and `nCS` pins have optional internal programmable pull-up resistors. If internal pull-up resistors are used, external 1-kΩ pull-up resistors should not be used on these pins.
`nCE`	N/A	All	Input	Active-low chip enable. The `nCE` pin activates the device with a low signal to allow configuration. The `nCE` pin must be held low during configuration, initialization, and user mode. In single device configuration, it should be tied low. In multi-device configuration, `nCE` of the first device is tied low while its `nCEO` pin is connected to `nCE` of the next device in the chain. The `nCE` pin must also be held low for successful JTAG programming of the FPGA.

Table 8–20. Dedicated Configuration Pins (Part 3 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCEO	N/A	All	Output	Output that drives low when device configuration is complete. In single device configuration, this pin is left floating. In multi-device configuration, this pin feeds the next device's nCE pin. The nCEO of the last device in the chain is left floating.
DCLK	N/A	Synchronous configuration schemes (PS and PPS)	Input	Clock input used to clock data from an external source into the target device. Data is latched into the FPGA on the rising edge of DCLK. In PPA mode, DCLK should be tied high to V_{CC} to prevent this pin from floating. After configuration, this pin is tri-stated. In schemes that use a configuration device, DCLK will be driven low after configuration is done. In schemes that use a control host, DCLK should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA0	N/A	All	Input	Data input. In serial configuration modes, bit-wide configuration data is presented to the target device on the DATA0 pin. After configuration, EPC1 and EPC1441 devices tri-state this pin, while EPC2 devices drive this pin high. In schemes that use a control host, DATA0 should be driven either high or low, whichever is more convenient. Toggling this pin after configuration does not affect the configured device.
DATA[7..1]	I/O	Parallel configuration schemes (PPS and PPA)	Inputs	Data inputs. Byte-wide configuration data is presented to the target device on DATA[7..0]. In serial configuration schemes, they function as user I/O pins during configuration, which means they are tri-stated. After PPA or PPS configuration, DATA[7..1] are available as a user I/O pins and the state of these pin depends on the Dual-Purpose Pin settings.
DATA7	I/O	PPA	Bidirectional	In the PPA configuration scheme, the DATA7 pin presents the RDYnBSY signal after the nRS signal has been strobed low. In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, DATA7 is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.

Table 8–20. Dedicated Configuration Pins (Part 4 of 4)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nWS	I/O	PPA	Input	Write strobe input. A low-to-high transition causes the device to latch a byte of data on the DATA[7..0] pins. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nWS is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nRS	I/O	PPA	Input	Read strobe input. A low input directs the device to drive the RDYnBSY signal on the DATA7 pin. If the nRS pin is not used in PPA mode, it should be tied high. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nRS is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
RDYnBSY	I/O	PPS and PPA	Output	Ready output. A high output indicates that the target device is ready to accept another data byte. A low output indicates that the target device is busy and not ready to receive another data byte. In PPS and PPA configuration schemes, this pin will drive out high after power-up, before configuration and after configuration before entering user-mode. In non-PPS and non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPS and PPA configuration, RDYnBSY is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.
nCS/CS	I/O	PPA	Input	Chip-select inputs. A low on nCS and a high on CS select the target device for configuration. The nCS and CS pins must be held active during configuration and initialization. During the PPA configuration mode, it is only required to use either the nCS or CS pin. Therefore, if only one chip-select input is used, the other must be tied to the active state. For example, nCS can be tied to ground while CS is toggled to control configuration. In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated. After PPA configuration, nCS and CS are available as a user I/O pins and the state of these pins depends on the Dual-Purpose Pin settings.

Table 8–21 describes the optional configuration pins. If these optional configuration pins are not enabled in the Quartus II software, they are available as general-purpose user I/O pins. Therefore during configuration they function as user I/O pins, which means they are tri-stated with weak pull-up resistors.

Table 8–21. Optional Configuration Pins

Pin Name	User Mode	Pin Type	Description
`CLKUSR`	N/A if option is on. I/O if option is off.	Input	Optional user-supplied clock input. Synchronizes the initialization of one or more devices. This pin is enabled by turning on the *Enable user-supplied start-up clock (CLKUSR)* option in the Quartus II software
`INIT_DONE`	N/A if option is on. I/O if option is off.	Output open-drain	Status pin. Can be used to indicate when the device has initialized and is in user mode. When `nCONFIG` is low and during the beginning of configuration, the `INIT_DONE` pin is tri-stated and pulled high due to an external 1-kΩ 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 and the FPGA enters user mode. Thus, the monitoring circuitry must be able to detect a low-to-high transition. This pin is enabled by turning on the *Enable INIT_DONE output* option in the Quartus II software.
`DEV_OE`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows the user to override all tri-states on the device. When this pin is driven low, all I/O pins are tri-stated; when this pin is driven high, all I/O pins behave as programmed. This pin is enabled by turning on the *Enable device-wide output enable (DEV_OE)* option in the Quartus II software.
`DEV_CLRn`	N/A if option is on. I/O if option is off.	Input	Optional pin that allows you to override all clears on all device registers. When this pin is driven low, all registers are cleared; when this pin is driven high, all registers behave as programmed. This pin is enabled by turning on the *Enable device-wide reset (DEV_CLRn)* option in the Quartus II software.

JTAG pins must be kept stable before and during configuration. JTAG pin stability prevents accidental loading of JTAG instructions. Table 8–22 describes the dedicated JTAG pins.

Table 8–22. Dedicated JTAG Pins

Pin Name	User Mode	Pin Type	Description
TDI	N/A	Input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TDO	N/A	Output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by leaving this pin unconnected.
TMS	N/A	Input	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur on the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to V_{CC}.
TCK	N/A	Input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.
TRST	N/A	Input	Active-low input to asynchronously reset the boundary-scan circuit. The TRST pin is optional according to IEEE Std. 1149.1. If the JTAG interface is not required on the board, the JTAG circuitry can be disabled by connecting this pin to GND.

Configuration Handbook, Volume 2

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

CF5V2-2.1

nsai

I.S. EN ISO 9001

Contents

Section II. Software Settings

Section IV. Board Layout Tips & Debugging Techniques

Chapter 11. Debugging Configuration Problems

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:	*July 2004*
	Part number:	*CF52001-2.0*
Chapter 2.	Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet	
	Revised:	*July 2004*
	Part number:	*CF52002-2.0*
Chapter 3.	Altera Enhanced Configuration Devices	
	Revised:	*July 2004*
	Part number:	*S52014-2.0*
Chapter 4.	Serial Configuration Devices (EPCS1, EPCS4, EPCS16 & EPCS64) Features	
	Revised:	*January 2005*
	Part number:	*C51014-1.3*
Chapter 5.	Configuration Devices for SRAM-Based LUT Devices Data Sheet	
	Revised:	*July 2004*
	Part number:	*CF52005-2.0*
Chapter 6.	Device Configuration Options	
	Revised:	*July 2004*
	Part number:	*CF52006-2.0*
Chapter 7.	Configuration File Formats	
	Revised:	*July 2004*
	Part number:	*CF52007-2.0*
Chapter 8.	Configuring Mixed Altera FPGA Chains	
	Revised:	*July 2004*
	Part number:	*CF52008-2.0*
Chapter 9.	Combining Different Configuration Schemes	
	Revised:	*July 2004*
	Part number:	*CF52009-2.0*

Chapter 10. Using Flash Memory to Configure FPGAs

Revised:	*July 2004*
Part number:	*CF52010-2.0*

Chapter 11. Debugging Configuration Problems

Revised:	*July 2004*
Part number:	*CF52011-2.0*

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, 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:

Revision History

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

Chapter	Date/Version	Changes Made
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.
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	July 2004, v2.0	● Added text regarding pointing to an incorrect page after Figure 3–8. ● Renamed .hexpof to .hexout throughout chapter.
	September 2003, v1.0	Initial Release.

Chapter	Date/Version	Changes Made
4	January 2005, v1.3	● 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	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.

1. Altera Configuration Devices

CF52001-2.1

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, see *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, see *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, see *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 see 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 *(4)*	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

CF52002-2.0

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

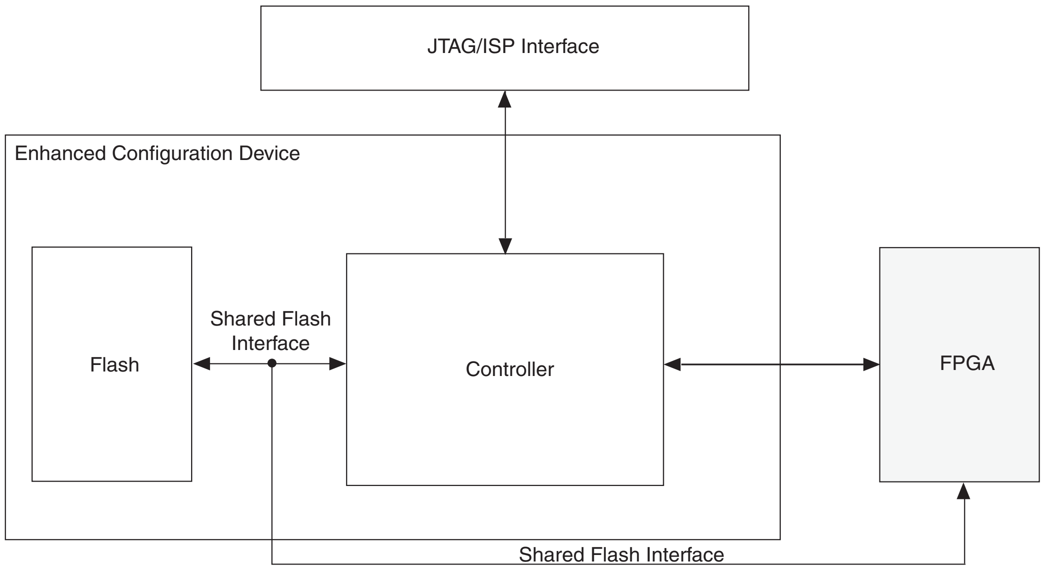
The Altera enhanced configuration device is a single-device, high-speed, advanced configuration solution for very high-density FPGAs. The core of an enhanced configuration device is divided into two major blocks, a configuration controller and a flash memory. The flash memory is used to store configuration 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 Stratix Handbook *Using Remote System Configuration with Stratix & Stratix GX Devices* chapter.

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.

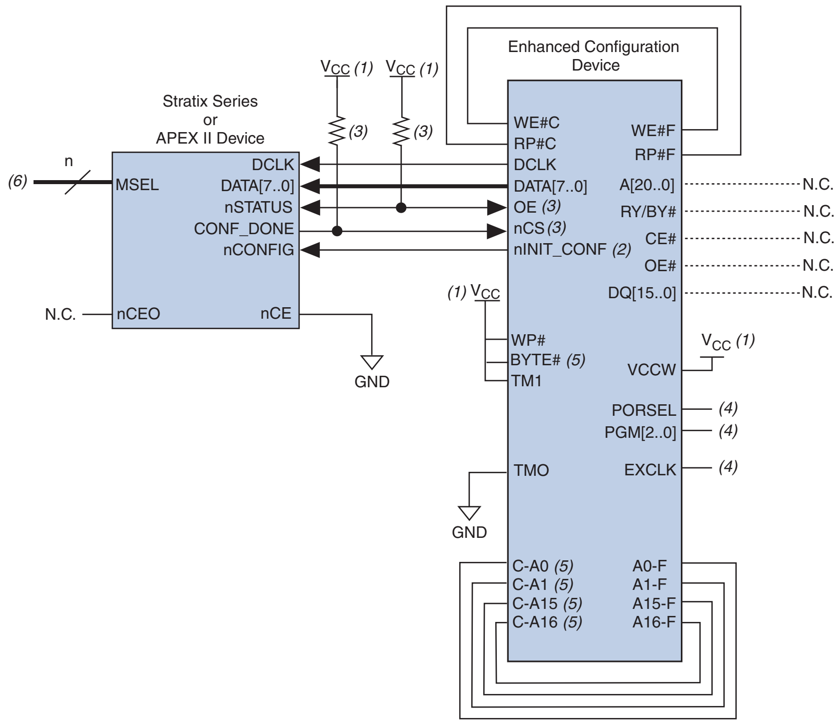
Table 2–3. Configuration Signals

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).

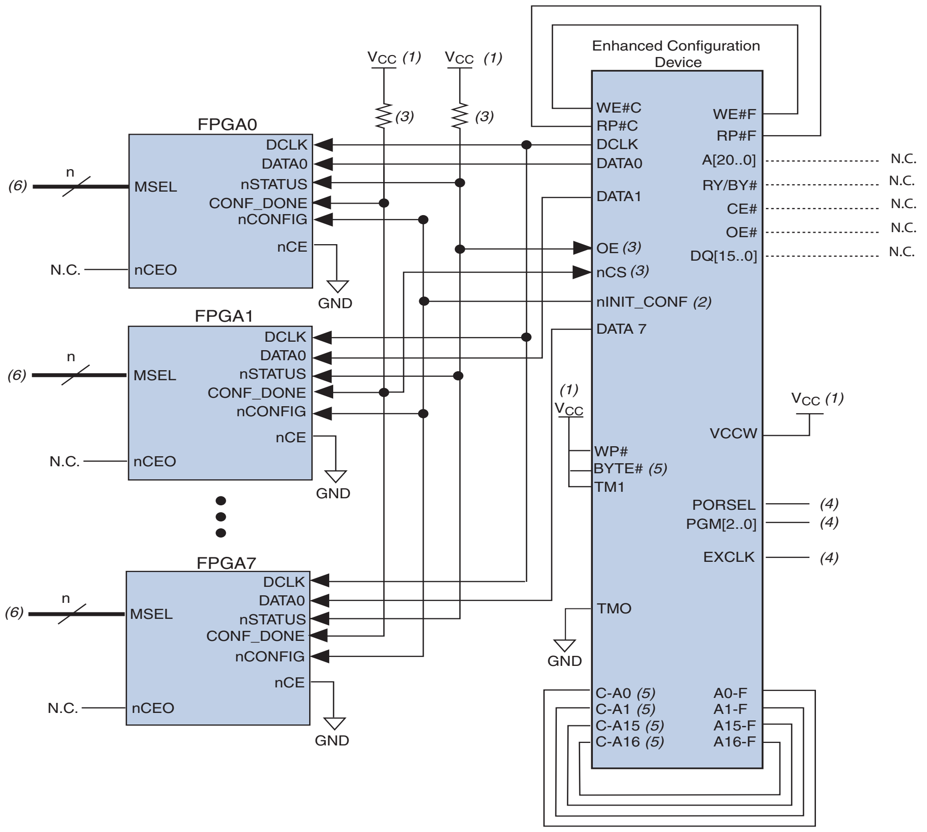
For configuration schematics and more information on multi-device PS configuration, refer to the appropriate FPGA family chapter in the Configuration Handbook.

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 Chapter 3, Altera Enhanced Configuration Devices 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 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:

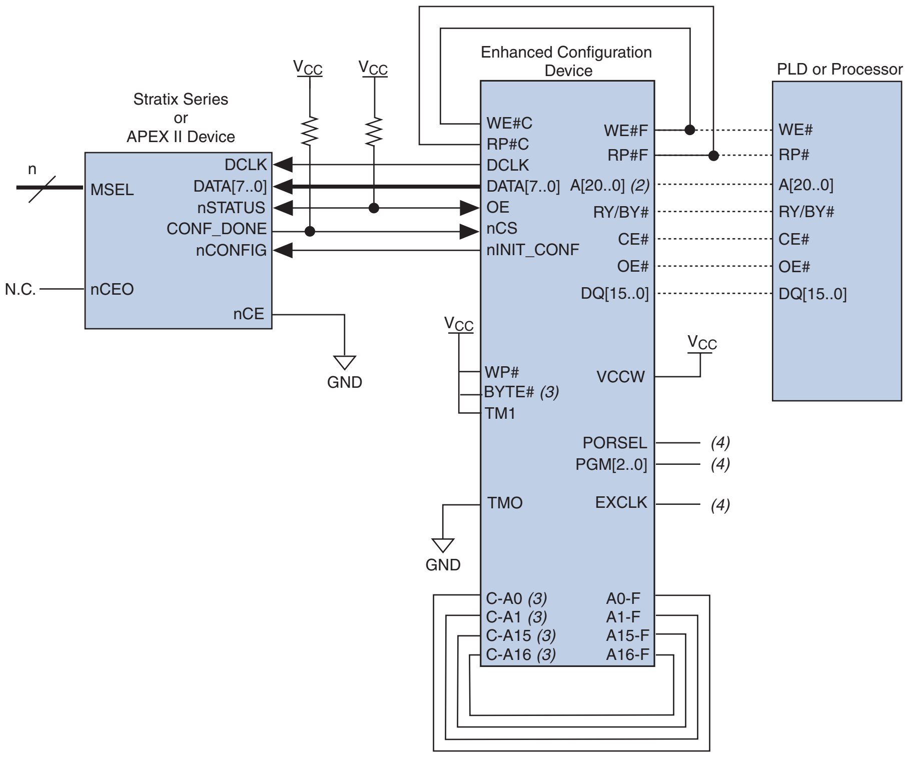
- FPGA configuration—reading configuration data from flash
- JTAG-based flash programming—storing configuration data in flash
- At POR—reading option bits from flash

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 Chapter 3, Altera Enhanced Configuration Devices 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 Chapter 3, Altera Enhanced Configuration Devices 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%

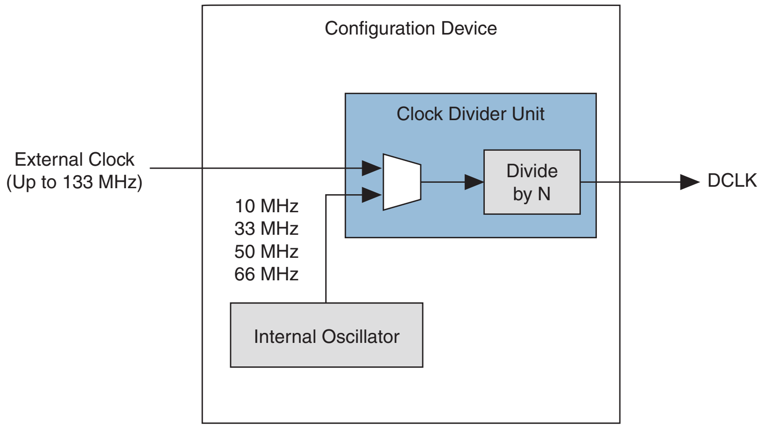
Note to Table 2–5:

(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 Chapter 3, Altera Enhanced Configuration Devices in 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 Chapter 3, Altera Enhanced Configuration Devices 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. See 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

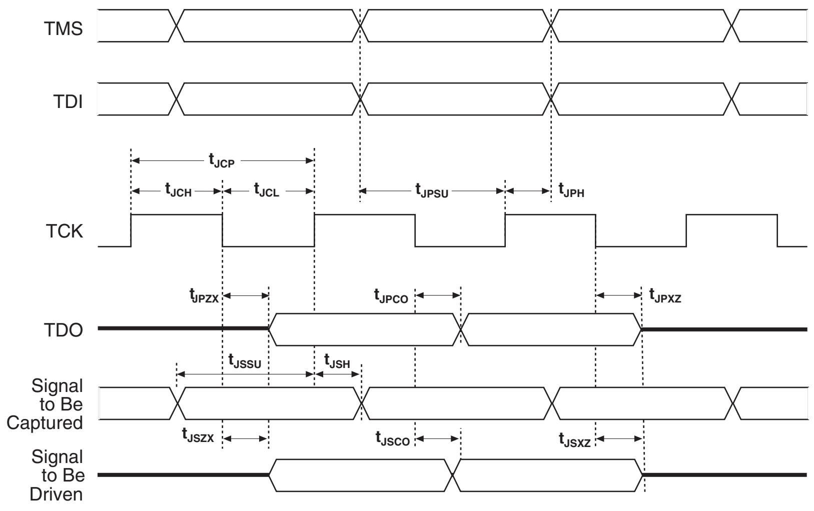


Table 2–12 shows the timing parameters and values for the enhanced configuration device.

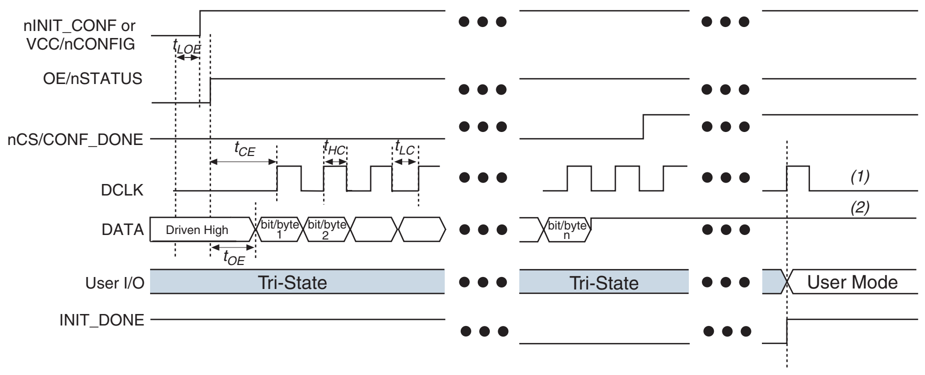
Table 2–12. JTAG Timing Parameters & Values

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

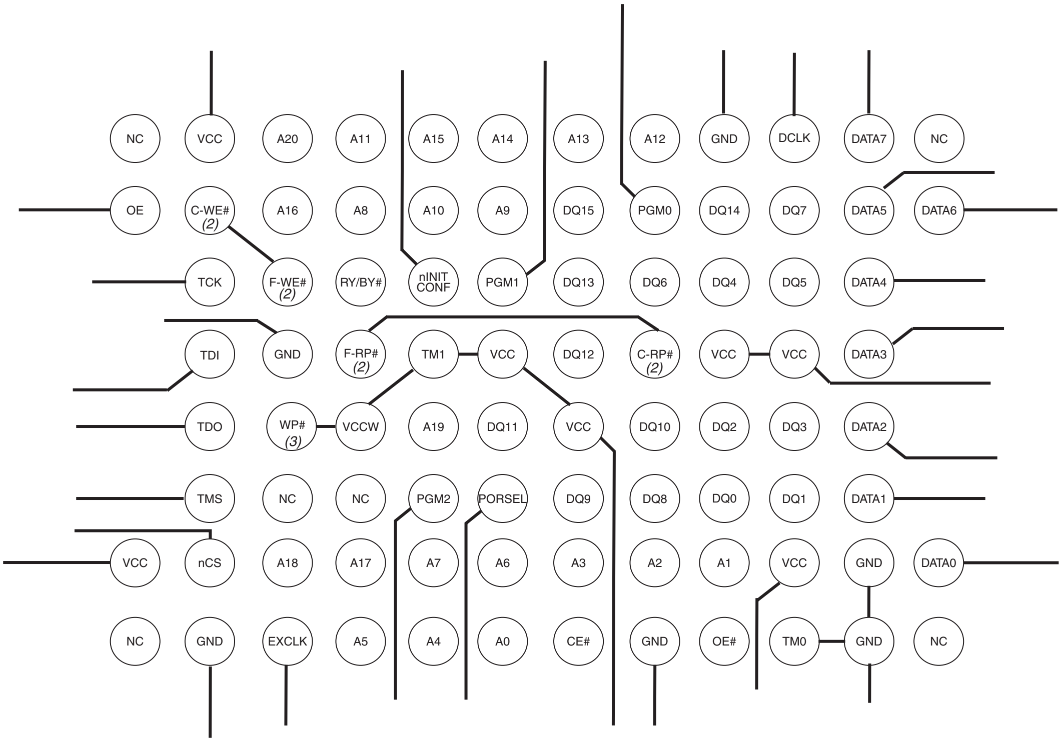
Package

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.

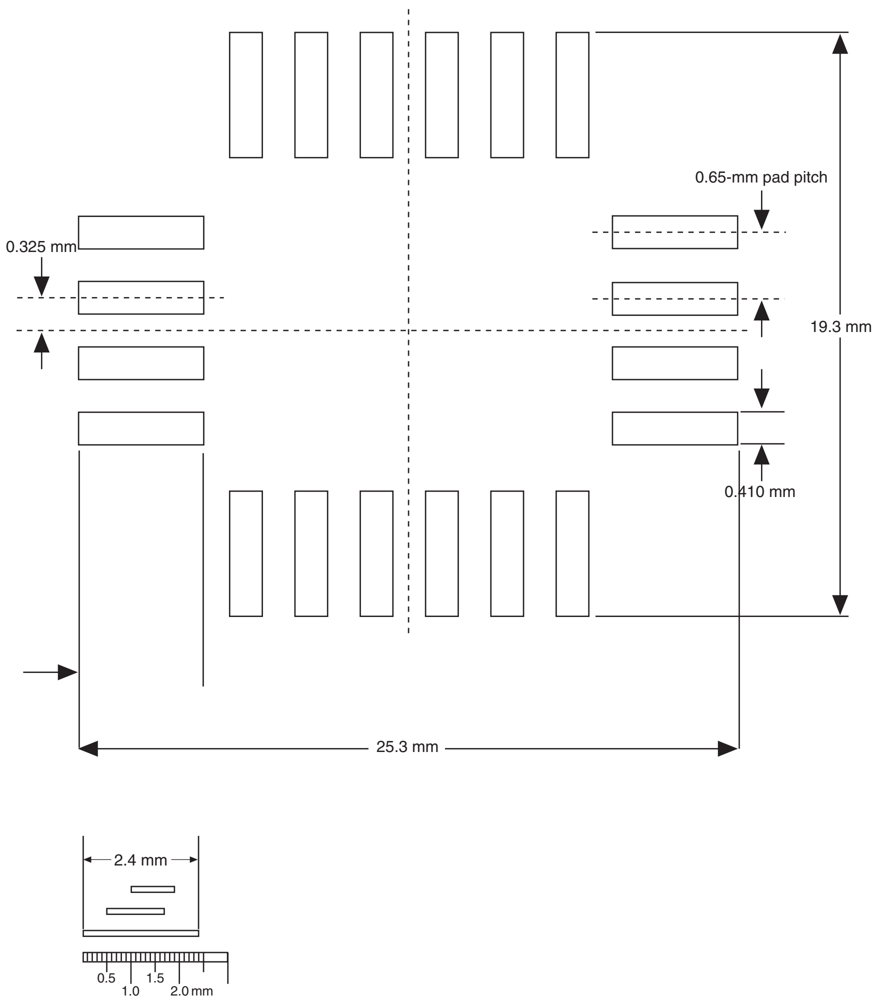
(2) `F-RP#` and `F-WE#` are pins on the flash die. `C-RP#` and `C-WE#` are pins on the controller die. `C-WE#` and `F-WE#` should be connected together on the PCB. `F-RP#` and `C-RP#` should also be connected together on the PCB.

(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

S52014-2.0

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

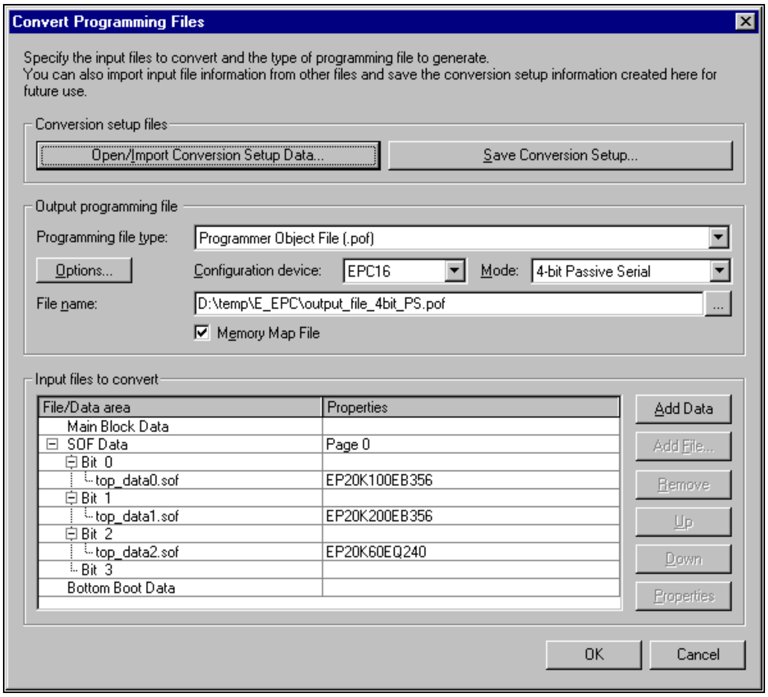
Number of Devices *(1)*	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

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


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

Open/Import Conversion Setup Data... | Save Conversion Setup...

Output programming file

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

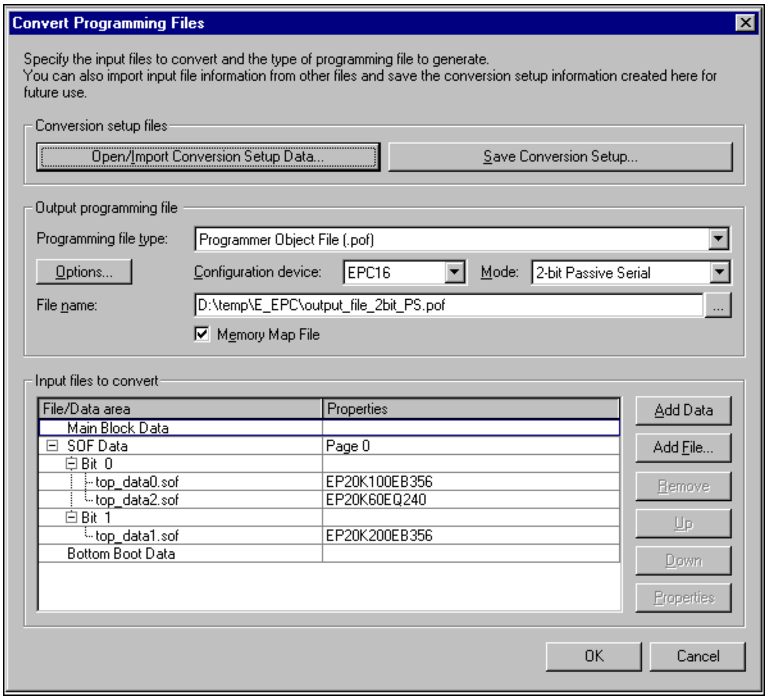
☑ Memory Map File

Input files to convert

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

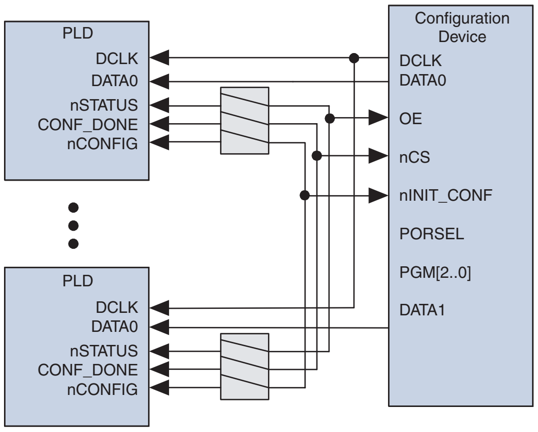
Input files to convert

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.

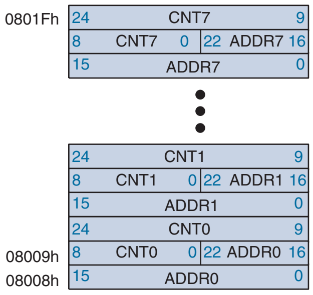
With the ability to dynamically switch pages, you can also configure Altera devices with various revisions for debugging without having to reprogram the configuration device. For example, you can configure a device that is on "stand-by" to perform another function and then reconfigure it back with the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

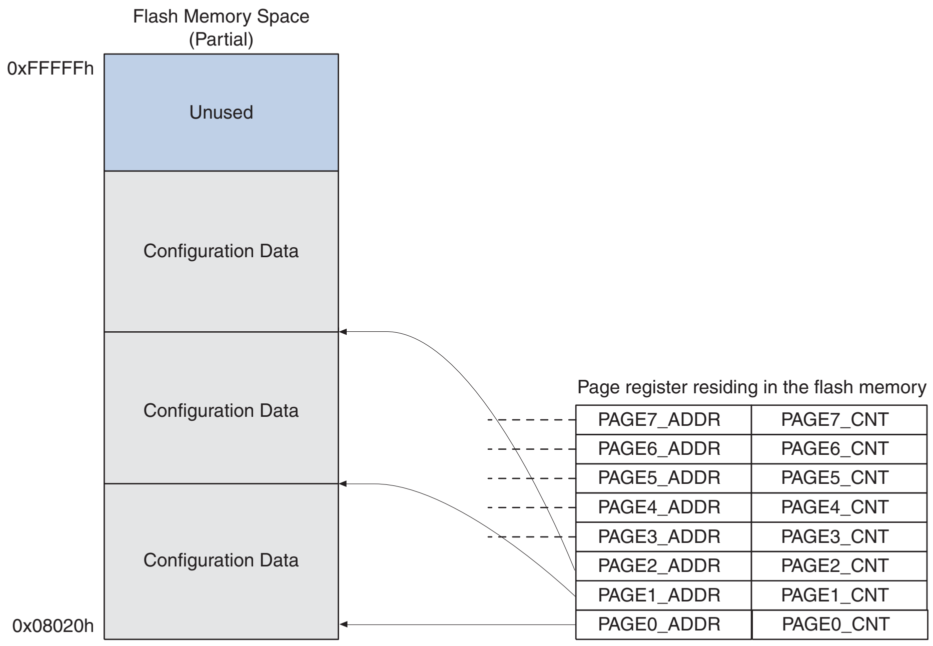
External page mode input pins `PGM[2..0]` determine which page to use during PLD configuration, and page pointers determine the data location. Each page pointer consists of a starting address register and a length count register. The word-addressable starting address register (23 bits) is used to determine 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.

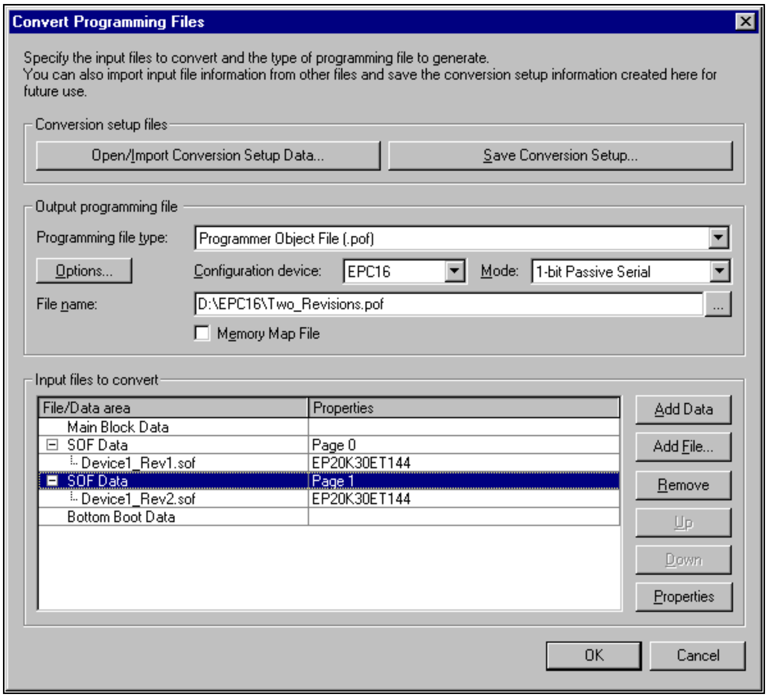
Software Implementation (Convert Programming Files)

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

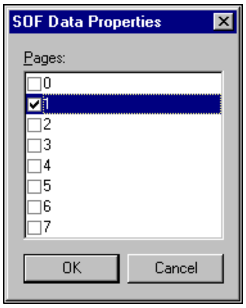
File name: D:\EPC16\Two_Revisions.pof

☐ 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).

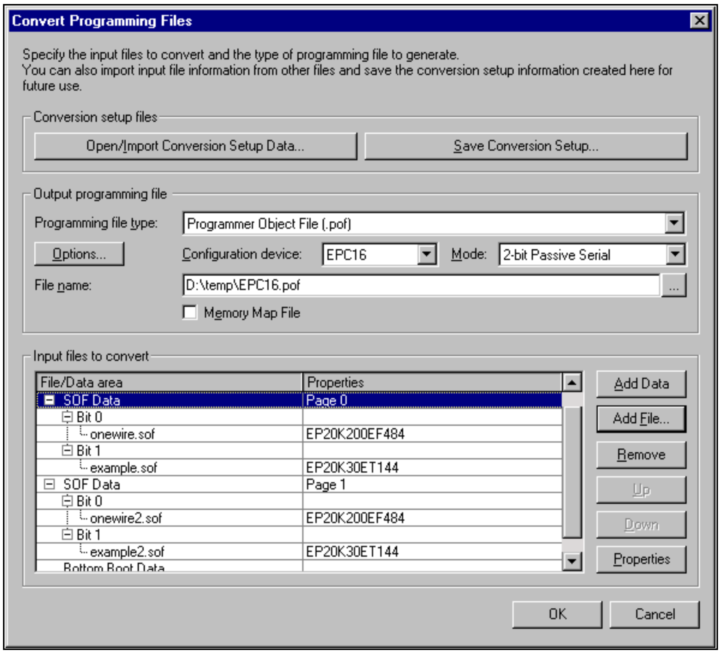
Figure 3–7. Software Setting for Selecting Pages



Pages: [✓ 1]

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
Open/Import Conversion Setup Data...
Save Conversion Setup...
Output programming file
Options...
Add Data
Add File...
Remove
Up
Down
Properties
OK
Cancel

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	

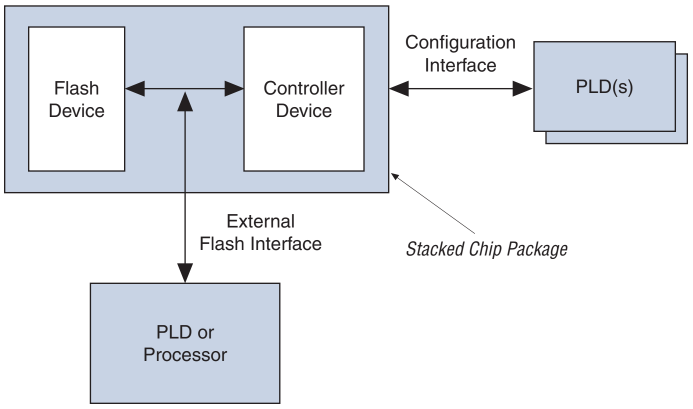
By selecting the *Memory Map File* option, the Quartus II Memory Map output file (**.map**) describing the flash memory address locations is generated. This information is typically useful when using the external flash interface feature.

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.

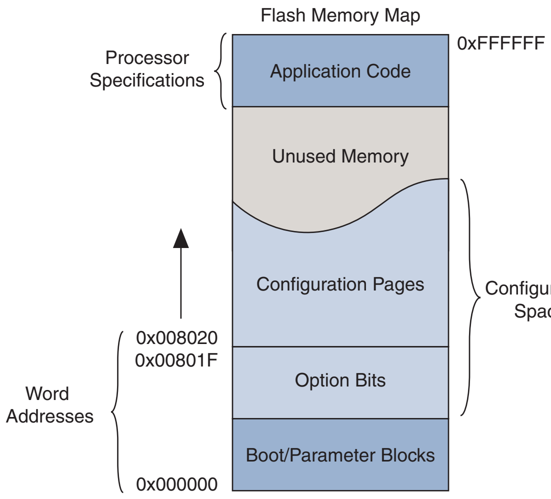
Flash Memory Map

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.

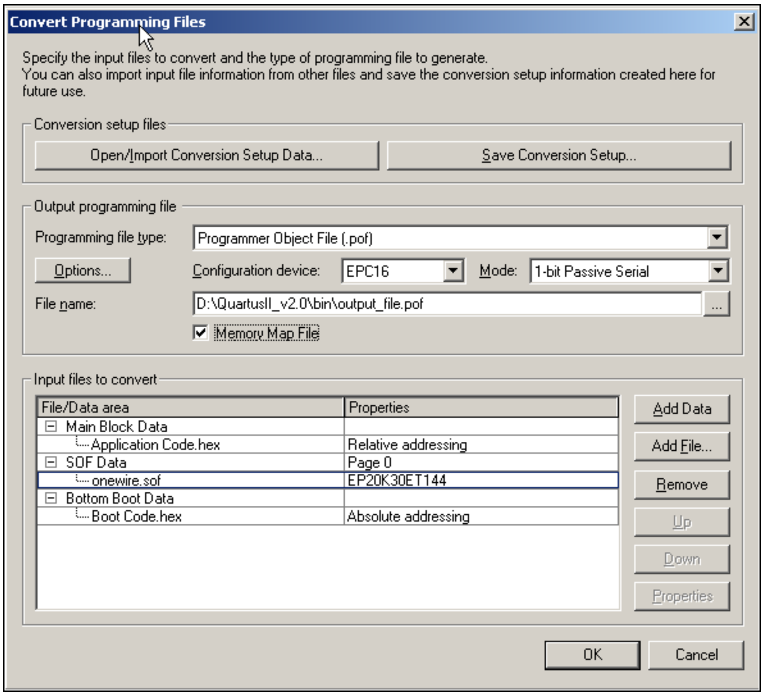
When you select relative addressing mode for **Main Block Data**, the Quartus II software aligns the last byte of information with the highest address (i.e., `0x1FFFFF`). Therefore, the starting address is dependent on the size of the **.hex** file. You can easily obtain the starting address of the application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

Open/Import Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: Programmer Object File (.pof)

Options... 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.

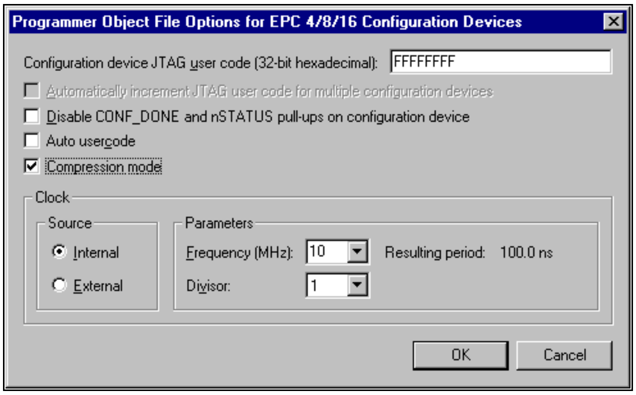
☞ Configuration data and processor space data could exist within the same physical data block. In such cases, erasing the physical data block would affect both configuration and processor data, requiring you to update both. You can avoid this situation by storing application data starting from the next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

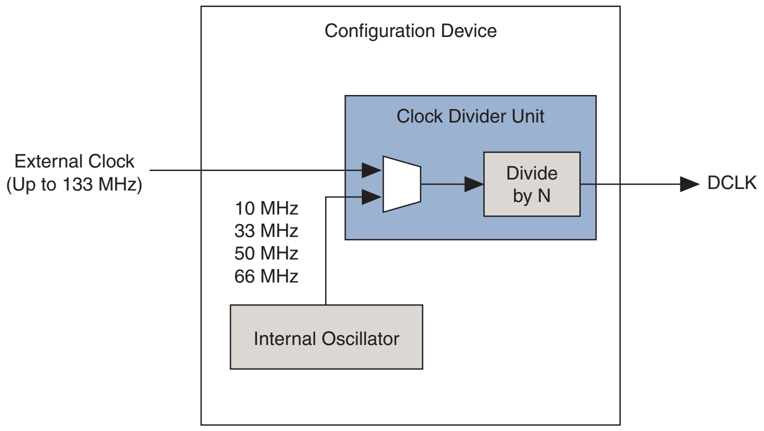
Clock Divider

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.

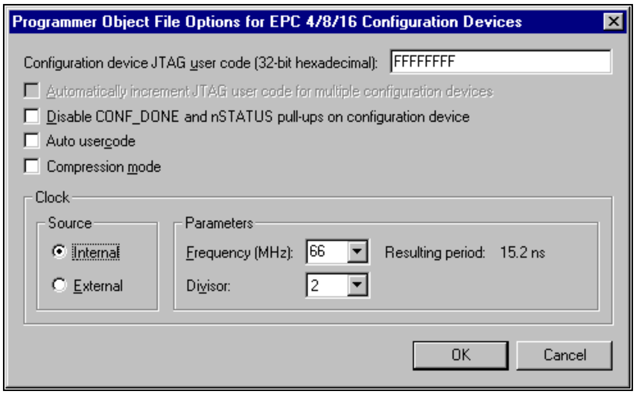
Figure 3–13. Clock Divider Unit in Enhanced Configuration Devices



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 — Resulting period: 15.2 ns — Divisor: 2

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

C51014-1.3

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™ 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 *(1)*
EPCS64	67,108,864 *(1)*

Note to Table 4–1:
(1) This information is preliminary.

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 only 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	
		EPCS16	EPCS64
EP2S15	5,000,000	✓	✓
EP2S30	10,100,000	✓	✓
EP2S60	17,100,000	✓ *(2)*	✓
EP2S90	27,500,000		✓
EP2S130	39,600,000		✓
EP2S180	52,400,000		✓

Notes to Table 4–2:
(1) These are preliminary, 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	1,223,980	✓ *(2)*	✓	✓	✓
EP2C8	1,983,792		✓	✓	✓
EP2C20	3,930,986		✓	✓	✓
EP2C35	7,071,234			✓	✓
EP2C50	9,122,148			✓	✓
EP2C70	10,249,694			✓	✓

Notes to Table 4–3:
(1) These are preliminary, 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 be used with all serial 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	925,000	✓	✓	✓	✓
EP1C6	1,167,216	✓ *(2)*	✓	✓	✓
EP1C12	2,326,528		✓	✓	✓
EP1C20	3,559,608		✓	✓	✓

Notes to Table 4–4:
(1) These are preliminary, 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.

See *Configuring Stratix II Devices* in the Configuration Handbook for more information regarding the Stratix II FPGA decompression feature.

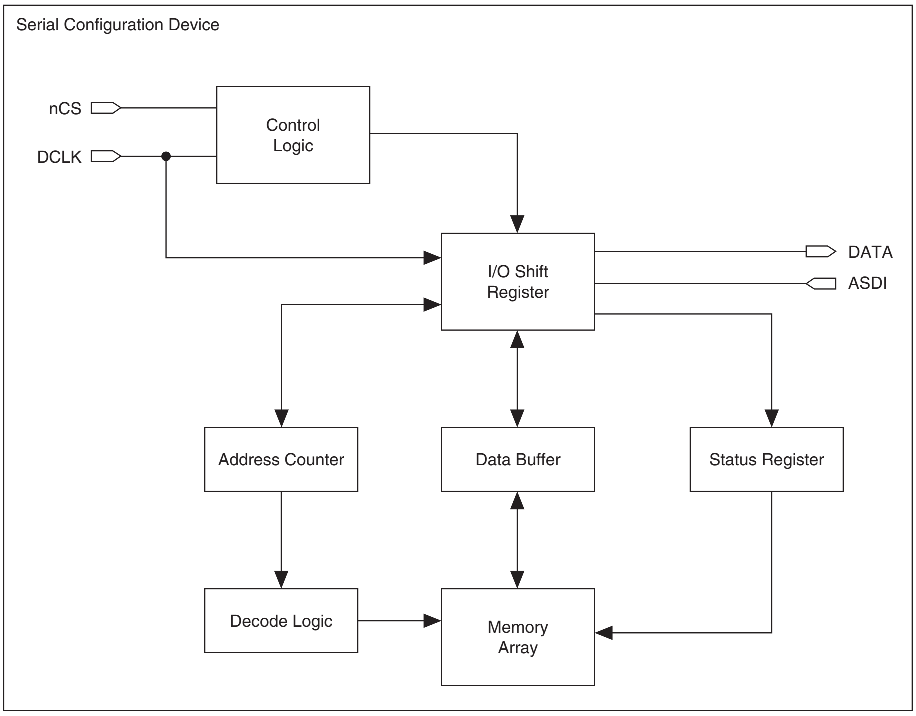
See *Configuring Cyclone II Devices* in the Configuration Handbook for more information regarding the Cyclone II FPGA decompression feature.

See *Configuring Cyclone FPGAs* in 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 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, see the *Active Serial Memory Interface Data Sheet*.

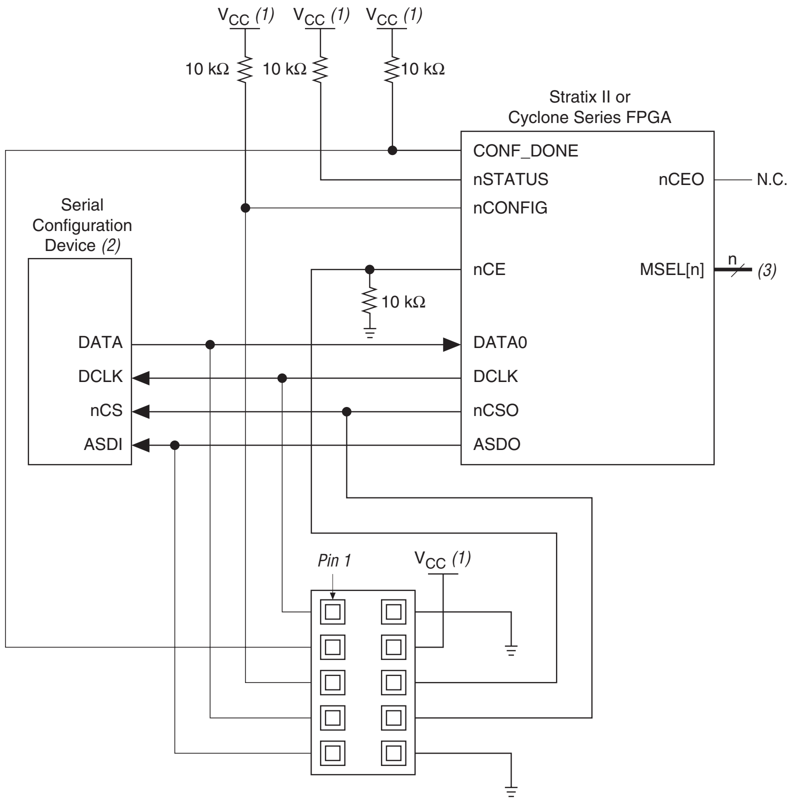
Active Serial FPGA Configuration

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)



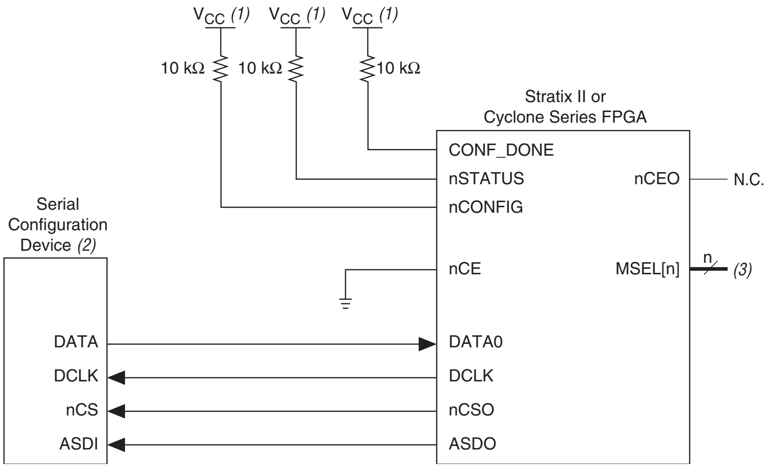
Notes to Figure 4–2:

(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 rising 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.

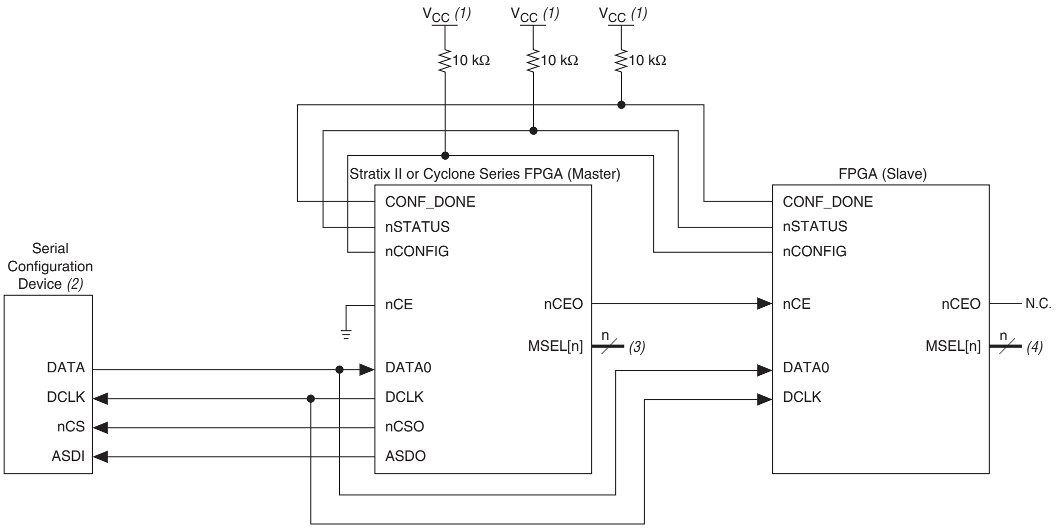
For more information on configuring Stratix II FPGAs in AS mode or other configuration modes, see *Configuring Stratix II Devices* in the Configuration Handbook.

For more information on configuring Cyclone II FPGAs in AS mode or other configuration modes, see Chapter 4, Configuring Cyclone II Devices in the *Configuration Handbook, Volume 1*.

For more information on configuring Cyclone FPGAs in AS mode or other configuration modes, see Chapter 5, Configuring Cyclone FPGAs in the *Configuration Handbook, Volume 1*.

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" 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)	524,888 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

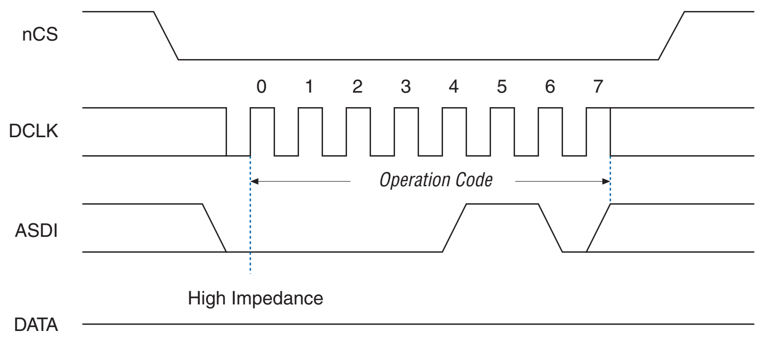
Operation	Operation Code (1)	Address Bytes	Dummy Bytes	Data Bytes	DCLK f_{MAX} (MHz)
Write enable	0000 0110	0	0	0	25
Write disable	0000 0100	0	0	0	25
Read status	0000 0101	0	0	1 to infinite (2)	25
Read bytes	0000 0011	3	0	1 to infinite (2)	20
Read silicon ID	1010 1011	0	3	1 to infinite (2)	25
Write status	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



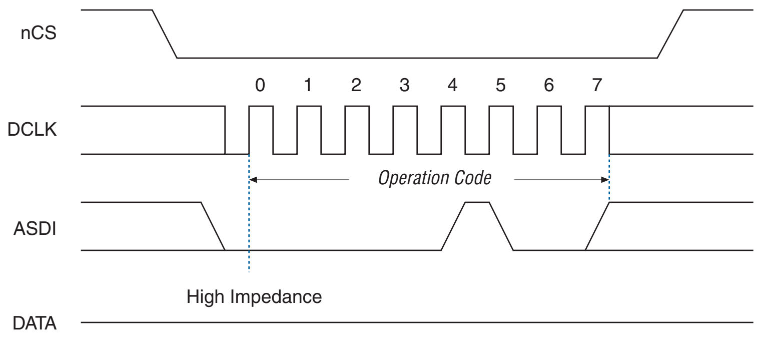
Write Disable Operation

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 Status Register Status Bits

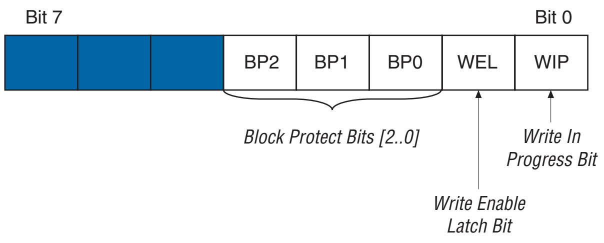
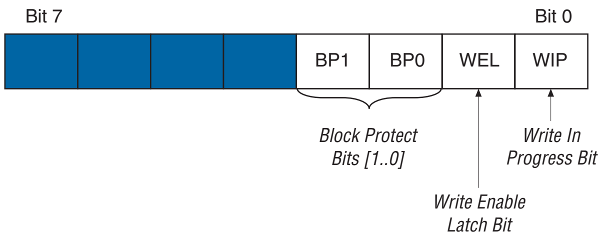


Figure 4–8. EPCS1 Status Register Status Bits



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 4–11 and 4–12 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 EPCS4 Devices

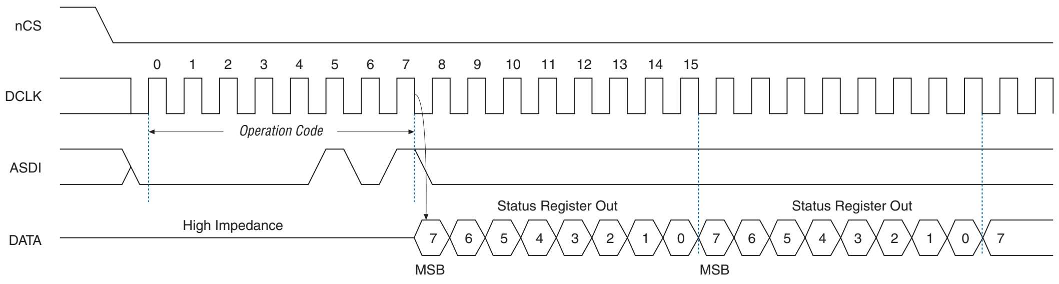
Status Register Content			Memory Content	
BP2 Bit	BP1 Bit	BP0 Bit	Protected Area	Unprotected Area
0	0	0	None	All eight sectors: 0 to 7
0	0	1	Sector 7	Seven sectors: 0 to 6
0	1	0	Sectors 6 and 7	Six sectors: 0 to 5
0	1	1	Four sectors: 4 to 7	Four sectors: 0 to 3
1	0	0	All sectors	None
1	0	1	All sectors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

Table 4–12. 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

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



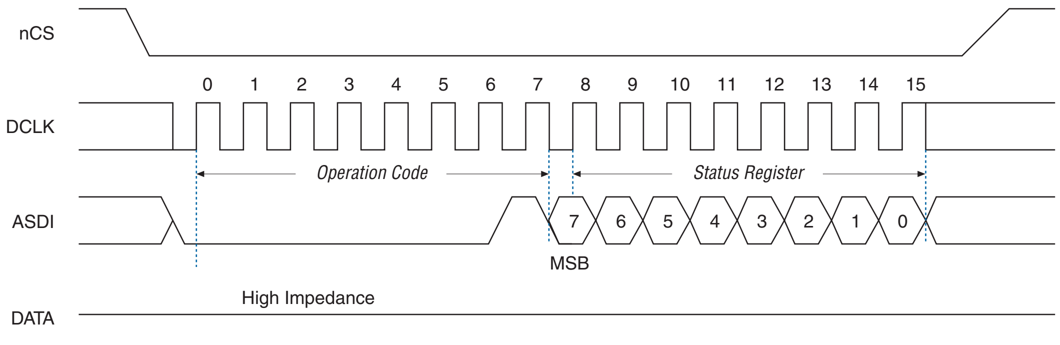
Write Status Operation

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 4–11 and 4–12. 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 both serial configuration devices and is guaranteed to be less than 15 ms (see t_{WS} in Table 4–14). 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

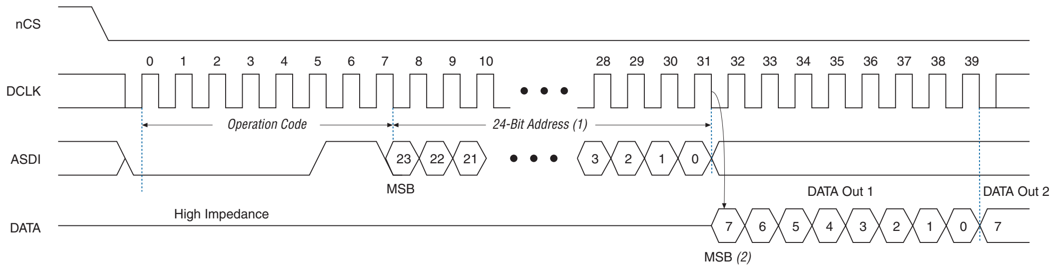


Read Bytes Operation

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

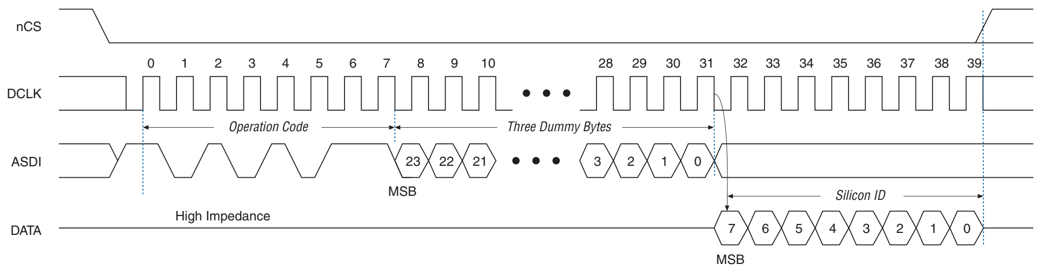
The read silicon ID operation code is `b'1010 1011`, with the MSB listed first. This operation reads the serial configuration device's 8-bit silicon ID from the `DATA` output pin. If this operation is shifted in during an erase or write cycle, it will be ignored and have no effect on the cycle that is in progress. Table 4–13 shows the EPCS1 and EPCS4 device silicon IDs.

Table 4–13. 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.

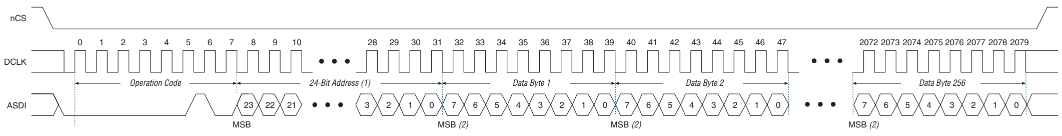
If more than 256 data bytes are shifted into the serial configuration device with a write bytes operation, the previously latched data is discarded and the last 256 bytes are written to the page. However, if less than 256 data bytes are shifted into the serial configuration device, they are guaranteed 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 for EPCS4 devices and 2 ms for EPCS1 devices and is guaranteed to be less than 5 ms (see t_{WB} in Table 4–14). 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.

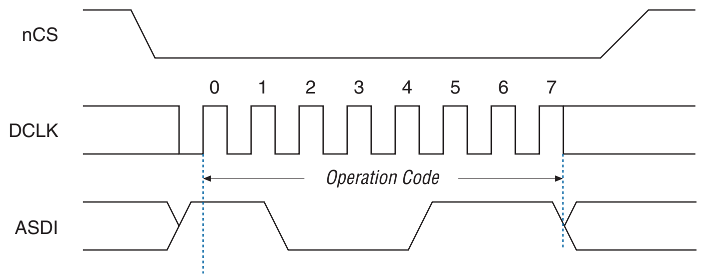
Erase Bulk Operation

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 5 s for EPCS4 devices (guaranteed to be less than 10 s) or 3 s for EPCS1 devices (guaranteed to be less than 6 s). See t_{EB} in Table 4–14. 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

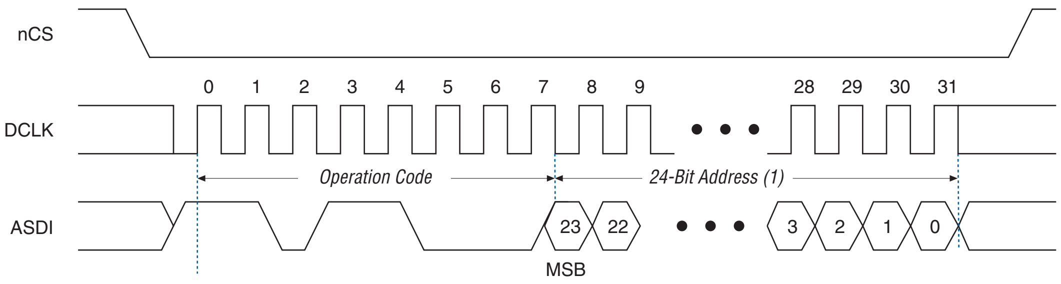
The erase sector operation code is `b'1101 1000`, with the MSB listed first. The erase sector operation allows the user to erase a certain sector in the serial configuration device by setting all bits inside the sector to `1` or `0xFF`. This operation is useful for users who access the unused sectors 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 both 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 4–20).

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

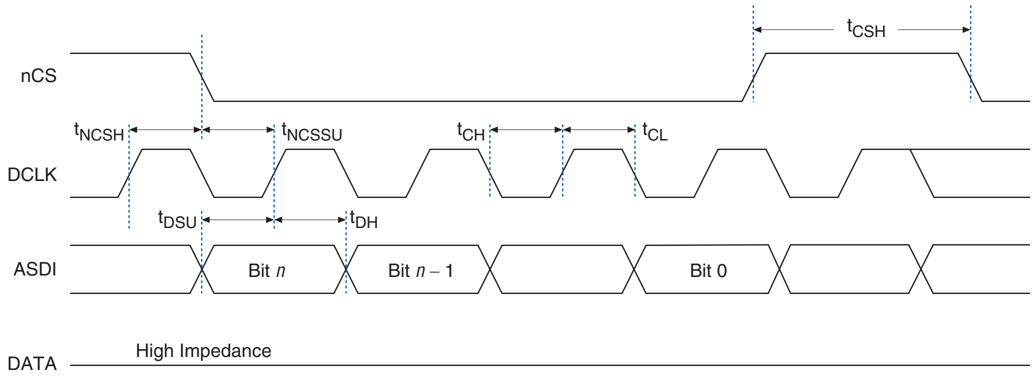


Table 4–14 defines the serial configuration device timing parameters for write operation.

Table 4–14. Write Operation Parameters

Symbol	Parameter	Min	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_EPCS1} *(1)*	Write bytes cycle time for EPCS1 devices	2	5	ms
t_{WB_EPCS4} *(1)*	Write bytes cycle time for EPCS4 devices	1.5	5	ms
t_{WB_EPCS16} *(1)*	Write bytes cycle time for EPCS16 devices	1.5	5	ms
t_{WB_EPCS64} *(1)*	Write bytes cycle time for EPCS64 devices	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 4–14:

(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

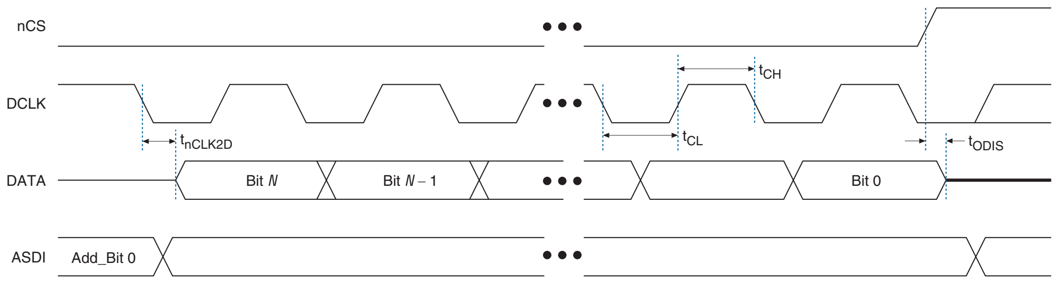


Table 4–15 defines the serial configuration device timing parameters for read operation.

Table 4–15. 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

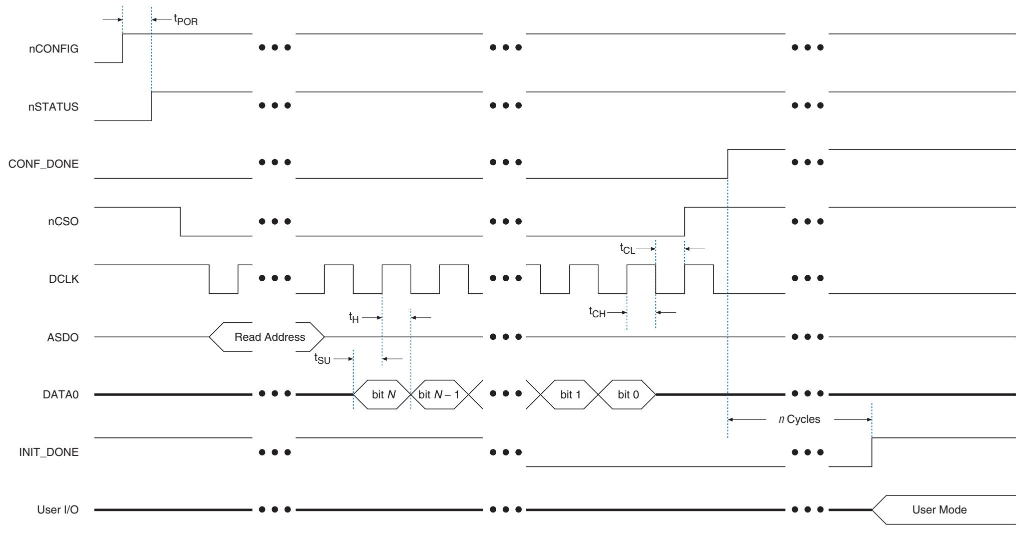


Table 4–16 shows the timing parameters for AS configuration mode.

Table 4–16. 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 *(1)*	20 *(2)*	26 *(2)*	40 *(2)*	MHz
		10	13	20	MHz
t_{CH}	DCLK high time	25			ns
t_{CL}	DCLK low time	25			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

Notes to Table 4–16:

(1) These values are preliminary

(2) Only the EPCS16 and EPCS64 devices support a DCLK frequency up to 40 MHz.

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, see 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, 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 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*

Operating Conditions

Tables 4–17 through 4–21 provide information on absolute maximum ratings, recommended operating conditions, DC operating conditions, and capacitance for serial configuration devices.

Table 4–17. 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 4–18. 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 4–19. 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 4–20. 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	14	mA

Table 4–21. 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 4–17 through 4–21:
(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 4–22 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

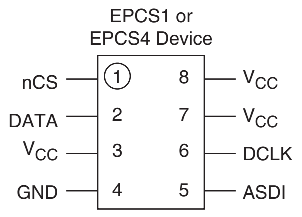
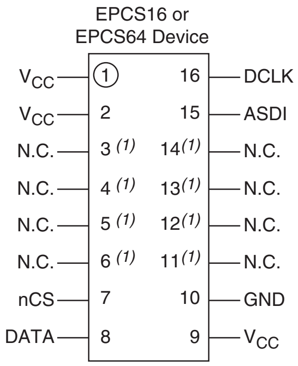


Figure 4–20 shows the Altera serial configuration device 16-pin SOIC package and its pin-out diagram.

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 4–22. Serial Configuration Device Pin Description

Pin Name	Pin Number	Pin Type	Description
DATA	2	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	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	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.
DCLK	6	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	Power	Power pins connect to 3.3 V.
GND	4	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 4–23 shows the ordering codes for serial configuration devices.

Table 4–23. Serial Configuration Device Ordering Codes

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

Note to Table 4–23:
(1) N: Lead free

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

CF52005-2.0

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)

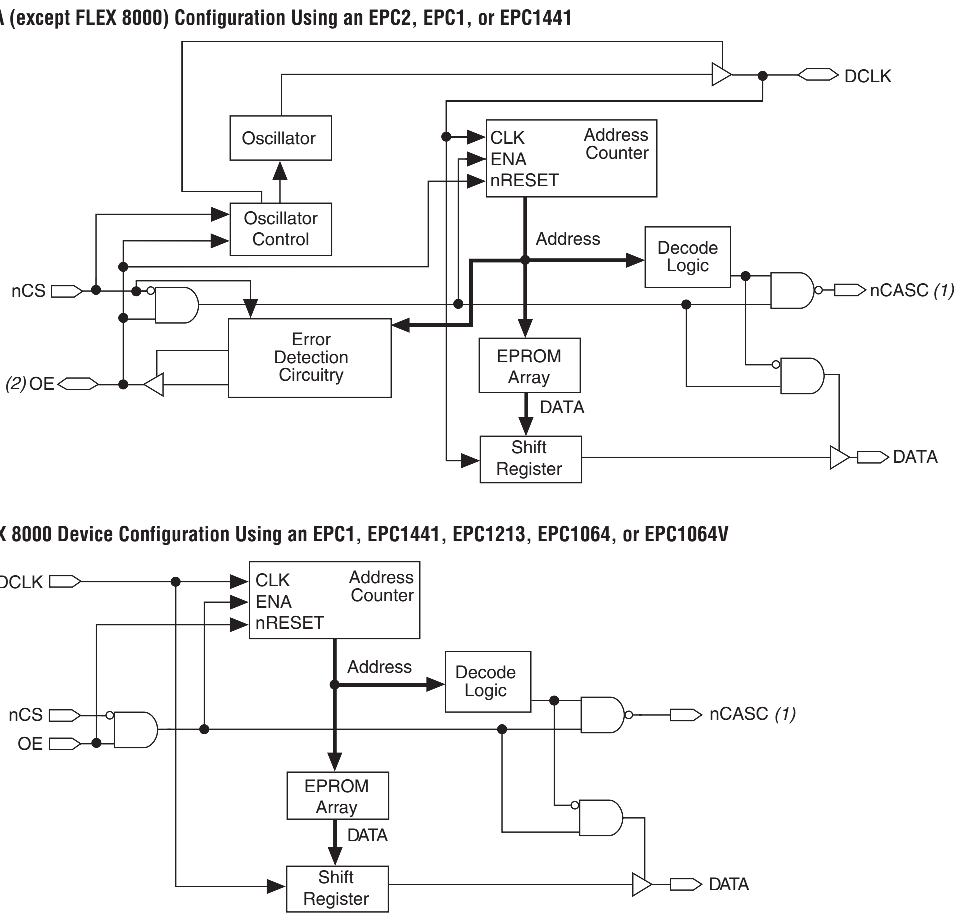
Family	Device	Data Size (Bits) (1)	EPC1064 /1064V	EPC1213	EPC1441	EPC1	EPC2
FLEX 10K (5.0V)	EPF10K10	118,000			1	1	1
	EPF10K20	231,000			1	1	1
	EPF10K30	376,000			1	1	1
	EPF10K40	498,000				1	1
	EPF10K50	621,000				1	1
	EPF10K70	892,000				1	1
	EPF10K100	1,200,000					1
FLEX 6000/A (3.3 V)	EPF6010A	260,000			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

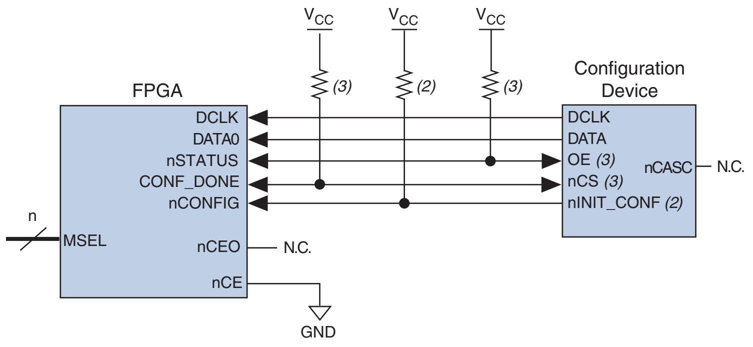
The EPC2, EPC1, and EPC1441 devices store configuration data in its EPROM array and serially clock data out using an internal oscillator. The `OE`, `nCS`, and `DCLK` pins supply the control signals for the address counter and the DATA output tri-state buffer. The configuration device sends a serial bitstream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

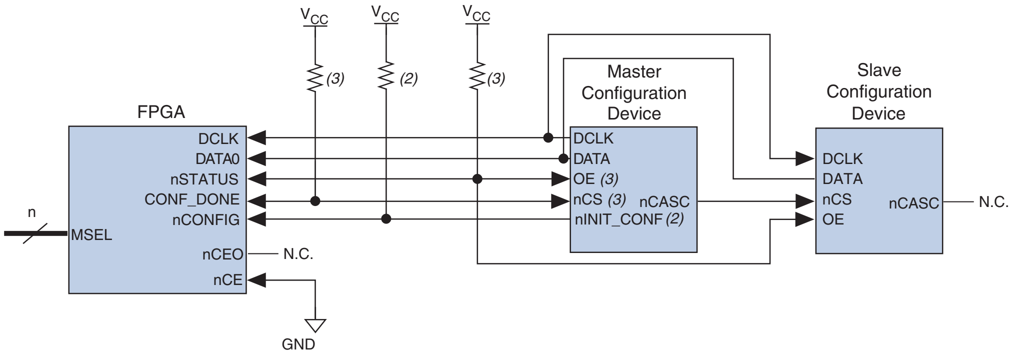
If the FPGA's configuration data exceeds the capacity of a single EPC2 or EPC1 configuration device, multiple EPC2 or EPC1 devices can be cascaded together. If multiple EPC2 or EPC1 devices are required, the nCASC and nCS pins provide handshaking between the configuration devices.

☞ 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, see 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, see *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

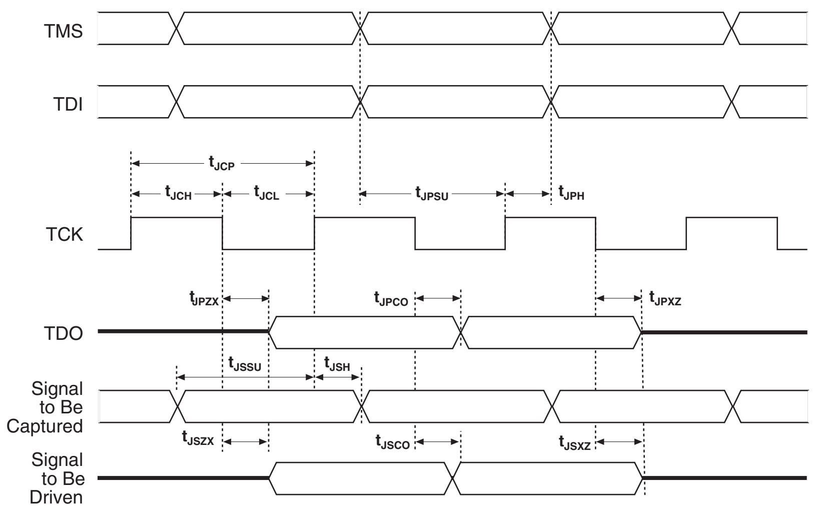


Table 5–7 shows the timing parameters and values for configuration devices.

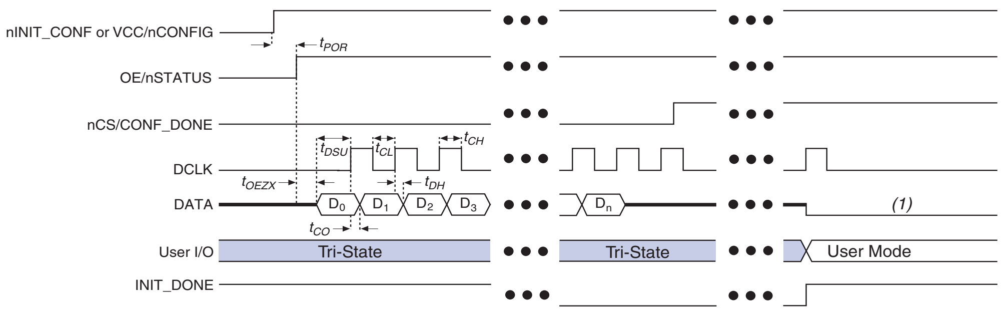
Table 5–7. JTAG Timing Parameters & Values

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:

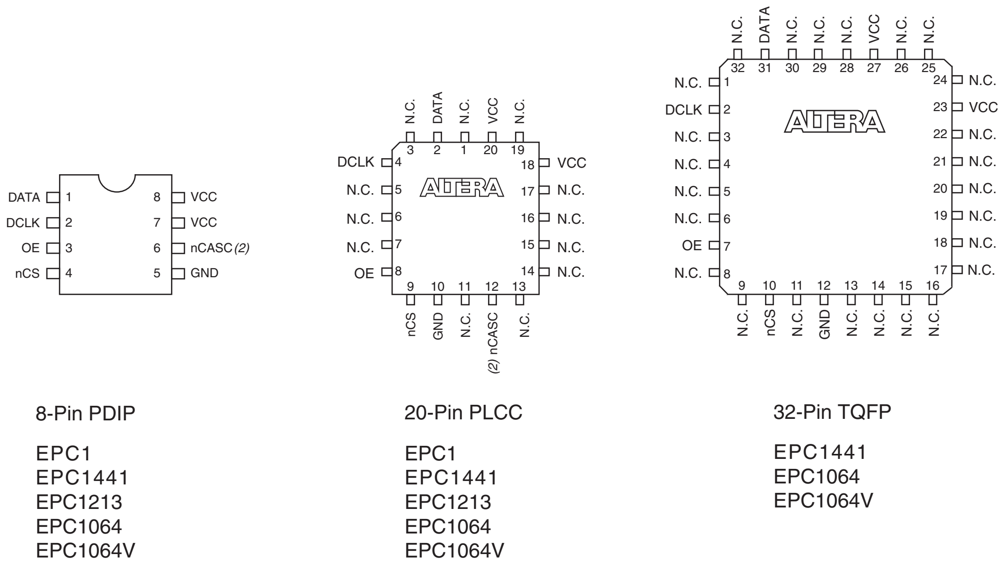
(1) This package is available for EPC1 and EPC1441 devices only.

(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)

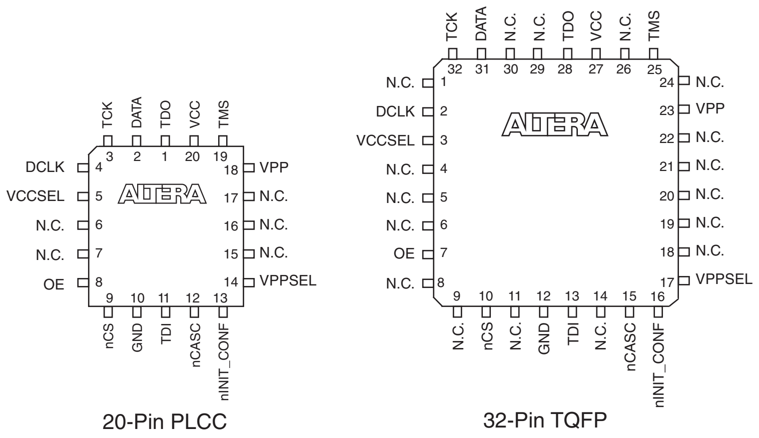


Notes to Figure 5–6:

(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	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	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.

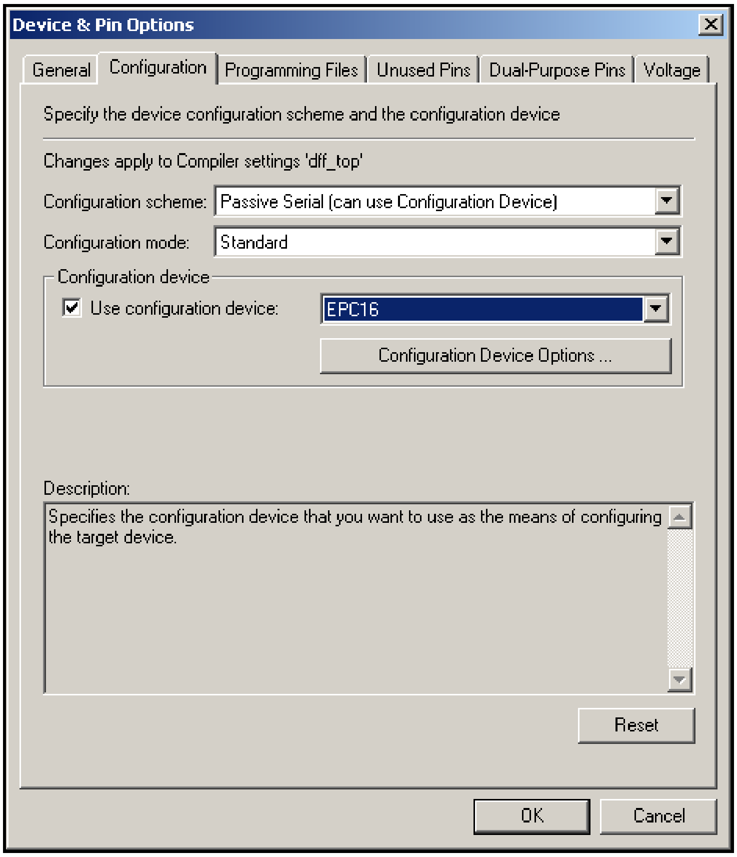
6. Device Configuration Options

CF52006-2.0

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 (see 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.

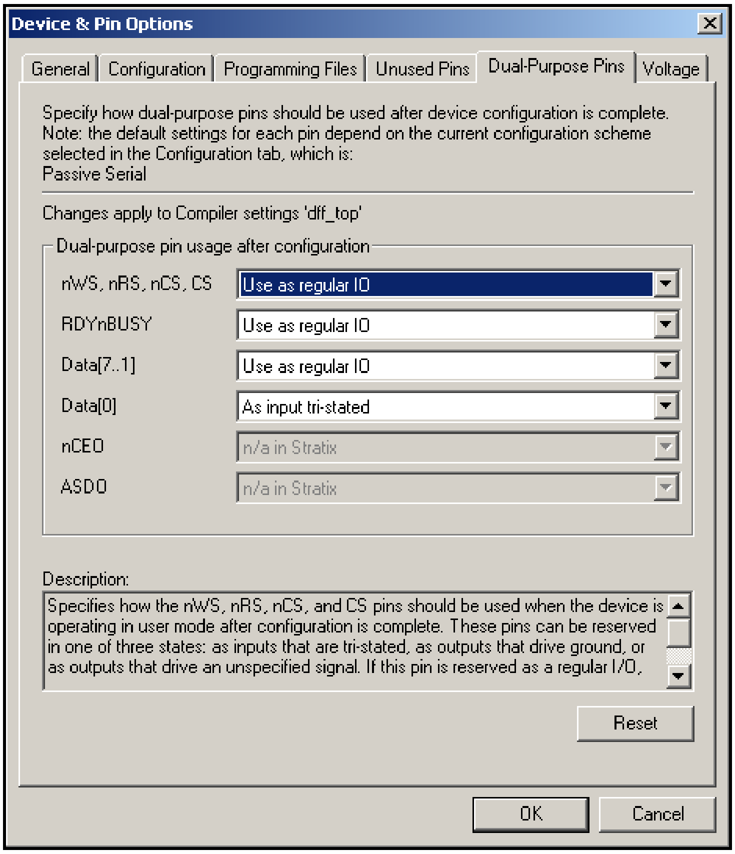
The **Configuration scheme** drop-down list will change with the chosen device family to only show the configuration schemes supported by that device family. The **Configuration mode** selection is only available for devices that support remote and local update, such as Stratix® and Stratix GX devices. 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, see 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



Device & Pin Options

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

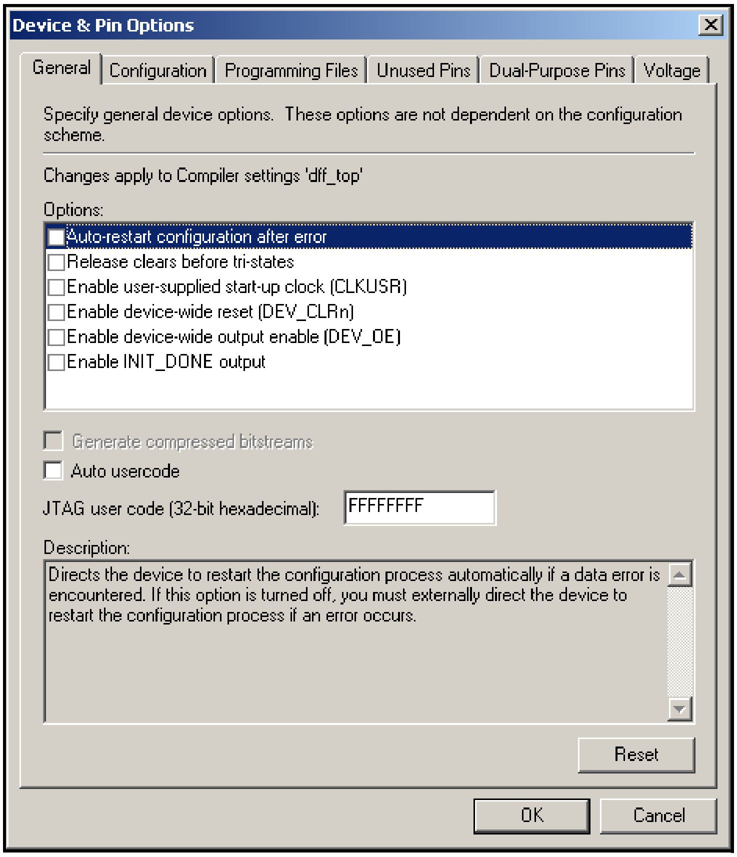
nWS, nRS, nCS, CS	Use as regular IO
RDYnBUSY	Use as regular IO
Data[7..1]	Use as regular IO
Data[0]	As input tri-stated
nCEO	n/a in Stratix
ASDO	n/a in Stratix

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

CF52007-2.0

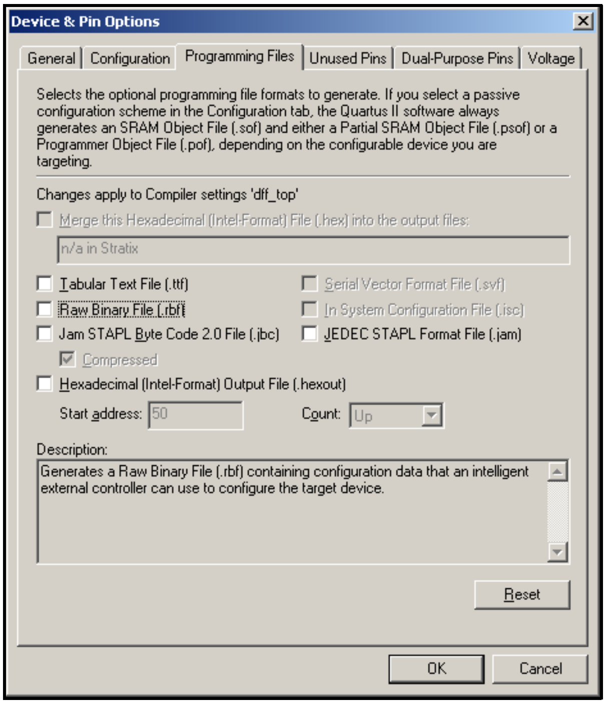
Introduction

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)

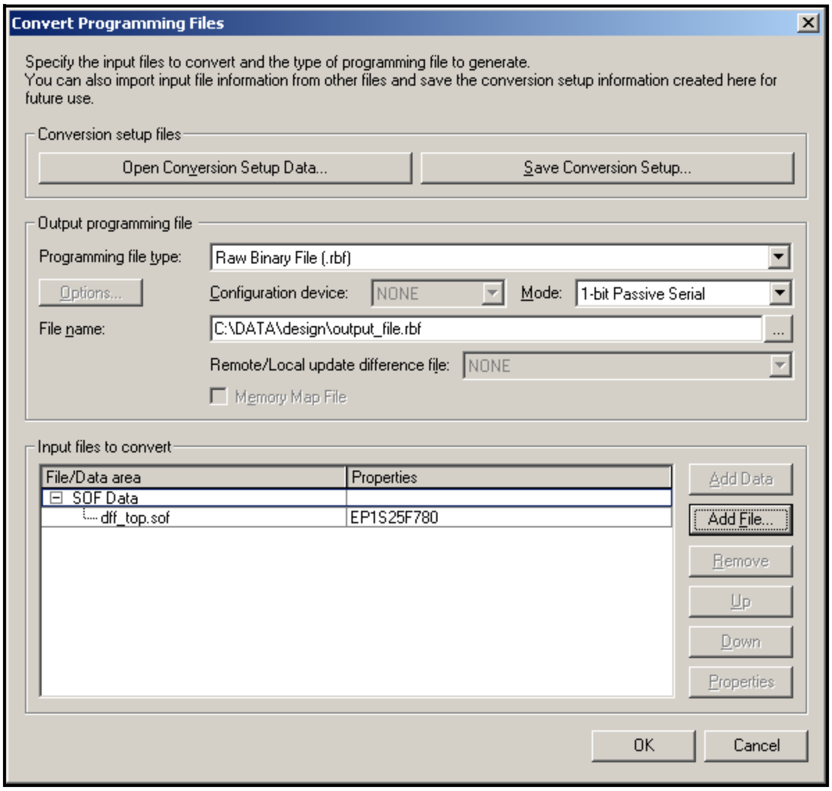
Start address: 50 Count: Up

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.

Conversion setup files

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: Raw Binary File (.rbf)

Options... | 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

Input files to convert

File/Data area	Properties
⊟ SOF Data	
dff_top.sof	EP1S25F780

Add Data | Add File... | Remove | Up | Down | Properties

OK | Cancel

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:

Revision History

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

Chapter	Date/Version	Changes Made
8	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.
9	July 2004, v2.0	Added Stratix II and Cyclone II device information throughout chapter.
	September 2003, v1.0	Initial Release.
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

CF52008-2.0

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" 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

CF52009-2.0

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.

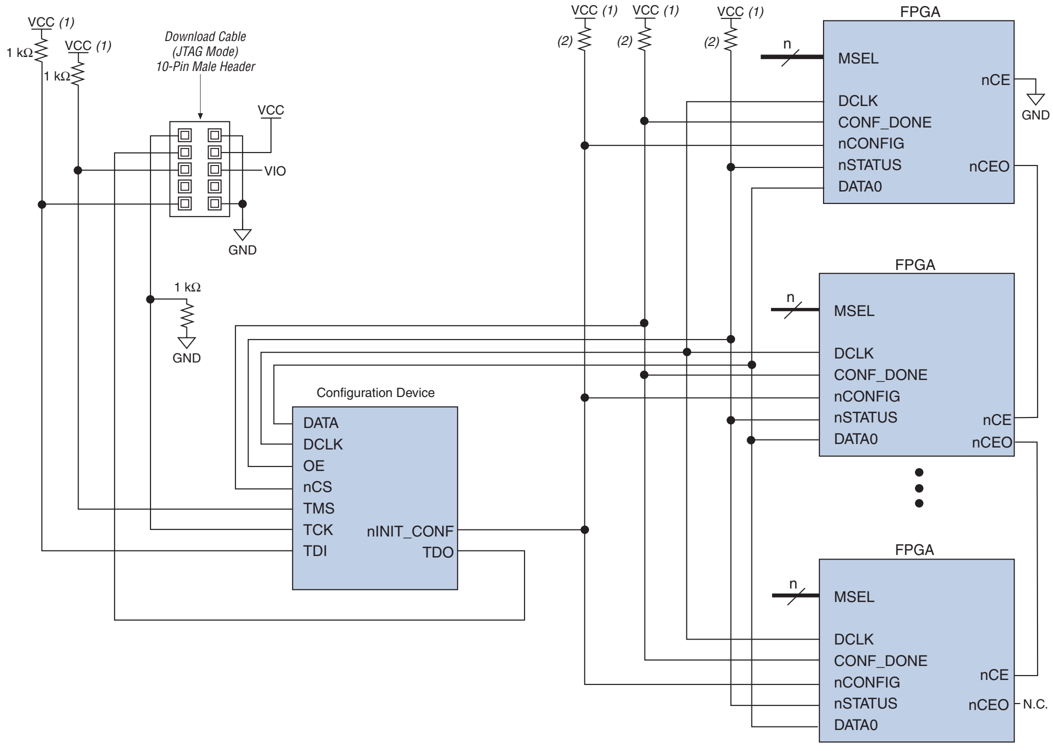
In this chapter, the generic term "download cable" includes the Altera USB Blaster universal serial bus (USB) port download cable, MasterBlaster™ serial/USB communications cable, EthernetBlaster, ByteBlaster™ II parallel port download cable, and the ByteBlasterMV™ parallel port download cable. In this 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:

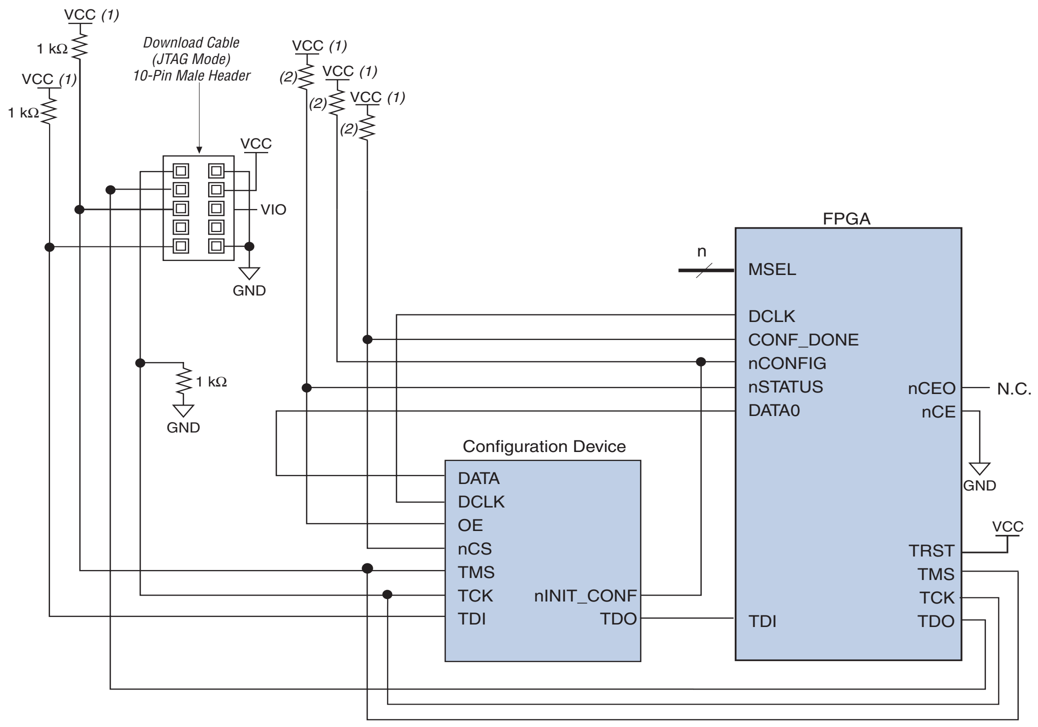
(1) V_{CC} should be connected to the same supply voltage as the configuration device. For APEX 20KE devices, `nCONFIG` should be pulled up to V_{CCINT}.

(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



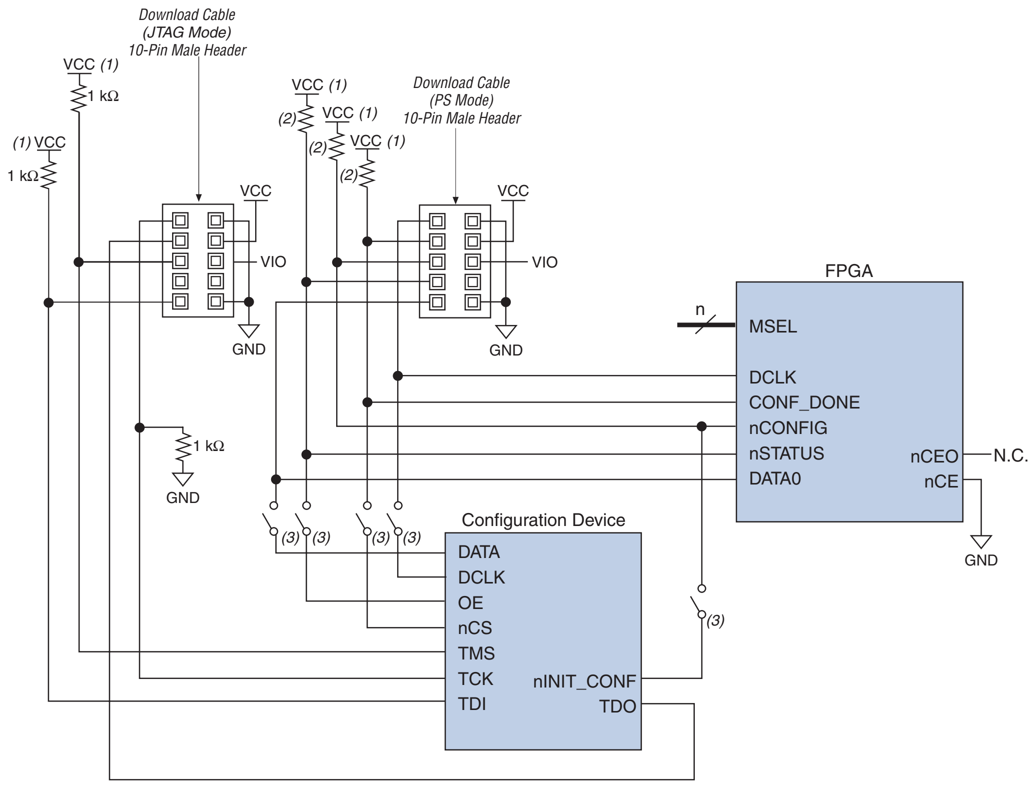
Notes to Figure 9–2:

(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.

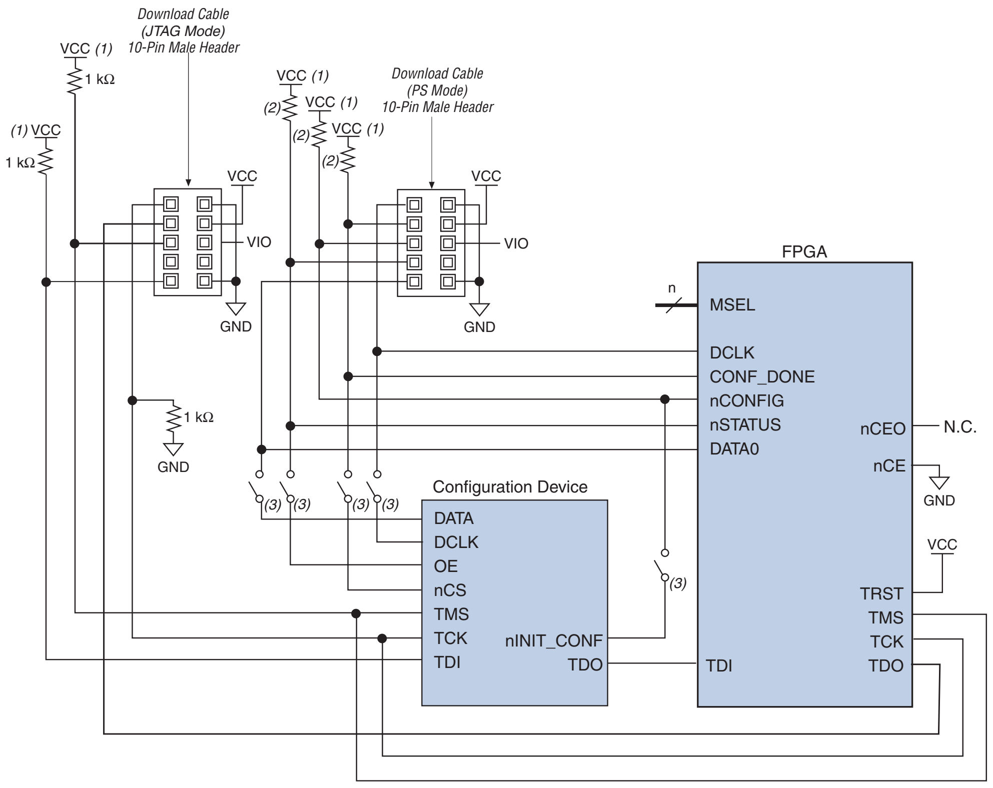
(3) To configure the FPGA with a download cable, you should either remove the configuration device from its socket or place a switch on the five common signals between the download cable and the configuration device.

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.

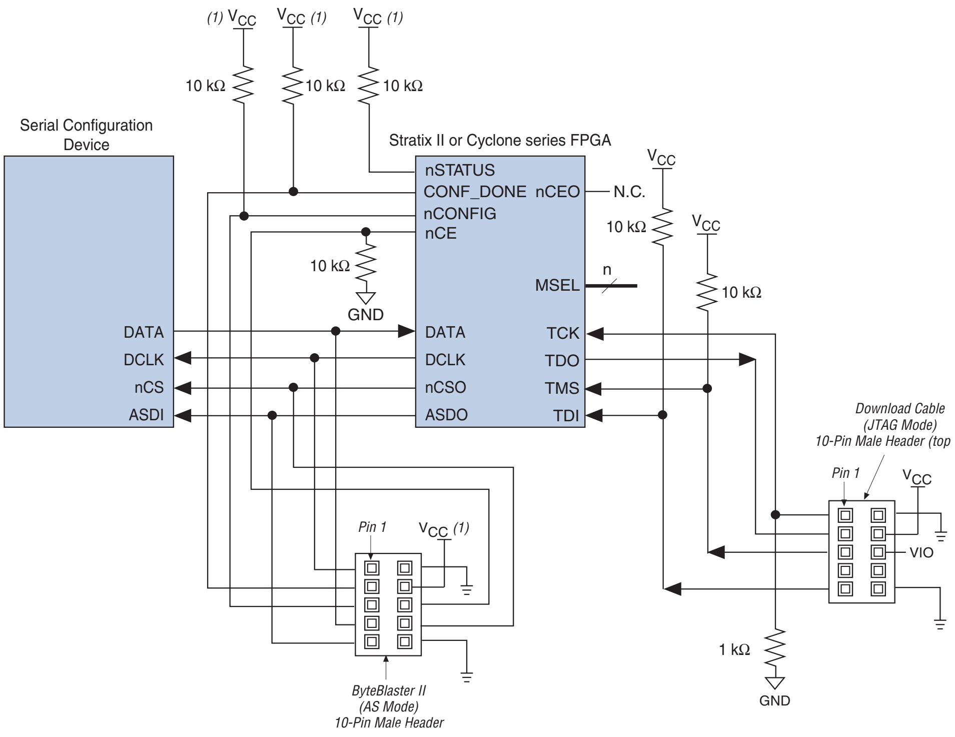
(3) To configure the FPGA with a download cable, you should either remove the configuration device from its socket or place a switch on the five common signals between the download cable and the configuration device.

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

CF52010-2.0

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

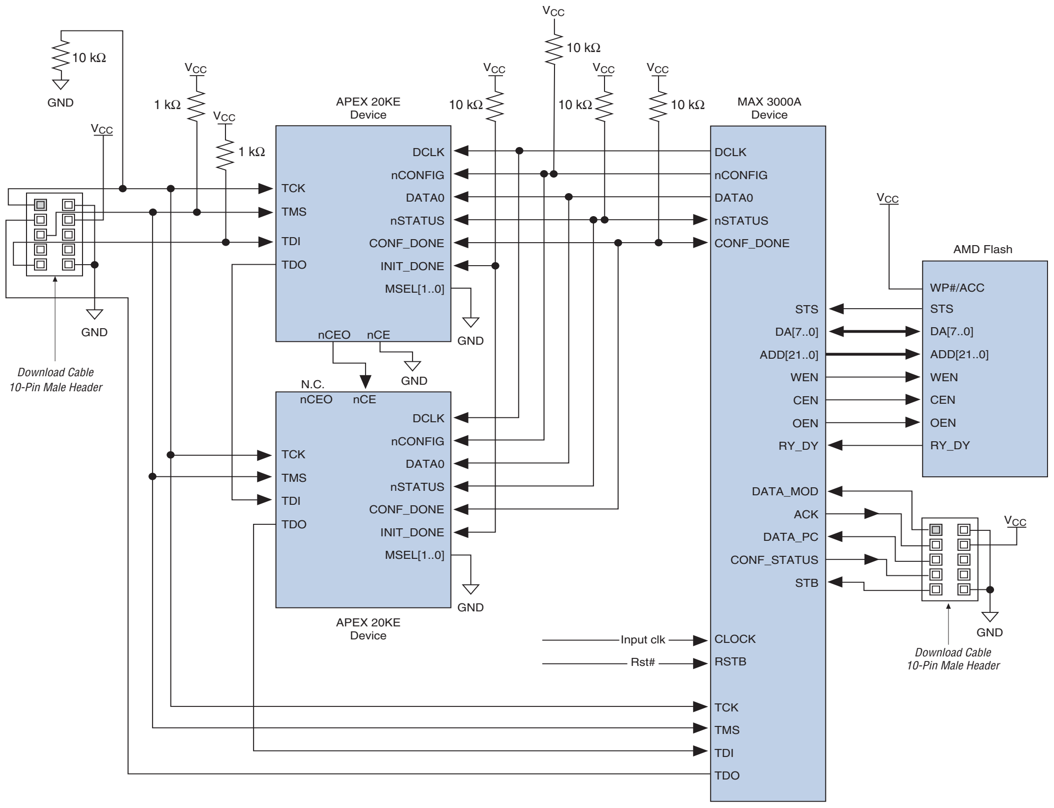
The flash memory controller can interface with a PC or microprocessor to receive configuration data via a parallel port (Figure 10–1). The controller generates a programming command sequence to program the flash memory and extract configuration data to configure FPGAs.

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



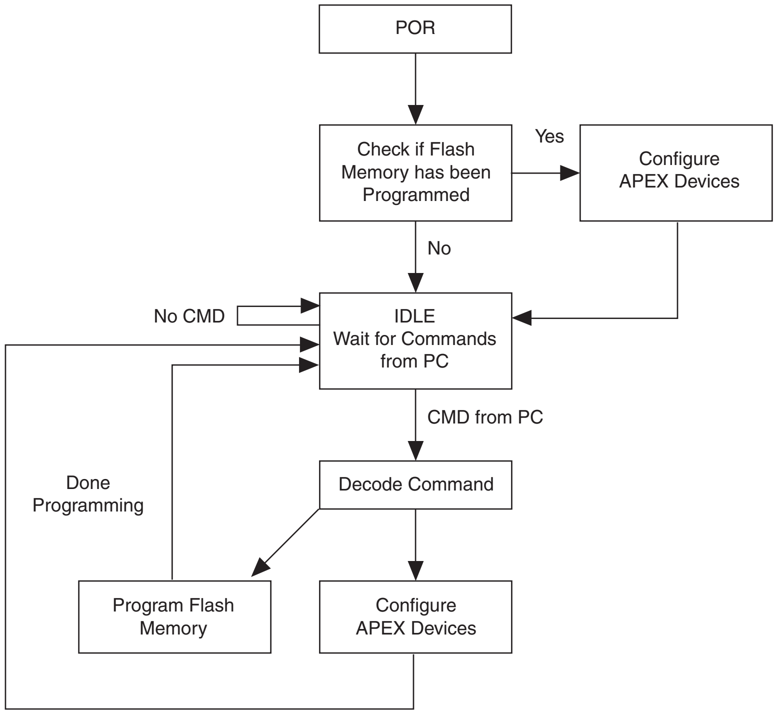
Flash Memory Controller Design Specification

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

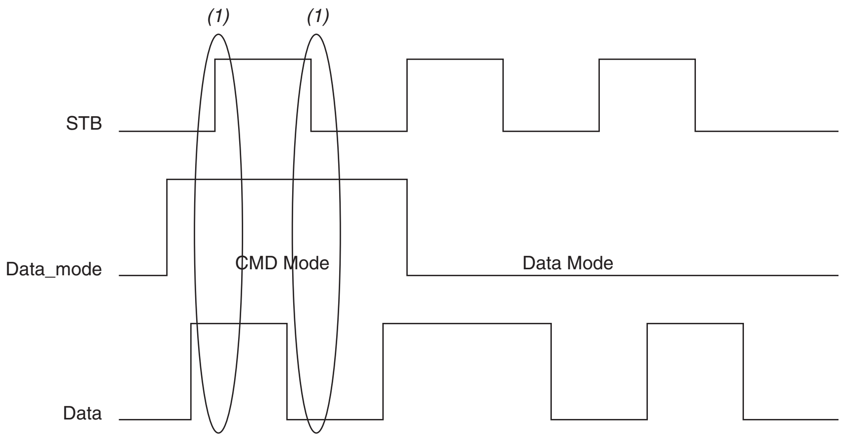
The PC or microprocessor uses the parallel port to interface with the controller. There are two types of signals involved in this connection (see Figure 10–3), a 3-bit input signal from the PC or microprocessor to the controller, and a 2-bit output signal from the controller to the PC or microprocessor.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

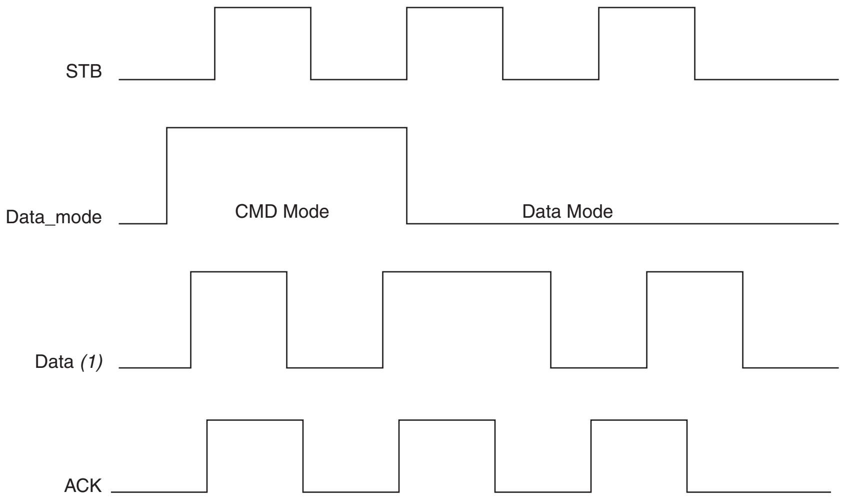
Figure 10–3. Getting Data from PC or Microprocessor



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.

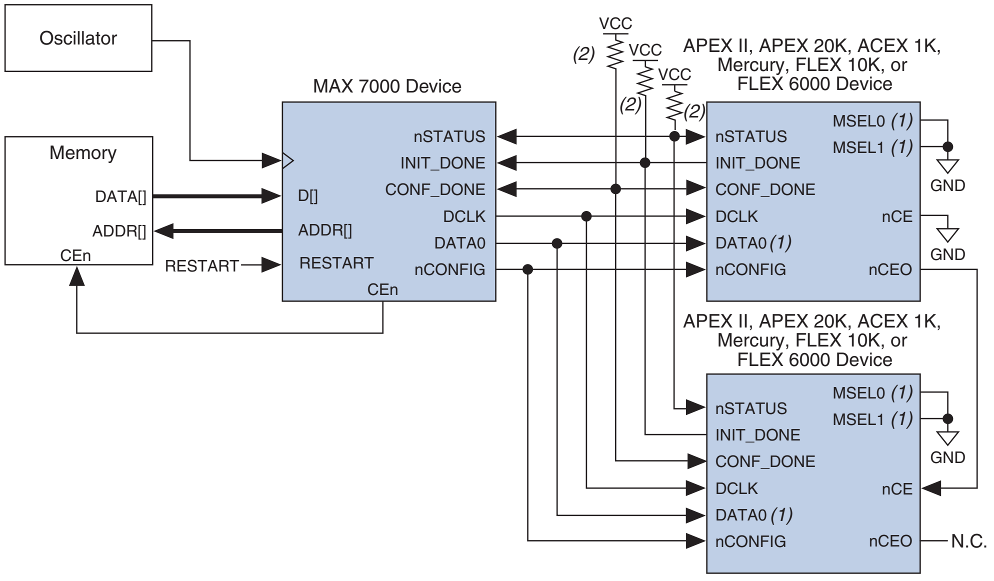
While programming the flash memory, the controller must check if a command (data_mode=1) has been received or not. A command indicates the end of data from the PC or microprocessor, and the controller will exit the Program_Flash_memory state and go into idle mode.

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



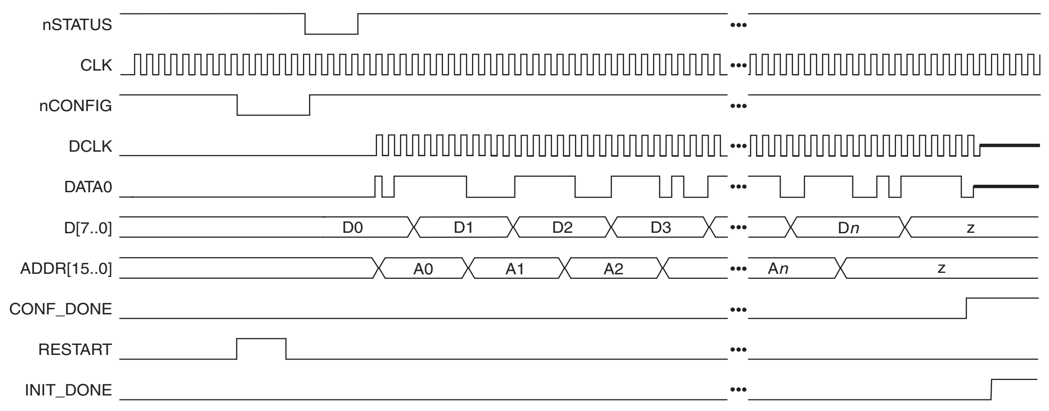
Notes to Figure 10–5:

(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).

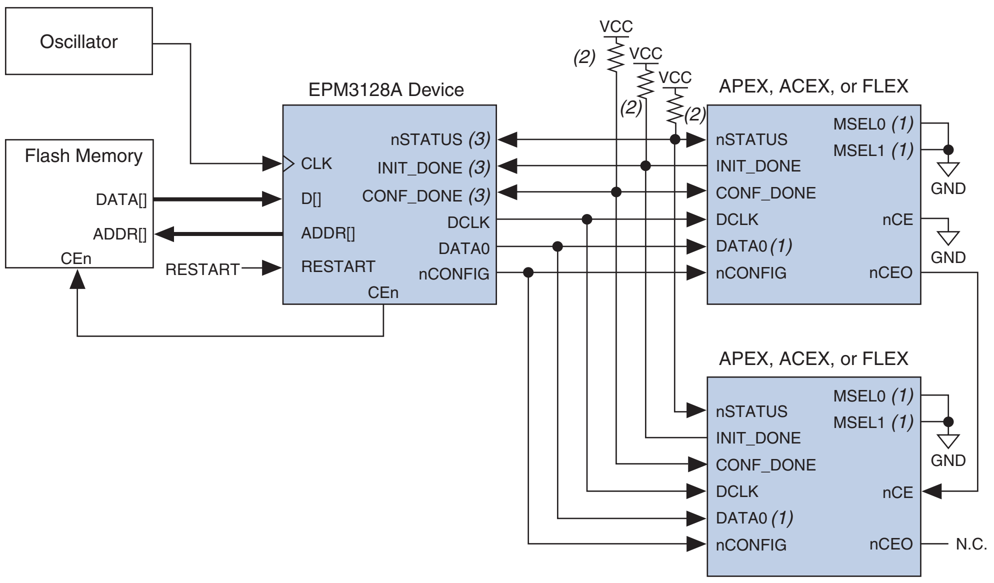
Flash memory is a type of nonvolatile memory that can be used as a data storage device. Flash memory can be erased and reprogrammed in units of memory called blocks.

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.

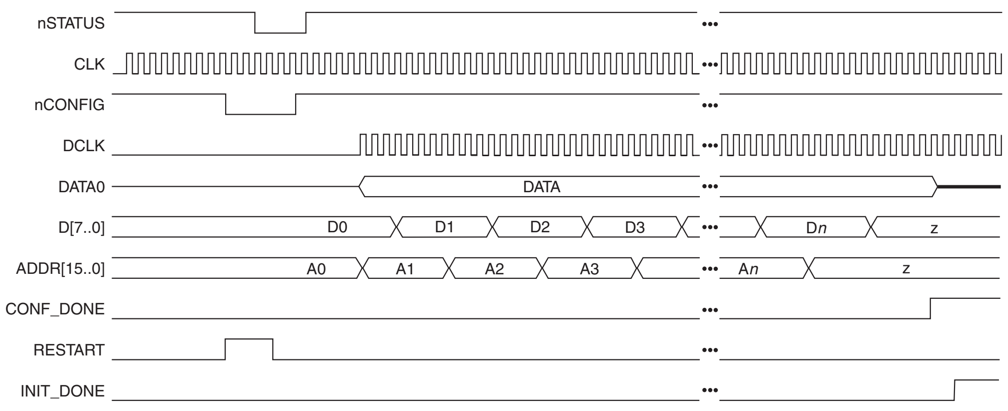
A VHDL design file called **MAXconfig**, shown in the "Configuration Design File" section, allows an EPM3128A device to control the configuration process. The **MAXconfig** design configures the FPGA using the configuration data stored in the attached flash memory. The **MAXconfig** design contains a 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	July 2004, v2.0	Renamed debugging tool to troubleshooter on page 11–6.
	September 2003, v1.0	Initial Release.

11. Debugging Configuration Problems

CF52011-2.0

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**.