

This section describes configuration and remote system upgrade. This section also provides configuration information for all of the supported configuration schemes for Stratix® devices. These configuration schemes use either a microprocessor, configuration device, or download cable. There is detailed information on how to design with Altera® enhanced configuration devices which includes information on how to manage multiple configuration files and access the on-chip FLASH memory space. The last chapter shows you how to perform remote and local upgrades for your designs.

This section contains the following chapters:

- [Chapter 11, Configuring Stratix & Stratix GX Devices](#)
- [Chapter 12, Remote System Configuration with Stratix & Stratix GX Devices](#)



For information on Altera enhanced configuration devices, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* chapter in the *Configuration Handbook, Volume 2*.

Revision History

The table below shows the revision history for [Chapters 11](#) through [12](#).

Chapter	Date/Version	Changes Made
11	July 2005, v3.2	<ul style="list-style-type: none"> Updated “PORSEL Pins” and “nIO_PULLUP Pins” sections. Updated “FPP Configuration Using an Enhanced Configuration Device” section. Updated “PPA Configuration” section.
	September 2004, v3.1	<ul style="list-style-type: none"> Corrected spelling error.
	April 2004, v3.0	<ul style="list-style-type: none"> 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 11–3. Two vertical V_{CC} lines removed in Figures 11–12 to 11–14. Three paragraphs added regarding the CONF_DONE and INIT_DONE pins on page 13-18. Value in Note 1 changed in Tables 11–8 and 11–9. Deleted reference to AS in Table 11–15 because Stratix does not support AS mode. Text added before callout of Figure 11–7.
	July 2003, v2.0	<ul style="list-style-type: none"> Updated Remote/local update PPA typical use description on page 11-1. Updated VCCSEL Pins section on page 11-3. Updated figures to use 10k resistors throughout for configuration control signals. Updated text on page 11-23 to tell how to connect a microprocessor to nSTATUS. Figure 11–19, Note 3. Updated Table 11–12. Added Note 6 to Figure 11–21 and the text below the figure describing the nCE pin. Updated definitions for CLKUSR, and JTAG pins in Table 11–16.
12	September 2004, v3.1	<ul style="list-style-type: none"> Editorial corrections.
	April 2004, v3.0	<ul style="list-style-type: none"> The input file in Figure 12–22 was changed to remote_update_initial_pgm.pdf. Title in Figure 12–23 was changed from Local... to Remote Update Partial Programming File Generation. Rearranged the “Quartus II Software Support” section.
	July 2003, v2.0	<ul style="list-style-type: none"> Added altremote_update Megafunction section on pages 12-18 to 12-21.

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 11–1](#).

Table 11–1. 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

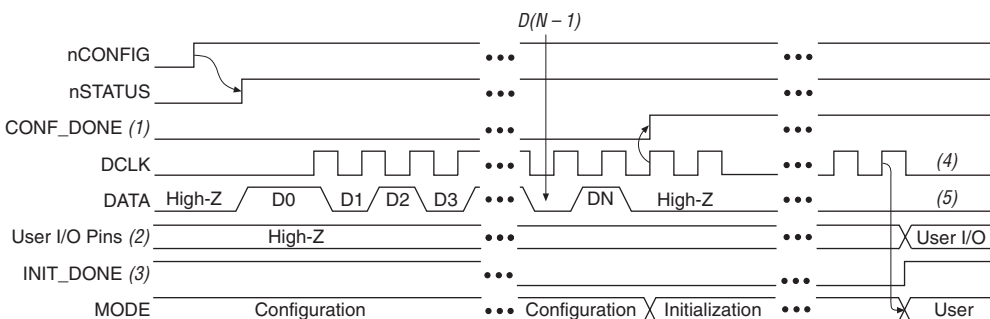


For more information on setting device configuration options or generating configuration files, see the *Software Setting* chapter in *Volume 2 of the Configuration Handbook*.

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 11–1](#) shows the state of the device during the configuration, initialization, and user mode.

Figure 11–1. Stratix & Stratix GX Configuration Cycle



Notes to [Figure 11–1](#):

- (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 11–2](#).

Table 11–2. 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 11–2](#):

- (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 `VCCIO` 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 `VCCIO` of the I/O bank they reside in or ground.

VCCSEL Pins

You can configure Stratix and Stratix GX devices using the 3.3-, 2.5-, 1.8-, or 1.5-V LVTTTL 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-k Ω 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-k Ω 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 11–3 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.

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-k Ω pull-down resistor on PORSEL. Therefore if you are using a pull-up resistor to pull up this signal, you need to use a 1-k Ω 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 11–4](#)).

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 VCC 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-k Ω pull-down resistor on nIO_PULLUP. Therefore, if you are using a pull-up resistor to pull up this signal, you need to use a 1-k Ω 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 11–5 and 11–6 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 11–5. 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 11–6. 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 11–5 and 11–6 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 11-3](#)).

In the configuration device scheme, `nCONFIG` is usually tied to `VCC` (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. The `CONF_DONE` pin must have an external 10-k Ω pull-up resistor in order for the device to initialize.

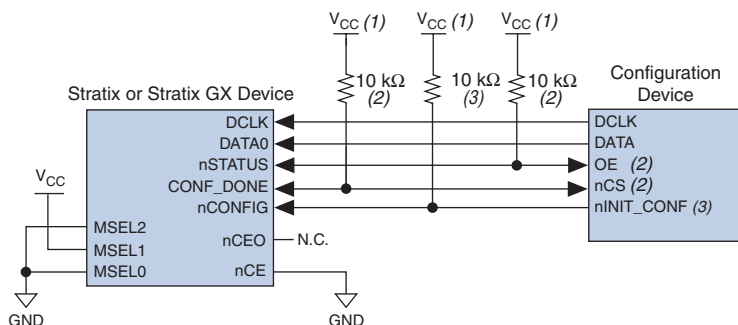
If an error occurs during configuration, the target device drives its `nSTATUS` pin low, resetting itself internally and resetting the configuration device. If the **Auto-Restart Configuration on Frame Error** option—available in the Quartus II **Global Device Options** dialog box (Assign menu)—is turned 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 11–2 shows how to configure one Stratix or Stratix GX device with one configuration device.

Figure 11–2. Single Device Configuration Circuit

Notes to Figure 11–2:

- (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. 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.

Figure 11–3 shows how to configure multiple Stratix and Stratix GX devices with multiple EPC2 or EPC1 configuration devices.

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`.

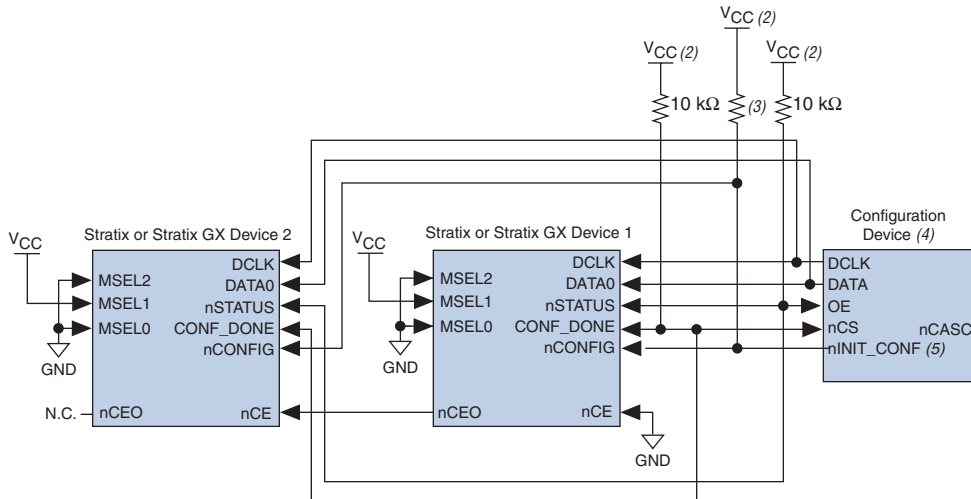
You can also cascade several EPC2/EPC1 configuration devices to configure multiple Stratix and Stratix GX devices. When all data from the first configuration device is sent, it drives `nCASC` low, which in turn drives `nCS` on the subsequent configuration device. Because a configuration device requires 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 11–4 shows an example of configuring multiple Stratix and Stratix GX devices using a configuration device.

Figure 11–4. Configuring Multiple Stratix & Stratix GX Devices with A Single Configuration Device *Note (1)***Notes to Figure 11–4:**

- (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 11–7 shows the status of the device DATA pins during and after configuration.

Table 11–7. 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 11–7:

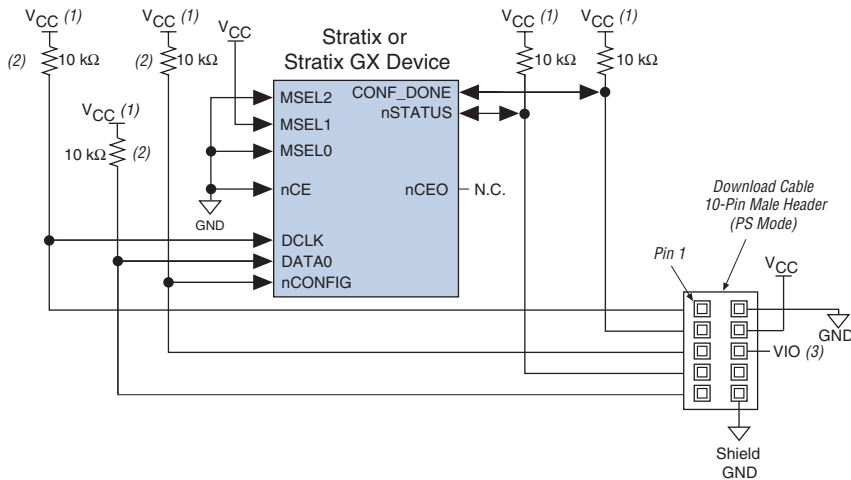
- (1) The status shown is for configuration with a configuration device.
- (2) The function of these pins depends upon the settings specified in the Quartus II software using the **Device & Pin Option** dialog box (see the *Software Settings* section in the *Configuration Handbook, Volume 2*, and the 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. The CONF_DONE pin must have an external 10-kΩ pull-up resistor in order for the device to initialize.

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 effect 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 11–5 shows PS configuration for the Stratix or Stratix GX device using a MasterBlaster, USB-Blaster, ByteBlaster II or ByteBlasterMV cable.

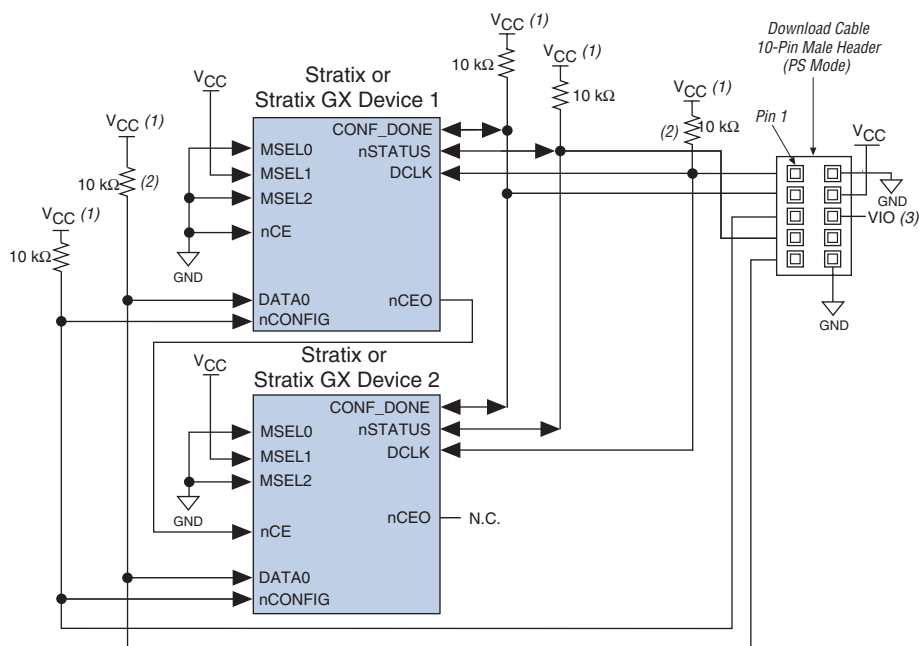
Figure 11–5. PS Configuration Circuit with a Download Cable**Notes to Figure 11–5:**

- (1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (V_{IO} 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.

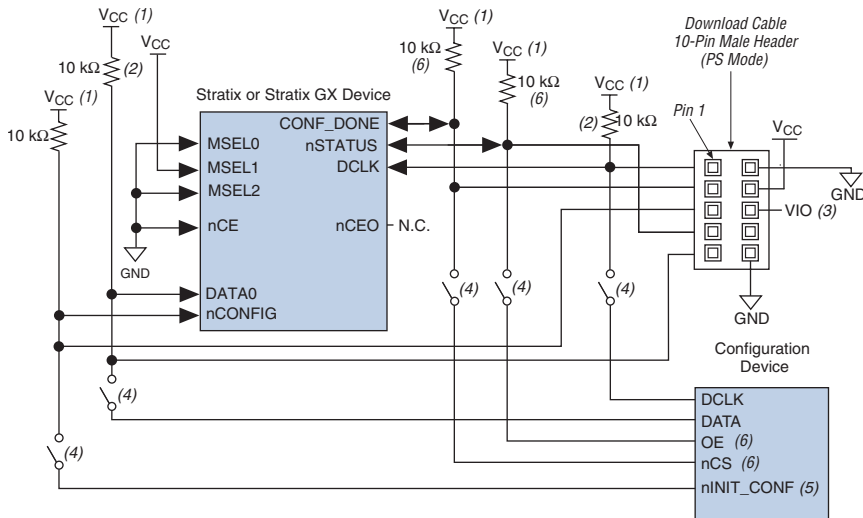
Because all CONF_DONE pins are tied together, all devices in the chain initialize and enter user mode at the same time. In addition, because the nSTATUS pins are tied together, the entire chain halts configuration if any device detects an error. In this situation, the Quartus II software must restart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

Figure 11–6 shows how to configure multiple Stratix and Stratix GX devices with a MasterBlaster or ByteBlasterMV cable.

Figure 11–6. Multi-Device PS Configuration with a Download Cable

Notes to Figure 11–6:

- (1) You should connect the pull-up resistor to the same supply voltage as the MasterBlaster (V_{IO} 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 11–7 shows a combination of a configuration device and a download cable to configure a Stratix or Stratix GX device.

Figure 11–7. Configuring with a Combined PS & Configuration Device Scheme

Notes to Figure 11–7:

- (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 k Ω 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 or Stratix GX device, the `CONF_DONE` pin goes high to show successful configuration and the start of initialization. The `CONF_DONE` pin must have an external 10-k Ω pull-up resistor in order for the device to initialize. 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 `tCD2UM` (see [Table 11-8](#)) after the `CONF_DONE` signal goes high to ensure the device has been initialized properly and that it has entered user mode.

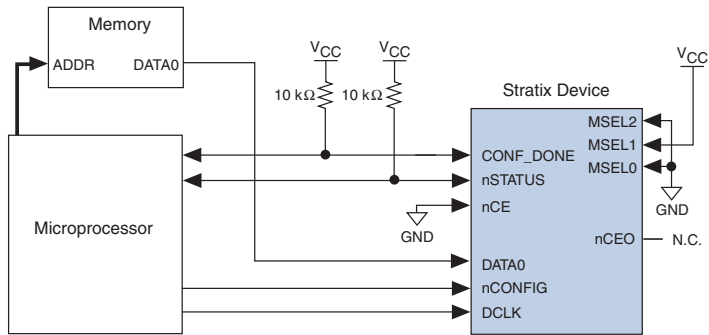
During configuration and initialization, and before the device enters user mode, the microprocessor must not drive the CONF_DONE signal low.



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 μ s).

Figure 11–8 shows the circuit for PS configuration with a microprocessor.

Figure 11–8. PS Configuration Circuit with Microprocessor



PS Configuration Timing

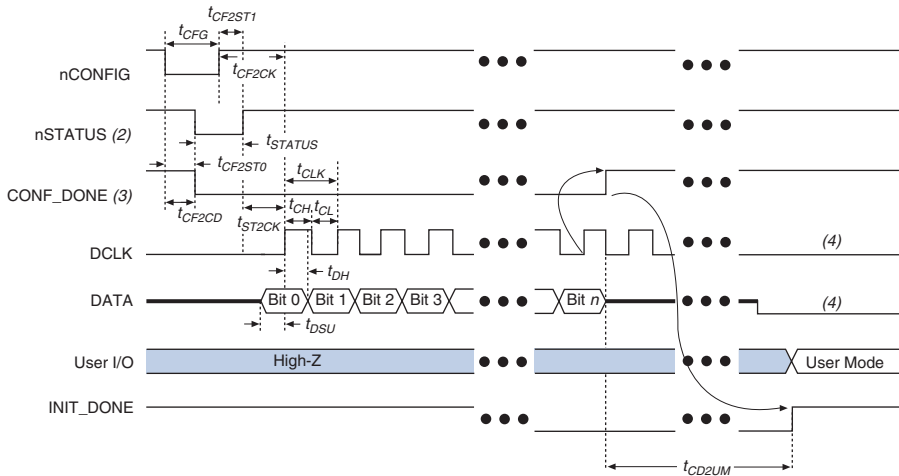
Figure 11–9 shows the PS configuration timing waveform for Stratix and Stratix GX devices. Table 11–8 shows the PS timing parameters for Stratix and Stratix GX devices.

Table 11–8. 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)	μ s
t_{CFG}	nCONFIG low pulse width	40		μ s
t_{STATUS}	nSTATUS low pulse width	10	40 (2)	μ s
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	40		μ s
t_{ST2CK}	nSTATUS high to first rising edge on DCLK	1		μ s
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	μ s

Notes to Table 11–8:

- (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 CLKUSER, 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 11–9. PS Timing Waveform for Stratix & Stratix GX Devices Note (1)**Notes to Figure 11–9:**

- (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.

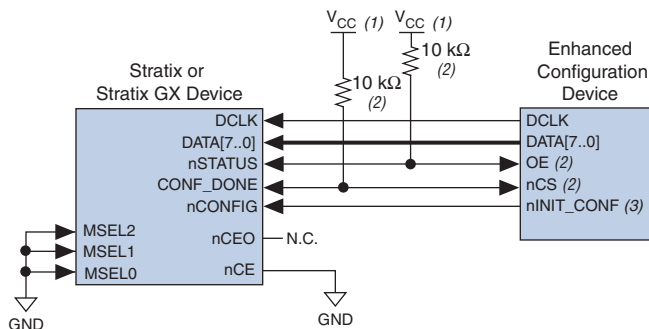
This section discusses the following schemes for FPP configuration in Stratix and Stratix GX devices:

- 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 11–10](#).

Figure 11–10. FPP Configuration Using Enhanced Configuration Devices



Notes to [Figure 11–10](#):

- (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.



`CONF_DONE` goes high one byte early in parallel synchronous (FPP) and asynchronous (PPA) modes using a microprocessor with `.rbf`, `.hex`, and `.ttf` file formats. This does not apply to FPP mode for enhanced configuration devices using `.pof` file format. This also does not apply to serial modes.

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 11-11 on page 11-25](#)).

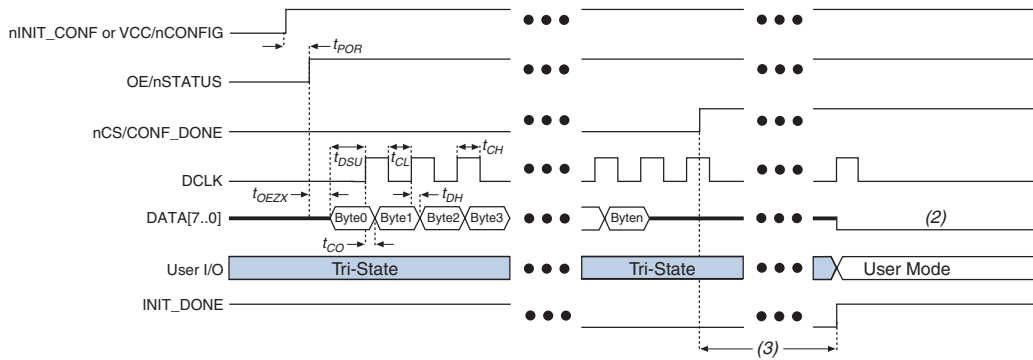
Do not drive CONF_DONE low after device configuration to delay initialization. Instead, use the **Enable User-Supplied Start-Up Clock (CLKUSR)** option in the **Device & Pin Options** dialog box. You can use this option to synchronize the initialization of multiple devices that are not in the same configuration 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 11-11 shows the FPP configuration with a configuration device timing waveform for Stratix and Stratix GX devices.

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



Notes to Figure 11–11:

- (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 goes high to show successful configuration and the start of initialization. The CONF_DONE pin must have an external 10-kΩ pull-up resistor in order for the device to initialize. 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 11-9](#)) 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.

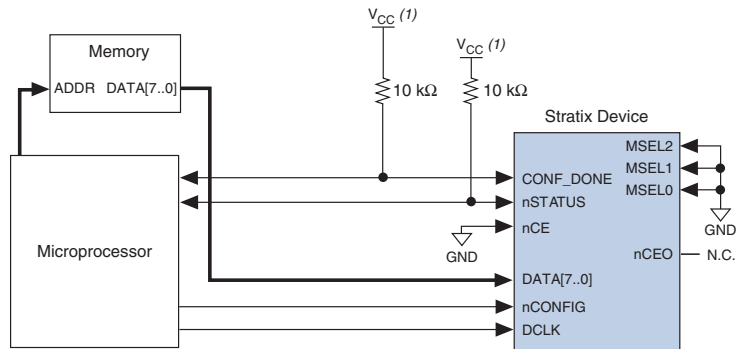


If the optional `CLKUSR` pin is used and `nCONFIG` is pulled low to restart configuration during device initialization, you need to ensure `CLKUSR` continues toggling during the time `nSTATUS` is low (maximum of 40 μ s).

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 11-12](#) shows the circuit for Stratix and Stratix GX parallel configuration using a microprocessor.

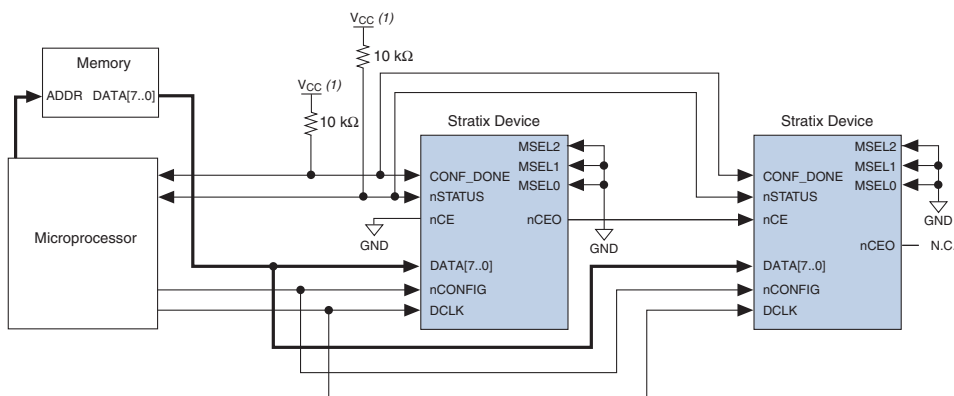
Figure 11–12. Parallel Configuration Using a Microprocessor**Note to Figure 11–12:**

- (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 11–13](#) shows multi-device configuration using a microprocessor. [Figure 11–14](#) 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.

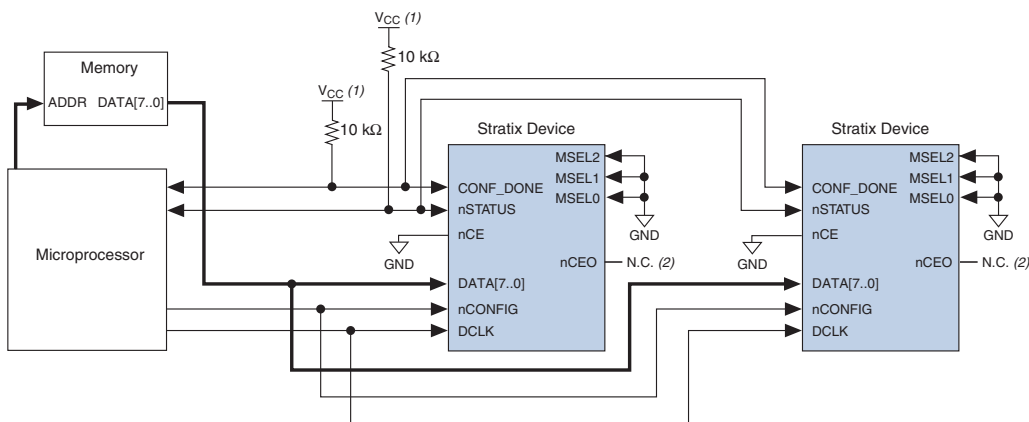
Figure 11–13. Parallel Data Transfer in Serial Configuration with a Microprocessor



Note to Figure 11–13:

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

Figure 11–14. Multiple Device Parallel Configuration with the Same Data Using a Microprocessor



Notes to Figure 11–14:

- (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.

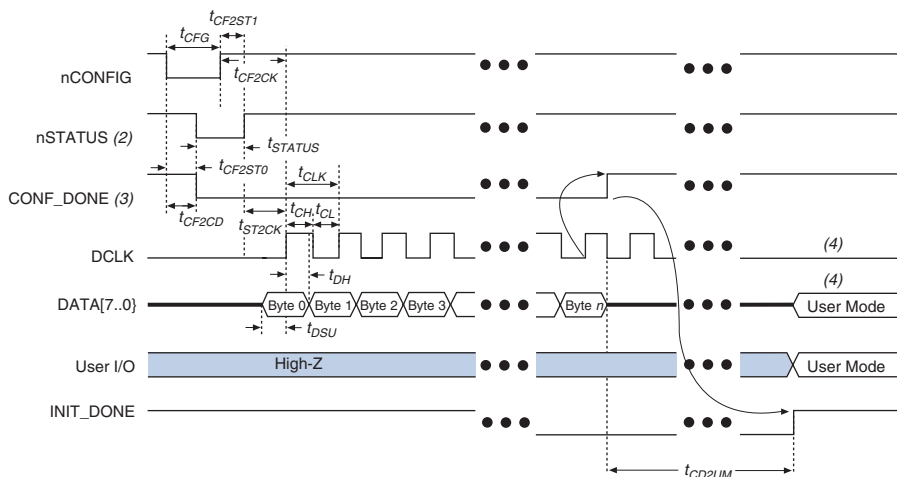


For more information on configuring multiple Altera devices in the same configuration chain, see the *Configuring Mixed Altera FPGA Chains* chapter in the *Configuration Handbook, Volume 2*.

FPP Configuration Timing

Figure 11-15 shows FPP timing waveforms for configuring a Stratix or Stratix GX device in FPP mode. Table 11-9 shows the FPP timing parameters for Stratix or Stratix GX devices.

Figure 11-15. Timing Waveform for Configuring Devices in FPP Mode Note (1)



Notes to Figure 11-15:

- (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 11-9. 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

Table 11–9. FPP Timing Parameters for Stratix & Stratix GX Devices (Part 2 of 2)

Symbol	Parameter	Min	Max	Units
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
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 11–9:

- (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

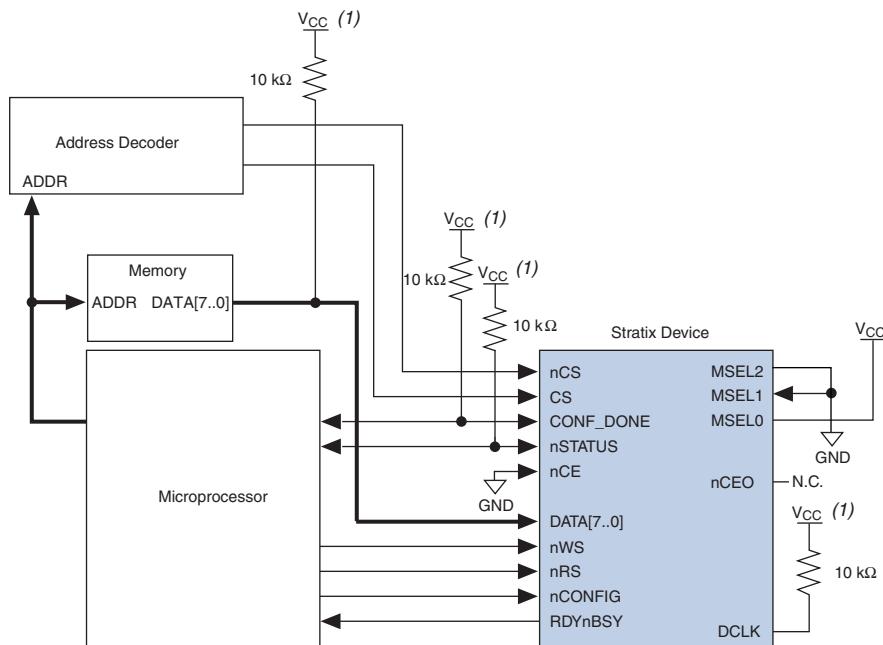
In PPA schemes, a microprocessor drives data to the Stratix or Stratix GX device through a download cable. When using a PPA scheme, use a 1-k Ω pull-up resistor to pull the DCLK pin high to prevent unused configuration pins from floating.

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. The CONF_DONE pin must have an external 10-k Ω pull-up resistor in order for the device to initialize. 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 11–16 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 11–16. PPA Configuration Circuit



Note to Figure 11–16:

(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 11–10 on page 11–36. 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 $t_{BUSY}(\text{max}) + t_{RDY2WS} + t_{W2SB}$ 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 $nSTATUS$ pin to detect errors and the $CONF_DONE$ pin to determine when programming completes ($CONF_DONE$ goes high one byte early in parallel mode). 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 11-10](#)) after the $CONF_DONE$ signal goes high to ensure the device has been initialized properly and that it has entered user mode.

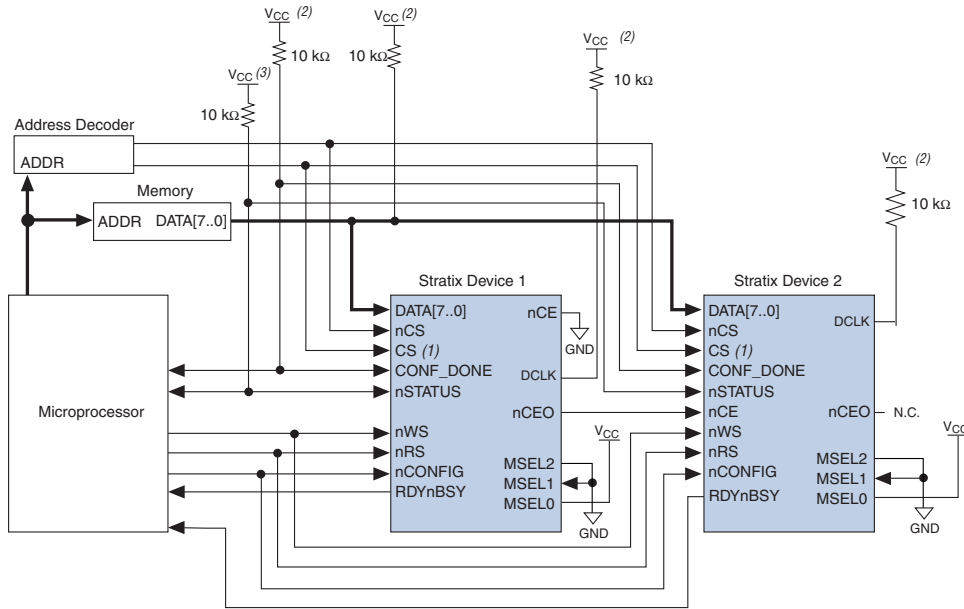
During configuration and initialization, and before the device enters user mode, the microprocessor must not drive the $CONF_DONE$ signal low.



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, $nCE0$ 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 11–17.

Figure 11–17. PPA Multi-Device Configuration Circuit



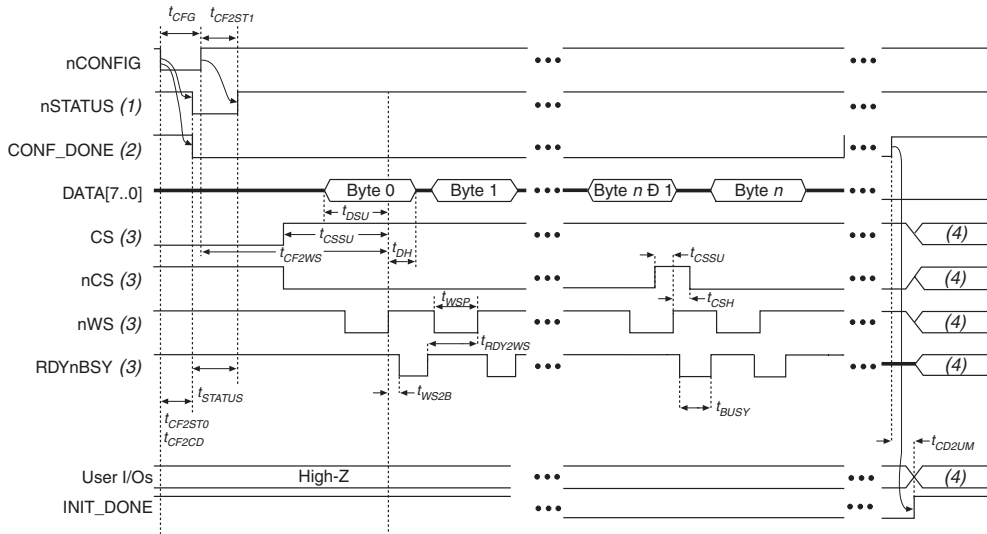
Notes to Figure 11–17:

- (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 11–18 shows the Stratix and Stratix GX device timing waveforms for PPA configuration.

Figure 11–18. PPA Timing Waveforms for Stratix & Stratix GX Devices

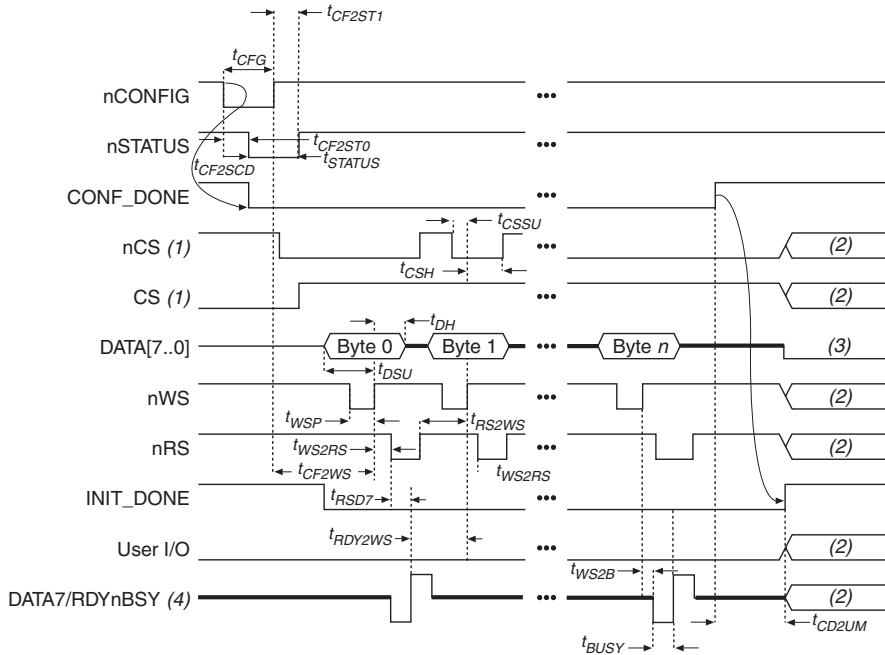


Notes to Figure 11–18:

- (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 11–19 shows the Stratix and Stratix GX timing waveforms when using strobed nRS and nWS signals.

Figure 11–19. PPA Timing Waveforms Using Strobed nRS & nWS Signals



Notes to Figure 11–19:

- (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 these 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 11–10 defines the Stratix and Stratix GX timing parameters for PPA configuration

Symbol	Parameter	Min	Max	Units
t_{CF2WS}	nCONFIG high to first rising edge on nWS	40		μ s
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		μ s
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	μ s
t_{STATUS}	nSTATUS low pulse width	10	40 (2)	μ s
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)	μ s

Notes to Table 11–10:

- (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*.

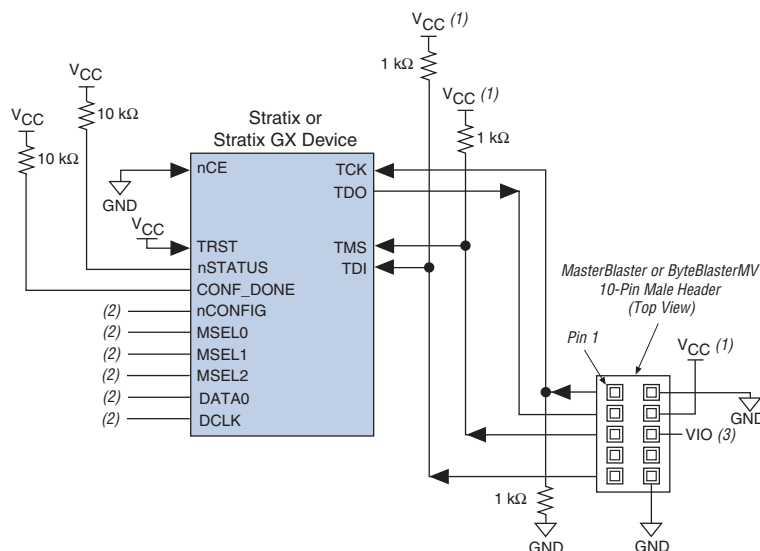
A device operating in JTAG mode uses four required pins, TDI, TDO, TMS, and TCK, and one optional pin, TRST. The four JTAG input pins (TDI, TMS, TCK and TRST) have weak, internal pull-up resistors, whose values range from 20 to 40 k Ω . All other pins are tri-stated during JTAG configuration. Do not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

Table 11–11. 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 11–20.

Figure 11–20. JTAG Configuration of a Single Device



Notes to Figure 11–20:

- (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 11–12](#) shows how you should connect these pins during JTAG configuration.

Table 11–12. Dedicated Configuration Pin Connections During JTAG Configuration

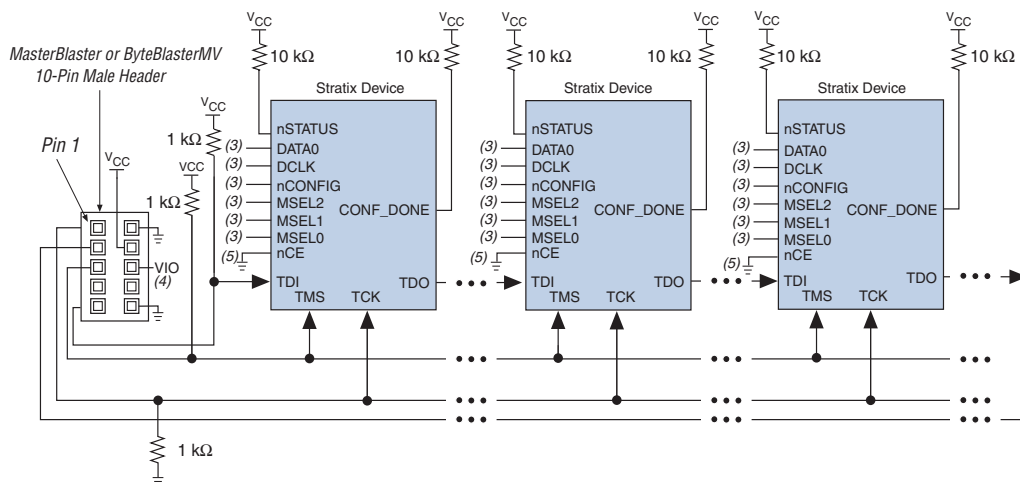
Signal	Description
nCE	On all Stratix and Stratix GX devices in the chain, nCE should be driven low by connecting it to ground, pulling it low via a resistor, or driving it by some control circuitry. For devices that are also in multi-device PS, FPP or PPA configuration chains, the nCE pins should be 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 11–21 shows multi-device JTAG configuration.

Figure 11–21. Multi-Device JTAG Configuration Notes (1), (2)




Notes to Figure 11–21:

- (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 11–11 on page 11–37.
- (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_{I/O} is a reference voltage for the MasterBlaster output driver. V_{I/O} 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.

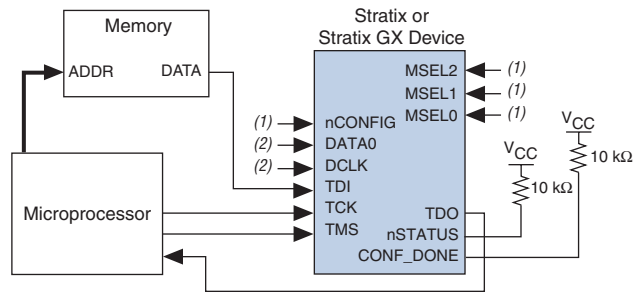
The Quartus II software verifies successful JTAG configuration upon completion. The software checks the state of CONF_DONE through the JTAG port. If CONF_DONE is not in the correct state, the Quartus II software indicates that configuration has failed. If CONF_DONE is in the correct state, the software 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 11–22 shows the JTAG configuration of a Stratix or Stratix GX device with a microprocessor.

Figure 11–22. JTAG Configuration of Stratix & Stratix GX Devices with a Microprocessor



Notes to Figure 11–22:

- (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

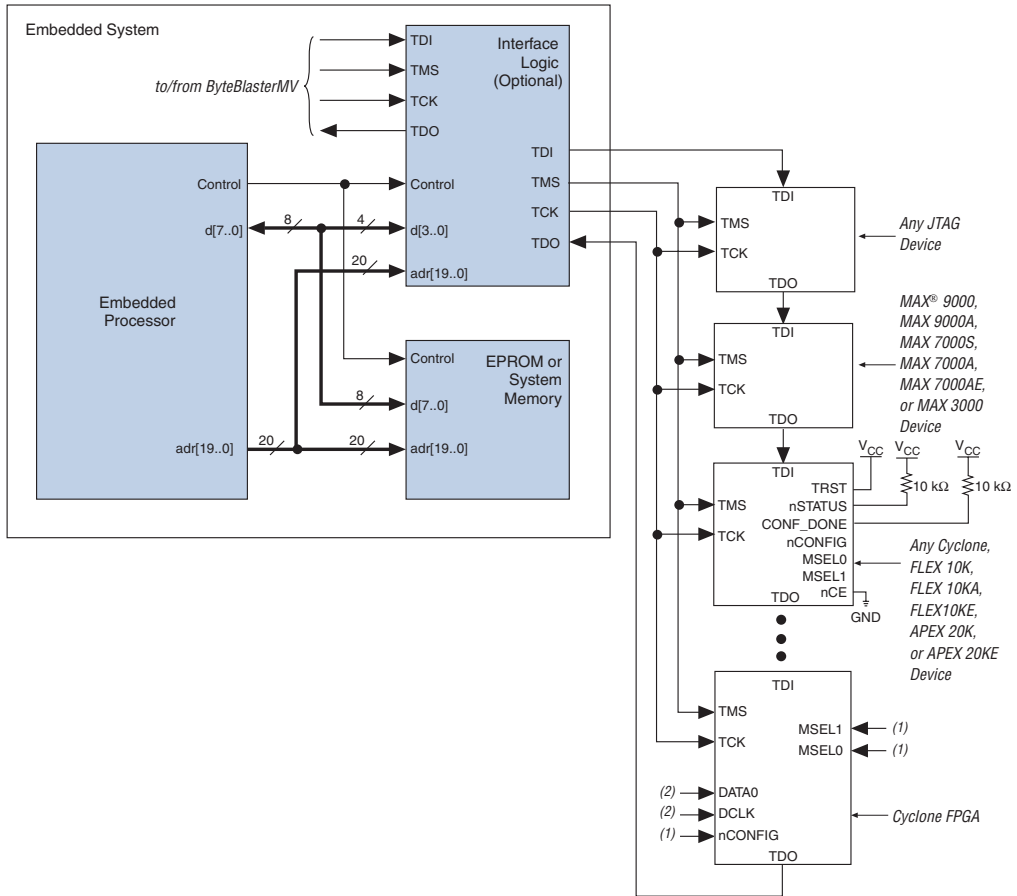
The Jam™ Standard Test and Programming Language (STAPL), JEDEC standard JESD-71, is a standard file format for in-system programmability (ISP) purposes. Jam STAPL supports programming or configuration of programmable devices and testing of electronic systems, using the IEEE 1149.1 JTAG interface. Jam 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 11–23](#) 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 11–23. Embedded System Block Diagram



Notes to Figure 11–23:

- (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 11–24](#) shows the flow of an IEEE Std. 1149.1 TAP controller state machine.

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 11–25 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 11–25. Jam Player Flow Diagram (Part 1 of 2)

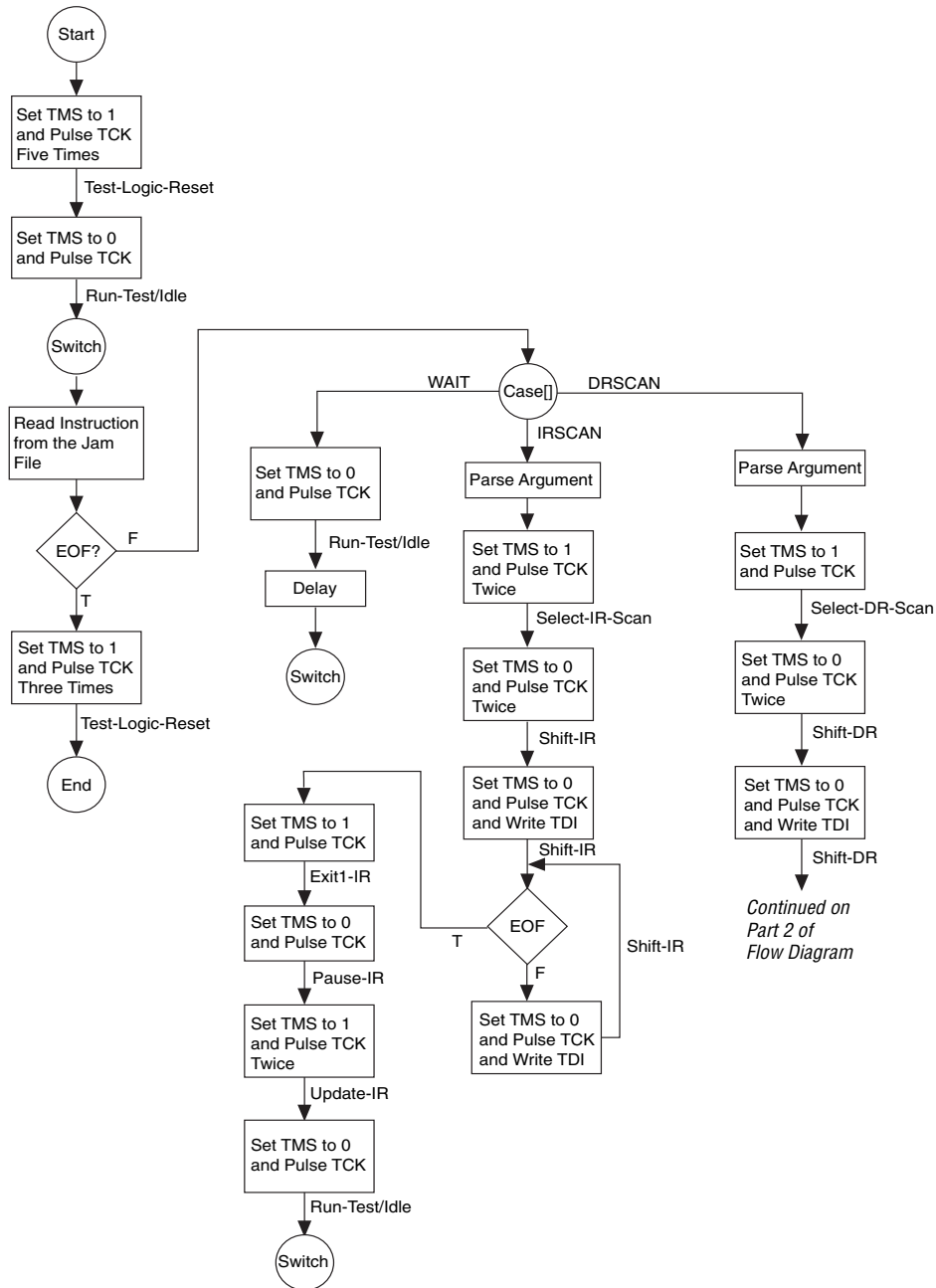
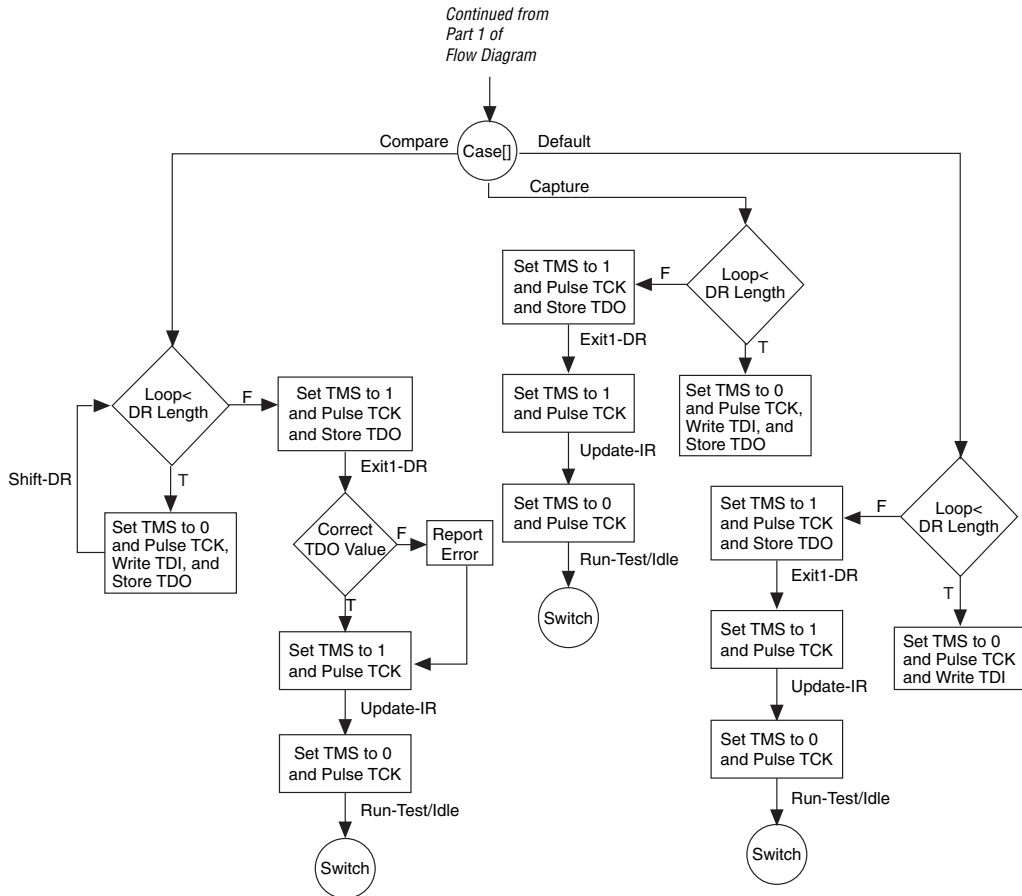


Figure 11–26. Jam Player Flow Diagram (Part 2 of 2)



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 11–13](#). 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 11–13. 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 11–13:

- (1) This instruction name is an optional language extension.

[Table 11–14](#) 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.

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 11–14. 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 11–27 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 11–27. 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 11–15](#) 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 11–15. 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	<p>Dedicated input that selects which input buffer is used on the configuration input pins; nCONFIG, DCLK, RUNLU, nCE, nWS, nRS, CS, nCS and CLKUSR.</p> <p>The VCCSEL input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active.</p> <p>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 “V_{CCSEL} Pins” section for more details.</p>
PORSEL	N/A	All	Input	<p>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.</p> <p>The PORSEL input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active.</p>

Table 11–15. 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	<p>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, RU_nLU, 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.</p> <p>The nIO_PULLUP input buffer is powered by V_{CCINT} and has an internal 2.5 kΩ pull-down resistor that is always active.</p>
MSEL [2 . . 0]	N/A	All	Input	<p>3-bit configuration input that sets the Stratix or Stratix GX device configuration scheme. See Table 11–2 for the appropriate connections.</p> <p>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.</p>
nCONFIG	N/A	All	Input	<p>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.</p> <p>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.</p>

Table 11–15. 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	<p>The device drives nSTATUS low immediately after power-up and releases it after the POR time.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>This pin uses Schmitt trigger input buffers.</p>

Table 11–15. 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	<p>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.</p> <p>Status input. After all data is received and CONF_DONE goes high, the target device initializes and enters user mode. The CONF_DONE pin must have an external 10-kΩ pull-up resistor in order for the device to initialize.</p> <p>Driving CONF_DONE low after configuration and initialization does not affect the configured device.</p> <p>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.</p> <p>This pin uses Schmitt trigger input buffers.</p>
nCE	N/A	All	Input	<p>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.</p> <p>The nCE pin must also be held low for successful JTAG programming of the FPGA. This pin uses Schmitt trigger input buffers.</p>

Table 11–15. 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	<p>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.</p> <p>The voltage levels driven out by this pin are dependent on the V_{CCIO} of the I/O bank it resides in.</p>
DCLK	N/A	Synchronous configuration schemes (PS, FPP)	Input (PS, FPP)	<p>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.</p> <p>In PPA mode, DCLK should be tied high to V_{CC} to prevent this pin from floating.</p> <p>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.</p>
DATA0	I/O	PS, FPP, PPA	Input	<p>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.</p> <p>After configuration, DATA0 is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.</p> <p>After configuration, EPC1 and EPC1441 devices tri-state this pin, while enhanced configuration and EPC2 devices drive this pin high.</p>

Table 11–15. 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	<p>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.</p> <p>In serial configuration schemes, they function as user I/Os during configuration, which means they are tri-stated.</p> <p>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.</p>
DATA7	I/O	PPA	Bidirectional	<p>In the PPA configuration scheme, the DATA7 pin presents the RDY_{nBSY} 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.</p> <p>In serial configuration schemes, it functions as a user I/O during configuration, which means it is tri-stated.</p> <p>After PPA configuration, DATA7 is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.</p>
nWS	I/O	PPA	Input	<p>Write strobe input. A low-to-high transition causes the device to latch a byte of data on the DATA [7 . . 0] pins.</p> <p>In non-PPA schemes, it functions as a user I/O during configuration, which means it is tri-stated.</p> <p>After PPA configuration, nWS is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.</p>

Table 11–15. 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	<p>Read strobe input. A low input directs the device to drive the RDYnBSY signal on the DATA7 pin.</p> <p>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.</p> <p>After PPA configuration, nRS is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.</p>
RDYnBSY	I/O	PPA	Output	<p>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.</p> <p>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.</p> <p>After PPA configuration, RDYnBSY is available as a user I/O and the state of this pin depends on the Dual-Purpose Pin settings.</p>

Table 11–15. Dedicated Configuration Pins on the Stratix or Stratix GX Device (Part 8 of 8)

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nCS/CS	I/O	PPA	Input	<p>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.</p> <p>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.</p> <p>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.</p>
RU _n LU	N/A if using Remote Configuration; I/O if not	Remote Configuration in FPP, PS or PPA	Input	<p>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.</p> <p>When not using remote update or local update configuration modes, this pins is available as general-purpose user I/O pin.</p>
PGM [2 . . 0]	N/A if using Remote Configuration; I/O if not using	Remote Configuration in FPP, PS or PPA	Input	<p>These output pins select one of eight pages in the memory (either flash or enhanced configuration device) when using a remote configuration mode.</p> <p>When not using remote update or local update configuration modes, these pins are available as general-purpose user I/O pins.</p>

Table 11–16 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.

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 $\overline{nCONFIG}$ is low and during the beginning of configuration, the INIT_DONE pin is tri-stated and pulled high due to an external 10-k Ω 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.
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 11–17 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.

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 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.

Introduction

Altera® Stratix® and Stratix GX devices are the first programmable logic devices (PLDs) featuring dedicated support for remote system configuration. Using remote system configuration, a Stratix or Stratix GX device can receive new configuration data from a remote source, update the flash memory content (through enhanced configuration devices or any other storage device), and then reconfigure itself with the new data.

Like all Altera SRAM-based devices, Stratix and Stratix GX devices support standard configuration modes such as passive serial (PS), fast passive parallel (FPP), and passive parallel asynchronous (PPA). You can use the standard configuration modes with remote system configuration.

This chapter discusses remote system configuration of Stratix and Stratix GX devices, and how to interface them with enhanced configuration devices to enable this capability. This document also explains some related remote system configuration topics, such as the watchdog timer, remote system configuration registers, and factory or application configurations files. The Quartus® II software (version 2.1 and later) supports remote system configuration.

Remote Configuration Operation

Remote system configuration has three major parts:

- The Stratix or Stratix GX device receives updated or new data from a remote source over a network (or through any other source that can transfer data). You can implement a Nios™ (16-bit ISA) or Nios® II (32-bit ISA) embedded processor within either a Stratix or Stratix GX device or an external processor to control the read and write functions of configuration files from the remote source to the memory device.
- The new or updated information is stored into the memory device, which can be an enhanced configuration device, industry-standard flash memory device, or any other storage device (see [Figure 12-2](#)).
- The Stratix or Stratix GX device updates itself with the new data from the memory.

[Figure 12-1](#) shows the concept of remote system configuration in Stratix and Stratix GX devices.

Figure 12–1. Remote System Configuration with Stratix & Stratix GX Devices

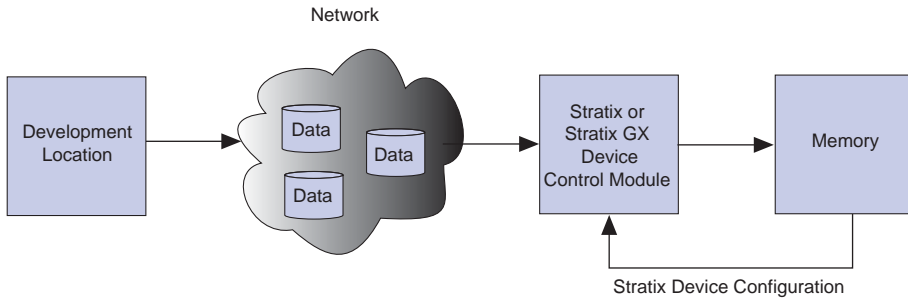
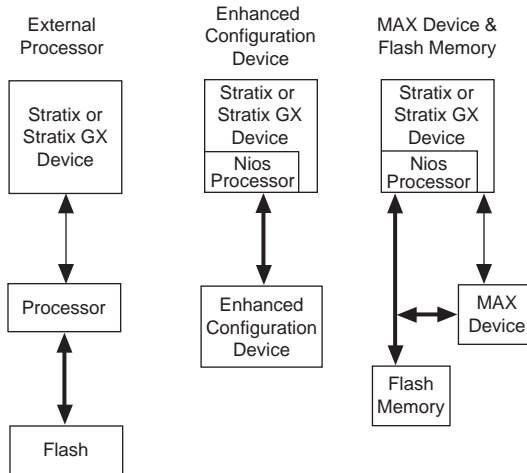


Figure 12–2. Different Options for Remote System Configuration



Remote System Configuration Modes

Stratix and Stratix GX device remote system configuration has two modes: remote configuration mode and local configuration mode. [Table 12-1](#) shows the pin selection settings for each configuration mode.

Table 12-1. Standard, Remote & Local Configuration Options <i>Note (1)</i>				
RUnLU (2)	MSEL [2] (3)	MSEL [1..0]	System Configuration Mode	Configuration Mode
–	0	00	Standard	FPP
–	0	01	Standard	PPA
–	0	10	Standard	PS
1	1	00	Remote	FPP
1	1	01	Remote	PPA
1	1	10	Remote	PS
0	1	00	Local	FPP
0	1	01	Local	PPA
0	1	10	Local	PS

Notes to [Table 12-1](#):

- (1) For detailed information on standard PS, FPP, and PPA models, see the *Configuring Stratix & Stratix GX Devices* chapter of the *Stratix Device Handbook, Volume 2*.
- (2) In Stratix and Stratix GX devices, the RUnLU (remote update/local update) pin, selects between local or remote configuration mode.
- (3) The MSEL [2] select mode selects between standard or remote system configuration mode.

Remote Configuration Mode

Using remote configuration mode, you can manage up to seven different application configurations for Stratix and Stratix GX devices. The seven-configuration-file limit is due to the number of pages that the PGM [] pins in the Stratix or Stratix GX device and enhanced configuration devices can select.



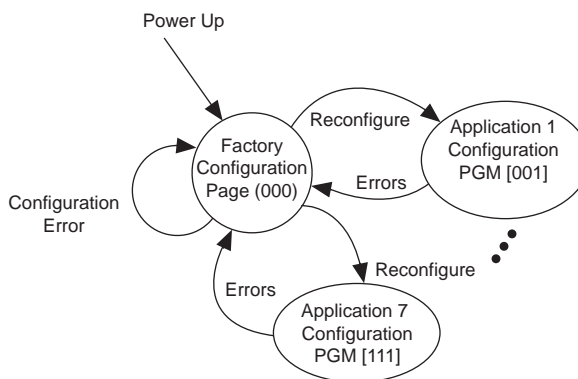
If more than seven files are sent to a system using remote configuration mode, previous files are overwritten.

Stratix and Stratix GX devices support remote configuration mode for PS, FPP, and PPA modes. Specify remote configuration mode by setting the MSEL2 and RUnLU pins to high. (See [Table 12-1](#)).

On power-up in remote configuration mode, the Stratix or Stratix GX device loads the user-specified factory configuration file, located in the default page address 000 in the enhanced configuration device. After the device configures, the remote configuration control register points to the

page address of the application configuration that should be loaded into the Stratix or Stratix GX device. If an error occurs during user mode of an application configuration, the device reloads the default factory configuration page. Figure 12-3 shows a diagram of remote configuration mode.

Figure 12-3. Remote Configuration Mode



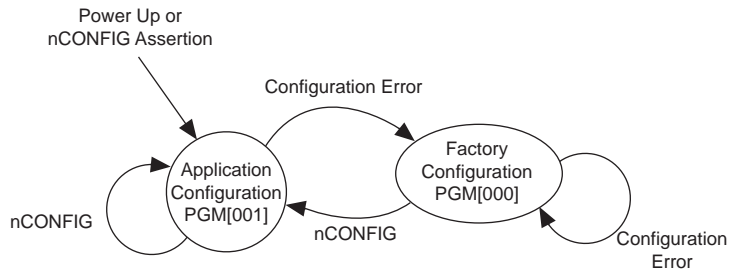
Local Configuration Mode

Local configuration mode—a simplified version of remote configuration mode—is suitable for systems that load an application immediately upon power-up. In this mode you can only use one application configuration, which you can update either remotely or locally.

In local configuration mode, upon power-up, or when `nCONFIG` is asserted, the Stratix or Stratix GX device loads the application configuration immediately. Factory configuration loads only if an error occurs during the application configuration's user mode. If you use an enhanced configuration device, page address 001 is the location for the application configuration data, and page address 000 is the location for the factory configuration data.

If the configuration data at page address 001 does not load correctly due to cyclic redundancy code (CRC) failure, or it times-out of the enhanced configuration device, or the external processor times-out, then the factory configuration located at the default page (page address 000) loads into the Stratix or Stratix GX device.

In local configuration mode (shown in Figure 12-4), the user watchdog timer is disabled. For more information on the watchdog timer, see "Watchdog Timer" on page 12-7.

Figure 12–4. Local Configuration Mode

In local configuration mode, one application configuration is available to the device. For remote or local configuration mode selection, see [Table 12–1](#).

Remote System Configuration Components

The following components are used in Stratix and Stratix GX devices to support remote and local configuration modes:

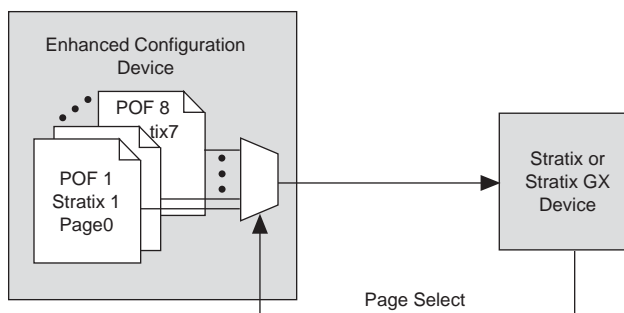
- Page mode feature
- Factory configuration
- Application configuration
- Watchdog timer
- Remote update sub-block
- Remote configuration registers

A description of each component follows.

Page Mode Feature

The page mode feature enables Stratix and Stratix GX devices to select a location to read back data for configuration. The enhanced configuration device can receive and store up to eight different configuration files (one factory and seven application files). Selection of pages to read from is performed through the PGM[2 . . 0] pins on the Stratix or Stratix GX device and enhanced configuration devices. These pins in the Stratix or Stratix GX device can be designated user I/O pins during standard configuration mode, but in remote system configuration mode, they are dedicated output pins. [Figure 12–5](#) shows the page mode feature in Stratix or Stratix GX devices and enhanced configuration devices.

Figure 12–5. Page Mode Feature in Stratix or Stratix GX Devices & Enhanced Configuration Devices



Upon power-up in remote configuration mode, the factory configuration (see description below) selects the user-specified page address through the Stratix or Stratix GX PGM[2..0] output pins. These pins drive the PGM[2..0] input pins of the enhanced configuration device and select the requested page in the memory.

If an intelligent host is used instead of an enhanced configuration device, you should create logic in the intelligent host to support page mode settings similar to that in enhanced configuration devices.

Factory Configuration

Factory configuration is the default configuration data setup. In enhanced configuration devices, this default page address is 000. Factory configuration data is written into the memory device only once by the system manufacturer and should not be remotely updated or altered. In remote configuration mode, the factory configuration loads into the Stratix or Stratix GX device upon power-up.

The factory configuration specifications are as follows:

- Receives new configuration data and writes it to the enhanced configuration or other memory devices
- Determines the page address for the next application configuration that should be loaded to the Stratix or Stratix GX device
- Upon an error in the application configuration, the system reverts to the factory configuration
- Determines the reason for any application configuration error
- Determines whether to enable or disable the user watchdog timer for application configurations

- Determines the user watchdog timer's settings if the timer is enabled (remote configuration mode)
- If the user watchdog timer is not reset after a predetermined amount of time, it times-out and the system loads the factory configuration data back to the Stratix or Stratix GX device

If a system encounters an error while loading application configuration data, or if the device re-configures due to `nCONFIG` assertion, the Stratix or Stratix GX device loads the factory configuration. The remote system configuration register determines the reason for factory re-configuration. Based on this information, the factory configuration determines which application configuration needs to be loaded.

Application Configuration

The application configuration is the configuration data received from the remote source and updated into different locations or pages of the memory storage device (excluding the factory default page).

Watchdog Timer

A watchdog timer is a circuit that determines whether another mechanism functions properly. The watchdog timer functions like a time-delay relay that remains in the reset state while an application runs properly. This action periodically sends a reset command from the working application to the watchdog timer. Stratix and Stratix GX devices are equipped with a built-in watchdog timer for remote system configuration.

A user watchdog timer prevents a faulty application configuration from indefinitely stalling the Stratix or Stratix GX device. The timer functions as a counter that counts down from an initial value, which is loaded into the device from the factory configuration. This is a 29-bit counter, but you use only the upper 12 bits to set the value for the watchdog timer. You specify the counter value according to your design needs.

The timer begins counting once the Stratix or Stratix GX device goes into user mode. If the application configuration does not reset the user watchdog timer after the specified time, the timer times-out. At this point, the Stratix or Stratix GX device is re-configured by loading the factory configuration and resetting the user watchdog timer.



The watchdog timer is disabled in local configuration mode.

Remote Update Sub-Block

The remote update sub-block is responsible for administrating the remote configuration feature. This sub-block, which is controlled by a remote configuration state machine, generates the control signals required to control different remote configuration registers.

Remote Configuration Registers

Remote configuration registers are a series of registers required to keep track of page addresses and the cause of configuration errors. [Table 12–2](#) gives descriptions of the registers' functions. You can control both the update and shift registers; the status and control registers are controlled by internal logic, but can be read via the shift register.

Register	Description
Control register	This register contains the current page address, the watchdog timer setting, and one bit specifying if the current configuration is a factory or application configuration. During a capture in an application configuration, this register is read into the shift register.
Update register	This register contains the same data as the control register, except that it is updated by the factory configuration. The factory configuration updates the register with the values to be used in the control register on the next re-configuration. During capture in a factory configuration, this register is read into the shift register.
Shift register	This register is accessible by the core logic and allows the update, status, and control registers to be written and sampled by the user logic. The update register can only be updated by the factory configuration in remote configuration mode.
Status register	This register is written into by the remote configuration block on every re-configuration to record the cause of the re-configuration. This information is used by factory configuration to determine the appropriate action following a re-configuration.

[Figure 12–6](#) shows the control, update, shift, and status registers and the data path used to control remote system configuration.

Figure 12–6. Remote Configuration Registers & Related Data Path

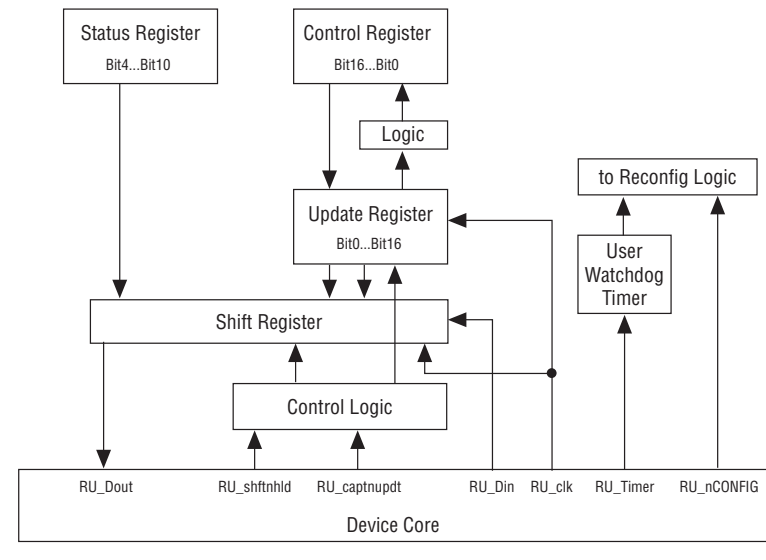


Table 12–3 describes the user configuration signals that are driven to/from the device logic array. The remote configuration logic has one input signal to the device logic array and six output signals from the device logic array.

Table 12–3. User Configuration Signals To/From Device Core (Part 1 of 2)		
Signal Name	To/From Device Core	Description
RU_Timer	Output from the core to the remote update block	Request from the application to reset the user watchdog timer with its initial count. A falling edge of this signal triggers a reset of the user watchdog timer.
RU_nCONFIG	Output from the core to the remote update block	When driven low, this signal triggers the device to reconfigure. If requested by the factory configuration, the application configuration specified in the remote update control register is loaded. If requested by the application configuration, the factory configuration is loaded.
RU_Clk	Output from the core to the remote update block	Clocks the remote configuration shift register so that the contents of the status and control registers can be read out, and the contents of update register can be loaded. The shift register latches data on the rising edge of the RU_Clk.

Table 12–3. User Configuration Signals To/From Device Core (Part 2 of 2)

Signal Name	To/From Device Core	Description
RU_shfthld	Output from the core to the remote update block	If its value is “1”, the remote configuration shift register shifts data on the rising edge of RU_Clk. If its value is “0” and RU_captnupdt is “0”, the shift register updates the update register. If its value is “0”, and RU_captnupdt is “1”, the shift register captures the status register and either the control or update register (depending on whether the configuration is factory or application).
RU_captnupdt	Output from the core to the remote update block	When RU_captnupdt is at value “1” and RU_shfthld is at value “0”, the system specifies that the remote configuration shift register should be written with the content of the status register and either the update register (in a factory configuration) or the control register (in an application configuration). This shift register is loaded on the rising edge of RU_Clk. When RU_captnupdt is at value “0” and RU_shfthld is at value “0”, the system specifies that the remote configuration update register should be written with the content of the shift register in a factory configuration. The update register is loaded on the rising edge of RU_Clk. This pin is enabled only for factory configuration in remote configuration mode (it is disabled for the application configuration in remote configuration or for local configuration modes). If RU_shfthld is at value “1”, RU_captnupdt has no function.
RU_Din	Output from the core to the remote update block	Data to be written into the remote configuration shift register on the rising edge of RU_Clk. To load into the shift register, RU_shfthld must be asserted.
RU_Dout	Input to the core from the remote update block	Output of the remote configuration shift register to be read by core logic. New data arrives on each rising edge of RU_Clk.

All of the seven device core signals (see [Figure 12–6](#)), are enabled for both remote and local configuration for both factory and application configuration, except RU_Timer and RU_captnupdt. [Figure 12–7](#) and [Table 12–4](#) specify the content of control register upon power-on reset (POR).

The difference between local configuration and remote configuration is how the control register is updated during a re-configuration and which core signals are enabled.

Figure 12–7. Remote System Configuration Control Register

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wd_timer[11..0]												Wd_en	PGM[2..0]			AnF
11 10 9 1 0												1	2	1	0	1

Table 12–4 shows the content of the control register upon POR.

Table 12–4. Control Register Contents			
Parameter	Definition	POR Reset Value	Comment
AnF	Current configuration is factory or applications	1 bit '1'	Applications
		1 bit '0'	Factory
PGM [2..0]	Page mode selection	3 bits '001'	Local configuration
		3 bits '000'	Remote configuration
Wd_en	User watchdog timer enable	1 bit '0'	–
Wd_timer [11..0]	User watchdog timer time-out value	12 bits '0'	High order bits of 29 bit counter

The status register specifies the reason why re-configuration has occurred and determines if the re-configuration was due to a CRC error, nSTATUS pulled low due to an error, the device core caused an error, nCONFIG was reset, or the watchdog timer timed-out. Figure 12–8 and Table 12–5 specify the content of the status register.

Figure 12–8. Remote System Configuration Status Register

4	3	2	1	0
Wd	nCONFIG	CORE	nSTATUS	CRC

Table 12–5 shows the content of the status register upon POR.

<i>Table 12–5. Status Register Contents</i>		
Parameter	Definition	POR Reset Value
CRC (from configuration)	CRC caused re-configuration	1 bit '0'
nSTATUS	nSTATUS caused re-configuration	1 bit '0'
CORE (1)	Device core caused re-configuration	1 bit '0'
nCONFIG	NCONFIG caused re-configuration	1 bit '0'
wd	Watchdog Timer caused re-configuration	1 bit '0'

Note to Table 12–5:

- (1) Core re-configuration enforces the system to load the application configuration data into the Stratix or Stratix GX device. This occurs after factory configuration specifies the appropriate application configuration data.

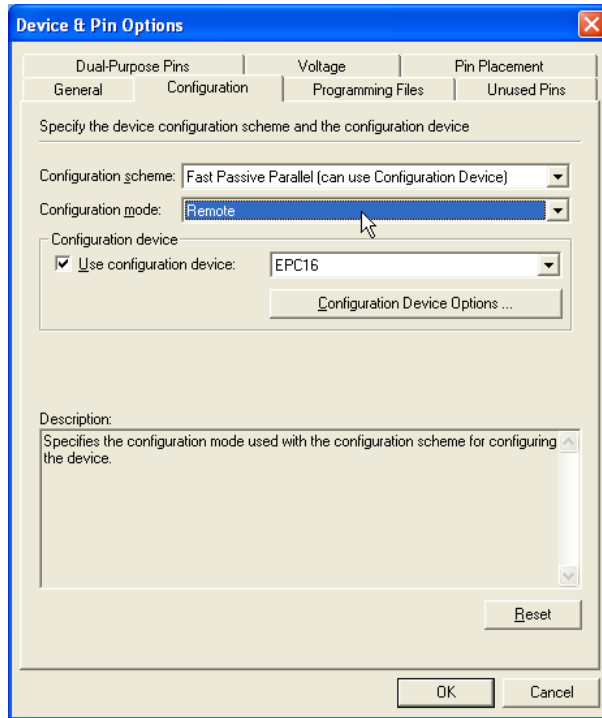
Quartus II Software Support

The Quartus II software supports implementation of both remote and local configuration modes in your Stratix or Stratix II device. To include the remote or local configuration feature to your design, select remote or local as the configuration mode under the **Device & Pin Options** compiler settings (prior to compilation). This selection reserves the dual-purpose RUNLU and PGM[2 : 0] pins for use as dedicated inputs in remote/local configuration modes.

To set the configuration mode as remote or local, follow these steps (See Figure 12–9):

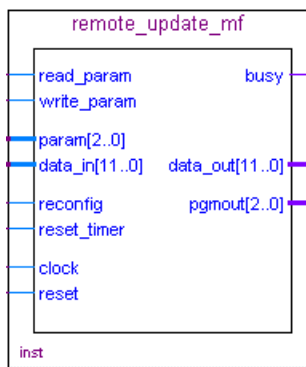
1. Open the **Device & Pin Options** settings window under the **Assignments** menu.
2. Select **Device & Pin Options** dialog box. The **Device & Pin Options** dialog box is displayed.
3. Click the **Configuration** tab.
4. In the **Configuration mode** list, select **Remote** or **Local**.

The Standard mode selection disables the remote system configuration feature. In addition to the mode selection, you can specify the configuration scheme and configuration device (if any) used by your setup.

Figure 12–9. Device & Pin Options Dialog Box

Additionally, the remote configuration mode requires you to either instantiate the `altremote_update` megafunction or the WYSIWYG (what-you-see-is-what-you-get) atom into your design. Without this atom or megafunction, you are not able to access the dedicated remote configuration circuitry or registers within the Stratix or Stratix GX device. See [Figure 12–10](#) for a symbol of the `altremote_update` megafunction.

The local configuration mode, however, can be enabled with only the device **Configuration Options** compiler setting.

Figure 12–10. *altremote_update* Megafunction Symbol


altremote_update Megafunction

A remote update megafunction, `altremote_update`, is provided in the Quartus II software to provide a memory-like interface to allow for easy control of the remote update parameters. Tables 12–6 and 12–7 describe the input and output ports available on the `altremote_update` megafunction. Table 12–8 shows the `param[2..0]` bit settings.

Table 12–6. Input Ports of the *altremote_update* Megafunction (Part 1 of 2)

Port Name	Required	Source	Description
clock	Y	Logic Array	Clock input to the <code>altremote_update</code> block. All operations are performed with respects to the rising edge of this clock.
reset	Y	Logic Array	Asynchronous reset, which is used to initialize the remote update block. To ensure proper operation, the remote update block must be reset before first accessing the remote update block. This signal is not affected by the busy signal and will reset the remote update block even if busy is logic high. This means that if the reset signal is driven logic high during writing of a parameter, the parameter will not be properly written to the remote update block.
reconfig	Y	Logic Array	When driven logic high, reconfiguration of the device is initiated using the current parameter settings in the remote update block. If busy is asserted, this signal is ignored. This is to ensure all parameters are completely written before reconfiguration begins.
reset_timer	N	Logic Array	This signal is required if you are using the watchdog timer feature. A logic high resets the internal watchdog timer. This signal is not affected by the busy signal and can reset the timer even when the remote update block is busy. If this port is left connected, the default value is 0.

Table 12–6. Input Ports of the *altremote_update* Megafunction (Part 2 of 2)

Port Name	Required	Source	Description
read_param	N	Logic Array	Once <code>read_param</code> is sampled as a logic high, the busy signal is asserted. While the parameter is being read, the busy signal remains asserted, and inputs on <code>param[]</code> are ignored. Once the busy signal is deactivated, the next parameter can be read. If this port is left unconnected, the default value is 0.
write_param	N	Logic Array	This signal is required if you intend on writing parameters to the remote update block. When driven logic high, the parameter specified on the <code>param[]</code> port should be written to the remote update block with the value on <code>data_in[]</code> . The number of valid bits on <code>data_in[]</code> is dependent on the parameter type. This signal is sampled on the rising edge of clock and should only be asserted for one clock cycle to prevent the parameter from being re-read on subsequent clock cycles. Once <code>write_param</code> is sampled as a logic high, the busy signal is asserted. While the parameter is being written, the busy signal remains asserted, and inputs on <code>param[]</code> and <code>data_in[]</code> are ignored. Once the busy signal is deactivated, the next parameter can be written. This signal is only valid when the <code>Current_Configuration</code> parameter is factory since parameters cannot be written in application configurations. If this port is left unconnected, the default value is 0.
param[2..0]	N	Logic Array	3-bit bus that selects which parameter should be read or written. If this port is left unconnected, the default value is 0.
data_in[11..0]	N	Logic Array	This signal is required if you intend on writing parameters to the remote update block 12-bit bus used when writing parameters, which specifies the parameter value. The parameter value is requested using the <code>param[]</code> input and by driving the <code>write_param</code> signal logic high, at which point the busy signal goes logic high and the value of the parameter is captured from this bus. For some parameters, not all 12-bits will be used in which case only the least significant bits will be used. This port is ignored if the <code>Current_Configuration</code> parameter is set to an application configuration since writing of parameters is only allowed in the factory configuration. If this port is left unconnected, the default values is 0.

Note to Table 12–6:

- (1) Logic array source means that you can drive the port from internal logic or any general-purpose I/O pin.

Table 12–7. Output Ports of the *altremote_update* Megafunction

Port Name	Required	Destination	Description
busy	Y	Logic Array	When this signal is a logic high, the remote update block is busy either reading or writing a parameter. When the remote update block is busy, it ignores its <code>data_in[]</code> , <code>param[]</code> , and <code>reconfig</code> inputs. This signal will go high when <code>read_param</code> or <code>write_param</code> is asserted and will remain asserted until the operation is complete.
pgm_out[2..0]	Y	PGM[2..0] pins	3-bit bus that specifies the page pointer of the configuration data to be loaded when the device is reconfigured. This port must be connected to the PGM[] output pins, which should be connected to the external configuration device
data_out[11..0]	N	Logic Array	12-bit bus used when reading parameters, which reads out the parameter value. The parameter value is requested using the <code>param[]</code> input and by driving the <code>read_param</code> signal logic high, at which point the busy signal will go logic high. When the busy signal goes low, the value of the parameter will be driven out on this bus. The <code>data_out[]</code> port is only valid after a <code>read_param</code> has been issued and once the busy signal is de-asserted. At any other time, its output values are invalid. For example, even though the <code>data_out[]</code> port may toggle during a writing of a parameter, these values are not a valid representation of what was actually written to the remote update block. For some parameters, not all 12-bits will be used in which case only the least significant bits will be used.

Note to [Table 12–7](#):

- (1) Logic array destination means that you can drive the port to internal logic or any general-purpose I/O pin.

Table 12–8. Parameter Settings for the *altremote_update* Megafunction (Part 1 of 2)

Selected Parameter	param[2..0] bit setting	width of parameter value	POR Reset Value	Description
Status Register Contents	000	5	5 bit '0	Specifies the reason for re-configuration, which could be caused by a CRC error during configuration, <code>nSTATUS</code> being pulled low due to an error, the device core caused an error, <code>nCONFIG</code> pulled low, or the watchdog timer timed-out. This parameter can only be read.
Watchdog Timeout Value	010	12	12 bits '0	User watchdog timer time-out value. Writing of this parameter is only allowed when in the factory configuration.
Watchdog Enable	011	1	1 bit '0	User watchdog timer enable. Writing of this parameter is only allowed when in the factory configuration

Table 12–8. Parameter Settings for the `altremote_update` Megafunction (Part 2 of 2)

Selected Parameter	param[2..0] bit setting	width of parameter value	POR Reset Value	Description
Page select	100	3	3 bit '001' - Local configuration	Page mode selection. Writing of this parameter is only allowed when in the factory configuration.
			3 bit '000' - Remote configuration	
Current configuration (AnF)	101	1	1 bit '0' - Factory	Specifies whether the current configuration is factory or and application configuration. This parameter can only be read.
			1 bit '1' - Application	
Illegal values	001			
	110			
	111			

Remote Update WYSIWYG ATOM

An alternative to using the `altremote_update` megafunction is to directly instantiate the remote update WYSIWYG atom. This atom should be included in the factory configuration and any application configuration image to access the remote configuration shift registers.

When implementing the atom, you should consider following:

1. Only one atom can be used in the circuit; more than one gives a no-fit.
2. All signals for the cell must be connected. The clock port (CLK) must be connected to a live cell. The others can be constant V_{CC} or GND.
3. The `pgmout` port must be connected and must feed `PGM[2..0]` output pins (it cannot be connected to anything else but output pins).
4. The Quartus II software reserves `RUNLU` as an input pin, and you must connect it to V_{CC} .

The Stratix and Stratix GX remote update atom ports are:

```

Stratix_rublock <rublock_name>
(
    .clk(<clock source>),
    .shiftnld(<shiftnld source>),
    .captnupdt(<shiftnld source>),
    .regin(<regin input source from the core>),
    .rsttimer(<input signal to reset the watchdog timer>),
    .config(<input signal to initiate configuration>),
    .regout(<data output destination to core>),
    .pgmout(<program output destinations to pins>)

```

Table 12–9 shows the remote update block input and output port names and descriptions.

Ports	Definition
<rublock_name>	The unique identifier for the instance. This identifier name can be anything as long as it is legal for the given description language (i.e., Verilog, VHDL, AHDL, etc.). This field is required.
.clk(<clock source>)	Designates the clock input of this cell. All operation is with respect to the rising edge of this clock. This field is required.
.shiftnld(<shiftnld source>)	An input into the remote configuration block. When .shiftnld = 1, the data shifts from the internal shift registers to the regout port at each rising edge of clk , and the data also shifts into the internal shift registers from regin port. This field is required.
.captnupdt(<shiftnld source>)	An input into the remote configuration block. This controls the protocol of when to read the configuration mode or when to write into the registers that control the configuration. This field is required.
.regin(<regin input source from the core>)	An input into the configuration block for all data loading into the core. The data shifts into the internal registers at the rising edge of clk . This field is required.
.rsttimer(<input signal to reset the watchdog timer>)	An input into the watchdog timer of the remote update block. When this is high, it resets the watchdog timer. This field is required.
.config(<input signal to initiate configuration>)	An input into the configuration section of the remote update block. When this signal goes high, the part initiates a re-configuration. This field is required.
.regout(<data output destination to core>)	A 1-bit output, which is the output of the internal shift register, and updated every rising edge of clk . The data coming out depends on the control signals. This field is required.
.pgmout(<program output destinations to pins>)	A 3-bit bus. It should always be connected only to output pins (not bidir pins). This bus gives the page address (000 to 111) of the configuration data to be loaded when the device is getting configured. This field is required.



For more information on the control signals for the remote block, see [Table 12-3 on page 12-9](#).

Using Enhanced Configuration Devices

This section describes remote system configuration of Stratix and Stratix GX devices with the Nios embedded processor using enhanced configuration devices. Enhanced configuration devices are composed of a standard flash memory and a controller. The flash memory stores configuration data, and the controller reads and writes to the flash memory.

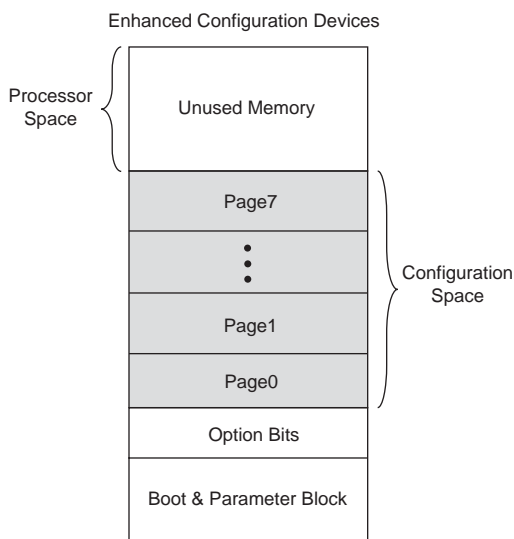
In remote system configuration, only PS and FPP modes are supported using an enhanced configuration device. A Stratix or Stratix GX device running a Nios embedded processor can receive data from a remote source through a network or any other appropriate media. A specific page of the enhanced configuration device stores the received data.

This scheme uses the page mode option in Stratix and Stratix GX devices. Up to eight pages can be stored in each enhanced configuration device, each of which can store a configuration file.

In enhanced configuration devices, a page is a section of the flash memory space. Its boundary is determined by the Quartus II software (the page size is programmable). In the software, you can specify which configuration file should be stored in which page within the flash memory. To access the configuration file on each page, set the three input pins ($PGM[2..0]$), which provide access to all eight pages. Because the $PGM[2..0]$ pins of an enhanced configuration device connect to the same pins of the Stratix or Stratix GX device, the Stratix or Stratix GX device selects one of the eight memory pages as a target location to read from. [Figure 12-11](#) shows the allocation of different pages in the enhanced configuration device.



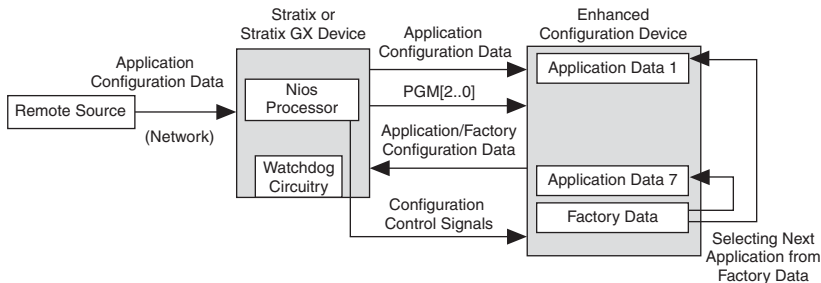
For more information on enhanced configuration devices, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* and the *Altera Enhanced Configuration Devices* chapter.

Figure 12–11. Memory Map in Enhanced Configuration Device

When the Stratix or Stratix GX device powers-up in remote configuration mode, the device loads configuration data located at page address 000. You should always load the factory default configuration data at this location and make sure this information is not altered.

The factory configuration contains information to determine the next application configuration to load into the Stratix or Stratix GX device. When the Stratix or Stratix GX device successfully loads the application configuration from the page selected by the PGM[2..0] pins, it enters user mode.

In user mode, the Nios embedded processor (or any other logic) assists the Stratix or Stratix GX device in detecting remote system configuration information. In remote system configuration, the Nios embedded processor receives the incoming data from the remote source via the network, writes it to the ECP16 enhanced configuration device, and then initiates loading of the factory configuration into the Stratix or Stratix GX device. Factory configuration reads the remote configuration status register and determines the appropriate application configuration to load into the Stratix or Stratix GX device. [Figure 12–12](#) shows the remote system configuration.

Figure 12–12. Remote System Configuration Using Enhanced Configuration Devices

The user watchdog timer in Stratix and Stratix GX devices ensures that an application configuration has loaded successfully and checks if the application configuration is operating correctly in user mode. The watchdog timer must be continually reset by the user logic. If an error occurs while the application configuration loads, or if the watchdog timer times-out during user mode, the factory configuration is reloaded to prevent the system from halting in an erroneous state. [Figure 12–3 on page 12–4](#) illustrates the remote configuration mode.

Upon power-up in local configuration scheme, the application configuration at page 001 (PGM[001] of the enhanced configuration device) loads into the Stratix or Stratix GX device. This application can be remotely or locally updated. If an error occurs during loading of the configuration data, the factory configuration loads automatically (see [Figure 12–4 on page 12–5](#)). The rest is identical to remote configuration mode.

Local Update Programming File Generation

This section describes the programming file generation process for performing remote system upgrades. The Quartus II convert programming files (CPF) utility generates the initial and partial programming files for configuration memory within the enhanced configuration devices.

The two pages that local configuration mode uses are a factory configuration stored at page 000, and an application configuration stored at page 001. The factory configuration cannot be updated after initial production programming. However, the application configuration can be erased and reprogrammed after initial system deployment.

In local update mode, you would first create the initial programming file with the factory configuration image and a version of the application configuration. Subsequently, you can generate partial programming files to update the application configuration (stored in page 001). Quartus II CPF can create partial programming files in **.hex** (Hexadecimal file), **JAM**, **.jbc** (JAM Byte-Code File), and **POF** formats.

In addition to the two configuration pages, user data or processor code can also be pre-programmed in the bottom boot and main data areas of the enhanced configuration device memory. The CPF utility accepts a HEX input file for the bottom and main data areas, and includes this data in the POF output file. However, this is only supported for initial programming file generation. Partial programming file generation for updating user HEX data is not supported, but can be performed using the enhanced configuration device external flash interface.

Initial Programming File Generation

The initial programming file includes configuration data for both factory and application configuration pages. The enhanced configuration device option's bits are always located between byte addresses 0x00010000 and 0x0001003F. Also, page 0 always starts at 0x00010040 while its end address is dependent on the size of the factory configuration data.

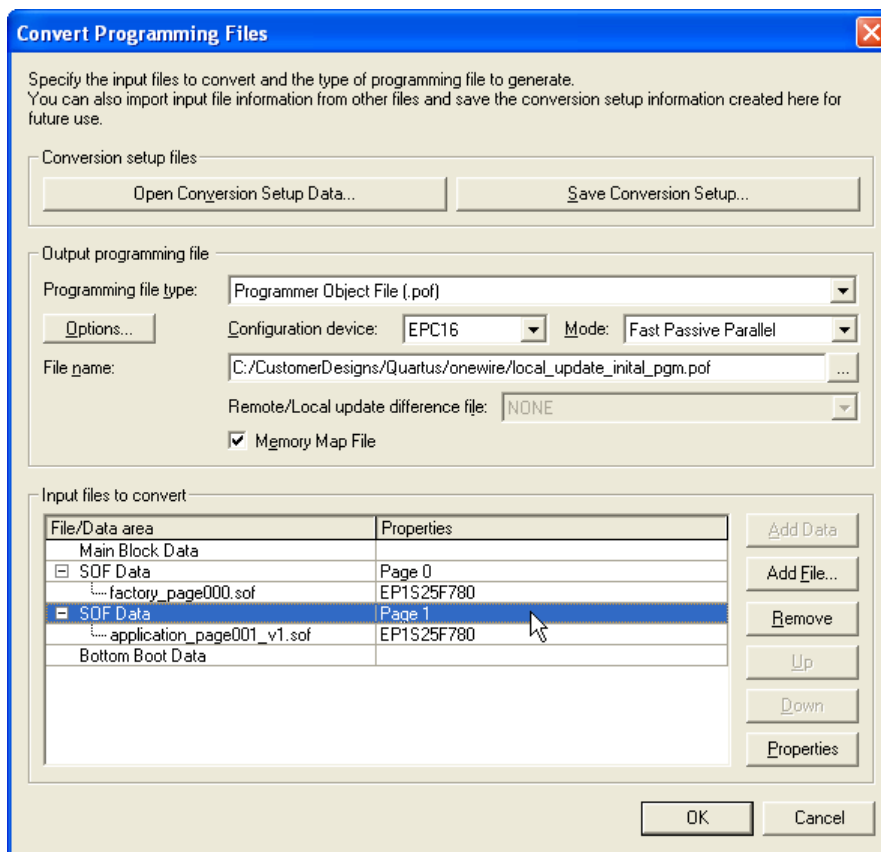
The two memory allocation options that exist for the application configuration are auto addressing and block addressing. In auto addressing mode, Quartus II automatically allocates memory for the application configuration. All the configuration memory sectors that are not used by the page 0 factory configuration are allocated for page 1. The memory allocated is maximized to allow future versions of the application configuration to grow and have bigger configuration files (when the compression feature is enabled). Processor or user data storage (HEX input file) is only supported by the bottom boot area in auto addressing mode.

The following steps and screen shot (see [Figure 12–13](#)) describe initial programming file generation with auto addressing mode.

1. Open the **Convert Programming Files** window from the **File** menu.
2. Select **Programmer Object File (*.pof)** from the drop-down list titled **Programming File Type**.

3. Select the enhanced configuration device used (EPC4, EPC8, EPC16), and the mode used (1-bit Passive Serial or Fast Passive Parallel). Only during the initial programming file generation can you specify the **Options**, **Configuration Device**, or **Mode** settings. While generating the partial programming file, all of these settings are grayed out and inaccessible.
4. In the **Input files to convert** box, highlight **SOF Data at Page 0** and click **Add File**. Select input SOF file(s) for this configuration page and insert them.
5. Repeat Step 4 for the **Page 1** application configuration page.
6. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of each configuration page and user data blocks.
7. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the **.cof** output file.
8. Click **OK** to generate initial programming and memory map files.

Figure 12–13. CPF Setup for Initial Programming File (Auto Addressing)



A sample memory map output file for the preceding setup is shown below. Configuration option bits and page 0 data occupy main flash sectors 0 through 4. See the *Sharp LHF16J06 Flash memory used in EPC16 devices* Data Sheet at www.altera.com to correlate memory addresses to the EPC16 flash sectors. In auto addressing mode, page 1 allocates all unused flash sectors. For this example, this unused area includes main sectors 5 through 30, and all of the bottom boot sectors. While this large portion of memory is allocated for page 1, the real application configuration data is top justified within this region with filler 1'b1 bits in lower memory addresses. Notice that the page 1 configuration data

wraps around the top of the memory and fills up the bottom boot area. The wrap around does not occur if the bottom boot area is used for processor/user HEX data file storage.

Block	Start Address	End Address
OPTION BITS	0x00010000	0x0001003F
PAGE 0	0x00010040	0x00054CC8
PAGE 1	0x001CB372	0x0000FFFD wrapped around

The block addressing mode allows better control of flash memory allocation. You can allocate a specific flash memory region for each application configuration page. This allocation is done by specifying a block starting and block ending address. While selecting the size of the region, you should account for growth in compressed configuration bitstream sizes due to design changes and additions. In local update mode, all configuration data is top justified within this allotted memory. In other words, the last byte of configuration data is stored such that it coincides with the highest byte address location within the allotted space. Lower unused memory address locations within the allotted region are filled with 1's. These filler bits are transmitted during a configuration cycle using page 1, but are ignored by the Stratix device. The memory map output file provides the exact byte address where real configuration data for page 1 begins. Note that any partial update of page 1 should erase all allotted flash sectors before storing new configuration data.

In the block addressing mode, HEX input files can be optionally added to the bottom boot and main flash data areas (one HEX file per area is allowed). The HEX file can be stored with relative addressing or absolute addressing. For more information on relative and absolute addressing, see the *Using Altera Enhanced Configuration Devices* chapter of the *Configuration Handbook*.

Figures 12–14 and 12–15, and the following steps illustrate generating an initial programming file with block addressing for local update mode. This example also illustrates preloading user HEX data into bottom boot and main flash sectors.

1. Open the **Convert Programming Files** window from the **File** menu.
2. Select **Programmer Object File (.pof)** from the drop-down list titled **Programming file type**.

3. Select the enhanced configuration device (EPC4, EPC8, EPC16), and the mode used (1-bit Passive Serial or Fast Passive Parallel). Only during the initial programming file generation can you specify the **Options**, **Configuration device**, or **Mode** settings. While generating the partial programming file, all of these settings are grayed out and inaccessible.
4. In the **Input files to convert** box, highlight **SOF Data at Page 0** and click **Add File**. Select input SOF file(s) for this configuration page and insert them.
5. Repeat Step 4 for the Page 1 application configuration page.
6. For enabling block addressing, select the **SOF Data** entry for **Page 1**, and click **Properties**. This opens the **SOF Data Properties** dialog box (see [Figure 12–15](#)).
7. Pick **Block** from the **Address Mode** drop down selection, and enter 32-bit Hexadecimal byte address for block **Starting Address** and **Ending Address**. Note that for partial programming support, the block start and end addresses should be aligned to a flash sector boundary. This prevents two configuration pages from overlapping within the same flash boundary. See the flash memory datasheet for data sector boundary information. Click **OK** to save SOF data properties.
8. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of each configuration page and user data blocks.
9. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the COF output file.
10. Click **OK** to generate initial programming and memory map files.

Figure 12–14. CPF Setup for Initial Programming File Generation (Block Addressing)

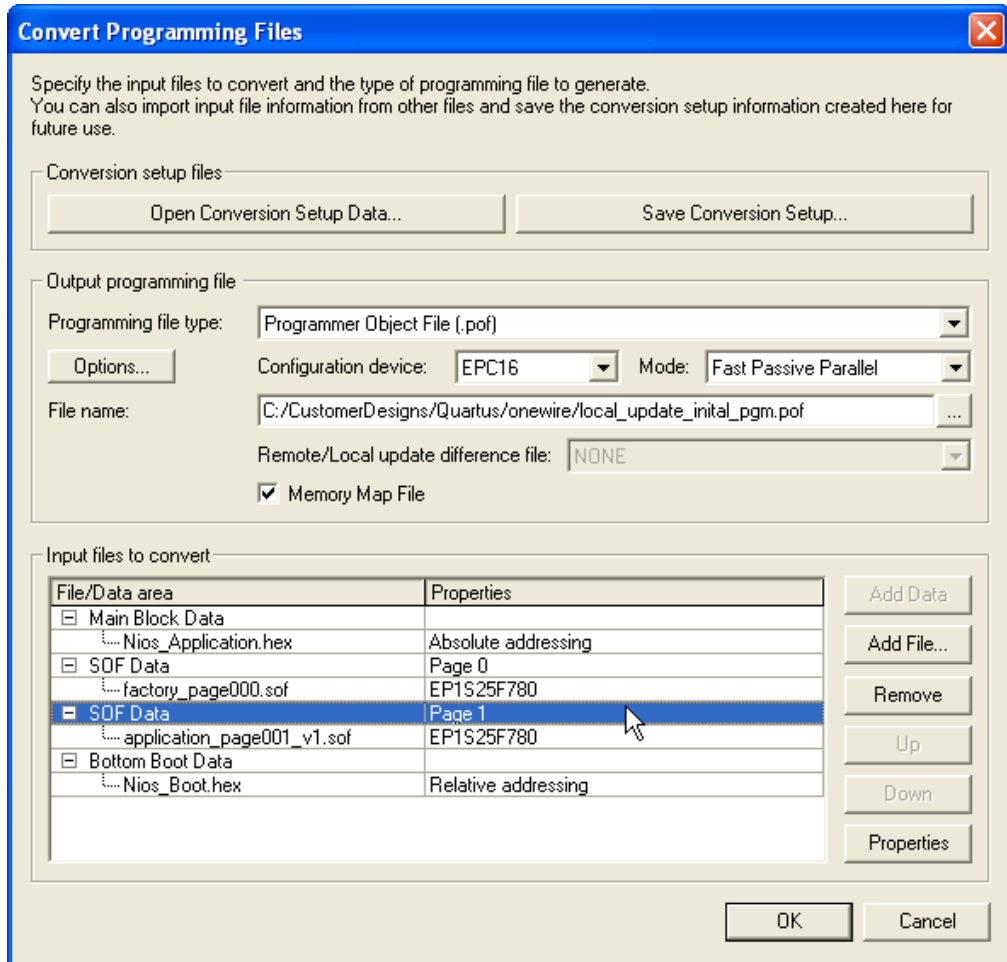
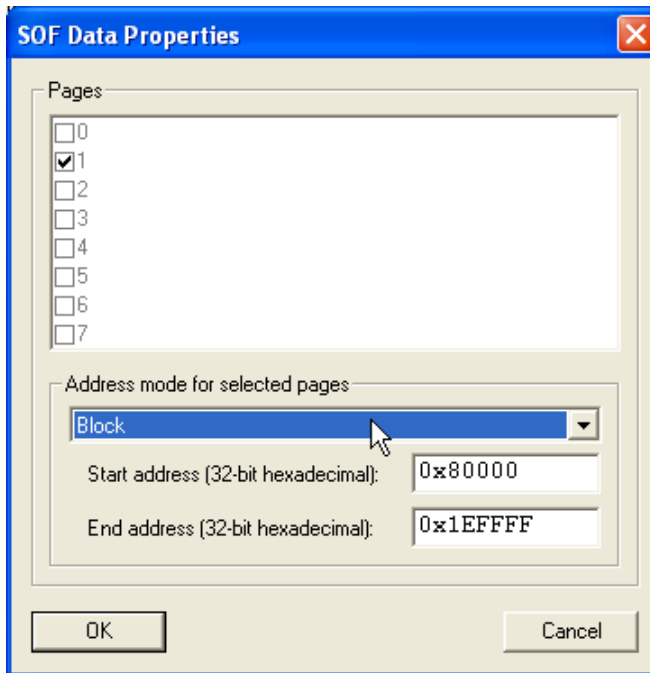


Figure 12–15. Specifying Block Addresses for Application Configuration

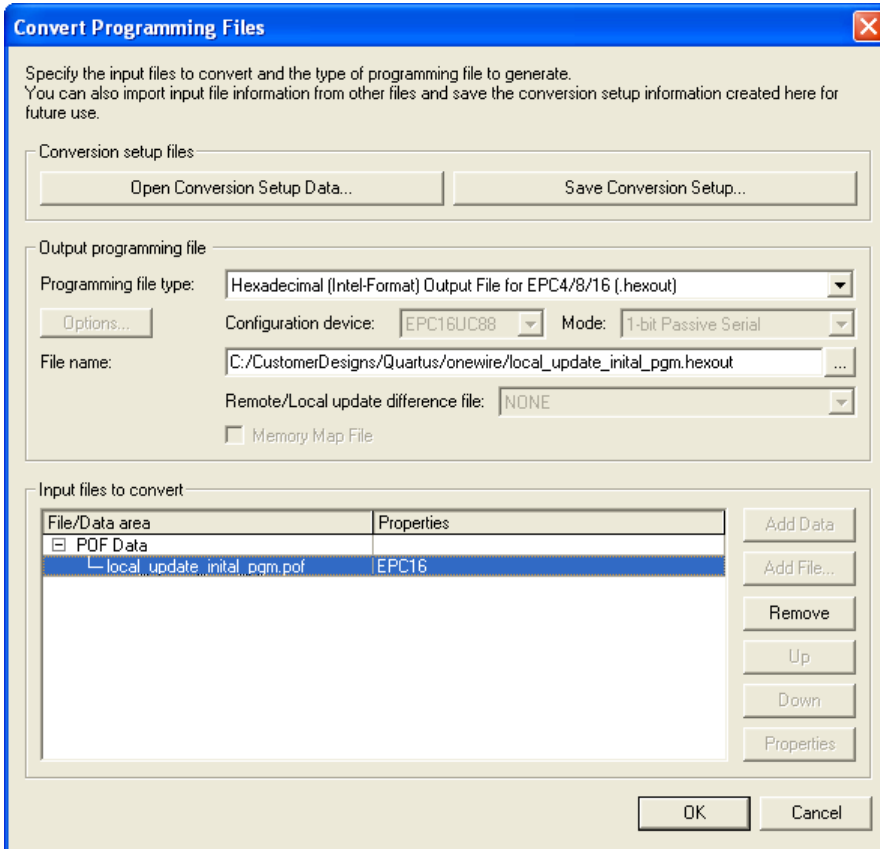
A sample memory map output file for the preceding example is shown below. Note that the allocated memory for page 1 is between 0x00080000 and 0x001EFFFF, while the actual region used by the current application configuration bitstream is between 0x001AB36C and 0x001EFFF7. The configuration data is top justified within the allocated SOF data region.

Block	Start Address	End Address
BOTTOM BOOT	0x00000000	0x000001FF
OPTION BITS	0x00010000	0x0001003F
PAGE 0	0x00010040	0x00054CC8
PAGE 1	0x001AB36C	0x001EFFF7
TOP BOOT/MAIN	0x001F0000	0x001F01FF

Also note that the HEX data stored in the main data area uses absolute addressing. If relative addressing were to be used, the main data contents would be justified with the top (higher address locations) of the memory.

The initial programming file (POF) can be converted to an Intel Hexadecimal format file (*.HEXOUT) using the Quartus II CPF utility. See [Figure 12–16](#).

Figure 12–16. Converting POF Programming File to Intel HEX Format



Partial Programming File Generation

The enhanced Quartus II CPF utility allows an existing application configuration page to be replaced with new data. Partial programming files are generated to perform such configuration data updates.

In order to generate a partial programming file, you have to input the initial programming file (POF) and new configuration data (SOF) to the Quartus II CPF utility. In addition, you have to specify the addressing mode (auto or manual) that was used during initial POF creation. And if

block addressing was used, you should specify the block start and end addresses. With this information, Quartus II ensures that the partial programming file only updates the flash region containing the application configuration. The factory configuration (page 0) and configuration option bits are left unaltered during this process.

Figure 12–17 and the following steps illustrate generation of a partial programming file:

1. Open the **Convert Programming Files** window from the **File** menu.
2. Select **Programmer Object File for Local Update (.pof)** from the drop-down list titled **Programming file type**, and specify an output **File name**.
3. In the **Input files to convert box**, highlight **POF Data** and click **Add File**. Select the initial programming POF file for this design and insert it.
4. In the **Input files to convert box**, highlight **SOF Data** and click **Add File**. Select the new application configuration bitstream (SOF) and insert it.
5. When using block addressing, select the **SOF Data** entry for **Page 1**, and click **Properties**. This opens the **SOF Data Properties** dialog box (see Figure 12–18).
6. Pick **Block** from the **Address Mode** drop down selection, and enter 32-bit Hexadecimal byte address for block **Starting Address** and **Ending Address**. These addresses should be identical to those used to generate the initial programming file. Click **OK** to save SOF data properties.
7. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of the new application configuration data in page 1.
8. Pick a local update difference file from the **Remote/Local Update Difference File** drop-down menu. You can select between an Intel HEX, JAM, JBC, and POF output file types. The output file name is the same as the POF output file name with a **_dif** suffix.
9. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the COF output file.
10. Click **OK** to generate initial programming and memory map files.

Figure 12–17. Local Update Partial Programming File Generation

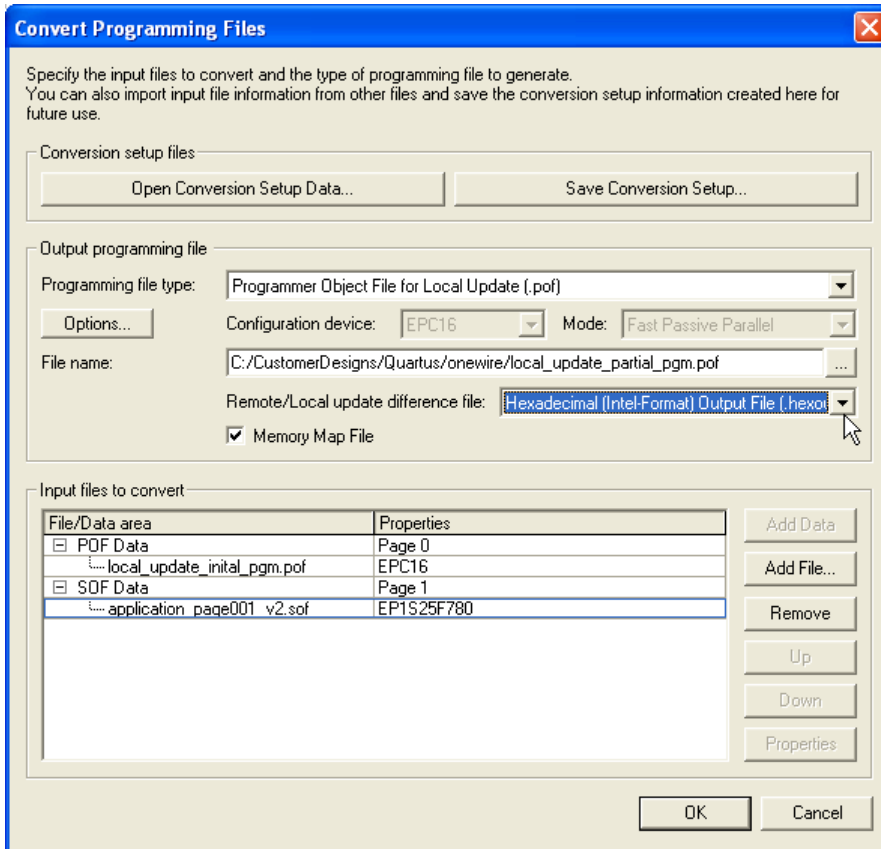
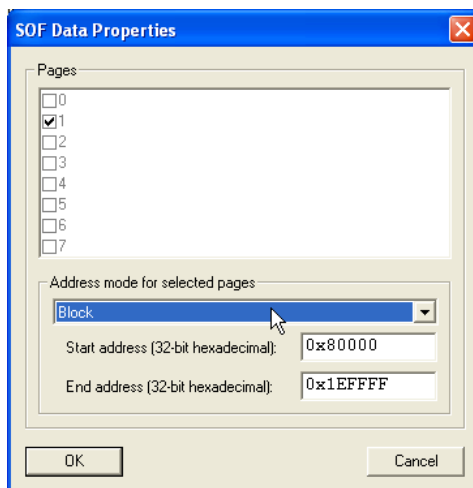


Figure 12–18. Specifying Block Addresses for Application Configuration

Remote Update Programming File Generation

This section describes the programming file generation process for performing remote system upgrades. The Quartus II CPF utility generates the initial and partial programming files for configuration memory within the enhanced configuration devices.

Remote configuration mode uses a factory configuration stored at page 0, and up to seven application configurations stored at pages 1 through 7. The factory configuration cannot be updated after initial production programming. However, the most recent application configuration can be erased and reprogrammed after initial system deployment. Alternatively, a new application configuration can be added provided adequate configuration memory availability.

In remote update mode, you would first create the initial programming file with the factory configuration image and the application configuration(s). Subsequently, you can generate partial programming files to update the most recent application configuration or add a new application configuration. Quartus II CPF can create partial programming files in HEX, JAM, JBC, and POF formats.

In addition to the configuration pages, user data or processor code can also be pre-programmed in the bottom boot and main data areas of the enhanced configuration device memory. The CPF utility accepts a HEX input file for the bottom and main data areas, and includes this data in the

POF output file. However, this is only supported for initial programming file generation. Partial programming file generation for updating user HEX data is not supported, but can be performed using the enhanced configuration device external flash interface.

Initial Programming File Generation

The initial programming file includes configuration data for both factory and application configuration pages. The enhanced configuration device option's bits are always located between byte addresses 0x00010000 and 0x0001003F. Also, page 0 always starts at 0x00010040 while its end address is dependent on the size of the factory configuration data.

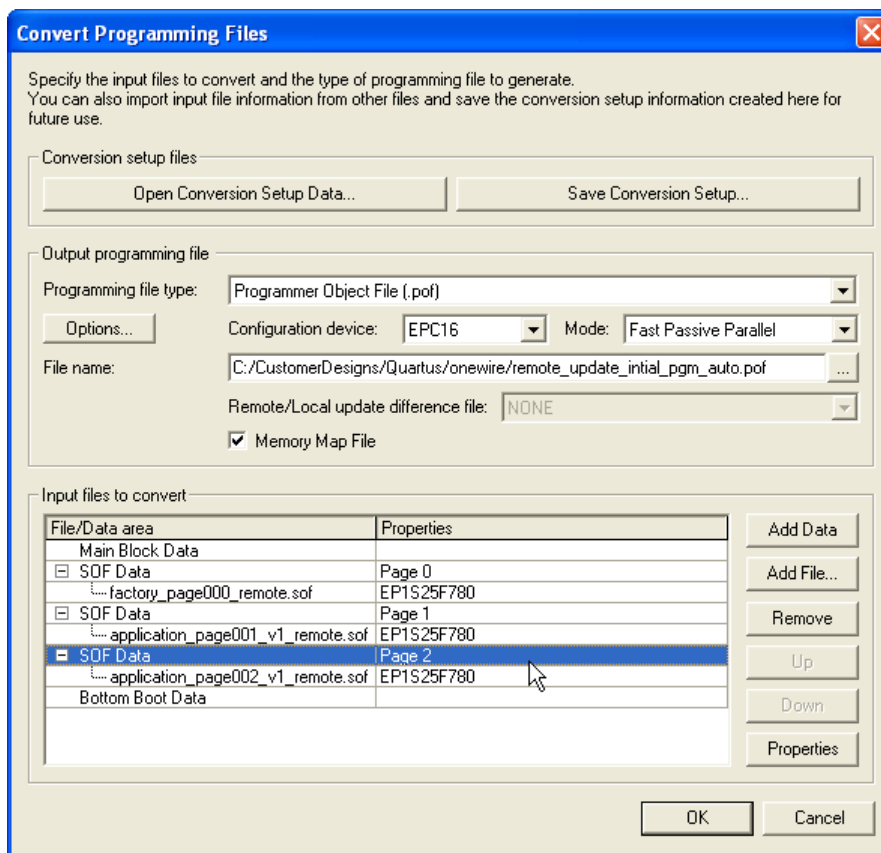
Two memory allocation options exist for application configurations: auto addressing and block addressing. In auto addressing mode, Quartus II packs all application configurations as close together as possible. This maximizes the number of application configurations that can be stored in memory. However, when auto addressing is used you cannot update existing application configurations. Only new application configurations can be added to the memory.

The following steps and screen shot (see [Figure 12-19](#)) describe initial programming file generation with auto addressing mode.

1. Open the **Convert Programming Files** window from the **File** menu.
2. Select **Programmer Object File (*.pof)** from the drop-down list titled **Programming file type**.
3. Select the enhanced configuration device used (EPC4, EPC8, EPC16), and the mode used (1-bit Passive Serial or Fast Passive Parallel). Only during the initial programming file generation can you specify the **Options**, **Configuration device**, or **Mode** settings. While generating the partial programming file, all of these settings are grayed out and inaccessible.
4. In the **Input files to convert box**, highlight **SOF Data at Page 0** and click **Add File**. Select input SOF file(s) for this configuration page and insert them.
5. Repeat Step 4 for all application configurations (up to 7 maximum).
6. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of each configuration page and user data blocks.

7. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the COF output file.
8. Click **OK** to generate initial programming and memory map files.

Figure 12–19. CPF Setup for Initial Programming File Generation (Auto Addressing)



A sample memory map output file for the preceding setup is shown below. Notice all configuration pages are packed such that two pages can share a flash data sector. This disallows partial programming of application configurations in auto addressing mode.

Block	Start Address	End Address
OPTION BITS	0x00010000	0x0001003F
PAGE 0	0x00010040	0x00054EFA
PAGE 1	0x00054EFC	0x00099DB6
PAGE 2	0x00099DB8	0x000DEC72



See the *Sharp LHF16J06 Data Sheet Flash memory used in EPC16 devices* at www.altera.com to correlate memory addresses to the EPC16 flash sectors.

The block addressing mode allows better control of flash memory allocation. You can allocate a specific flash memory region for each application configuration page. This allocation is done by specifying a block starting and block ending address. While selecting the size of the region, you should account for growth in compressed configuration bitstream sizes due to design changes and additions. In remote update mode, all configuration data is top justified within this allotted memory. In other words, the last byte of configuration data is stored such that it coincides with the highest byte address location within the allotted space. Lower unused memory address locations within the allotted region are filled with 1's. These filler bits are transmitted during the application configuration cycle, but are ignored by the Stratix device. The memory map output file provides the exact byte address where real application configuration data for each page begins. Note that any partial update of the most recent application configuration should erase all allotted flash sectors for that page before storing new configuration data.

In the block addressing mode, HEX input files can be optionally added to the bottom boot and main flash data areas (one HEX file per area is allowed). The HEX file can be stored with relative addressing or absolute addressing. For more information on relative and absolute addressing, see the *Enhanced Configuration Devices (EPC4, EPC8 & EPC16) Data Sheet* chapter of the *Configuration Handbook, Volume 2*.

Figures 12–20 and 12–21, and the following steps illustrate generating an initial programming file with block addressing for remote update mode. This example also illustrates preloading user HEX data into bottom boot and main flash sectors.

1. Open the **Convert Programming Files** window from the **File** menu.

2. Select **Programmer Object File (*.pof)** from the drop-down list titled **Programming file type**.
3. Select the enhanced configuration device used (EPC4, EPC8, EPC16), and the mode used (1-bit Passive Serial or Fast Passive Parallel). Only during the initial programming file generation can you specify the **Options**, **Configuration device**, or **Mode** settings. While generating the partial programming file, all of these settings are grayed out and inaccessible.
4. In the **Input files to convert** box, highlight **SOF Data at Page 0** and click **Add File**. Select input SOF file(s) for this configuration page and insert them.
5. Repeat Step 4 for all the application configuration pages (pages 1 and 2 in this example).
6. For enabling block addressing, select the **SOF Data** entry for **Page 1**, and click **Properties**. This opens the **SOF Data Properties** dialog box (see [Figure 12–21](#)).
7. Pick **Block** from the **Address Mode** drop down selection, and enter 32-bit Hexadecimal byte address for block **Starting Address** and **Ending Address**. Note that for partial programming support, the block start and end addresses should be aligned to a flash sector boundary. This prevents two configuration pages from overlapping within the same flash boundary. See the flash memory datasheet for data sector boundary information. Click **OK** to save SOF data properties.
8. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of each configuration page and user data blocks.
9. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the COF output file.
10. Click **OK** to generate initial programming and memory map files.

Figure 12–20. CPF Setup for Initial Programming File Generation (Block Addressing)

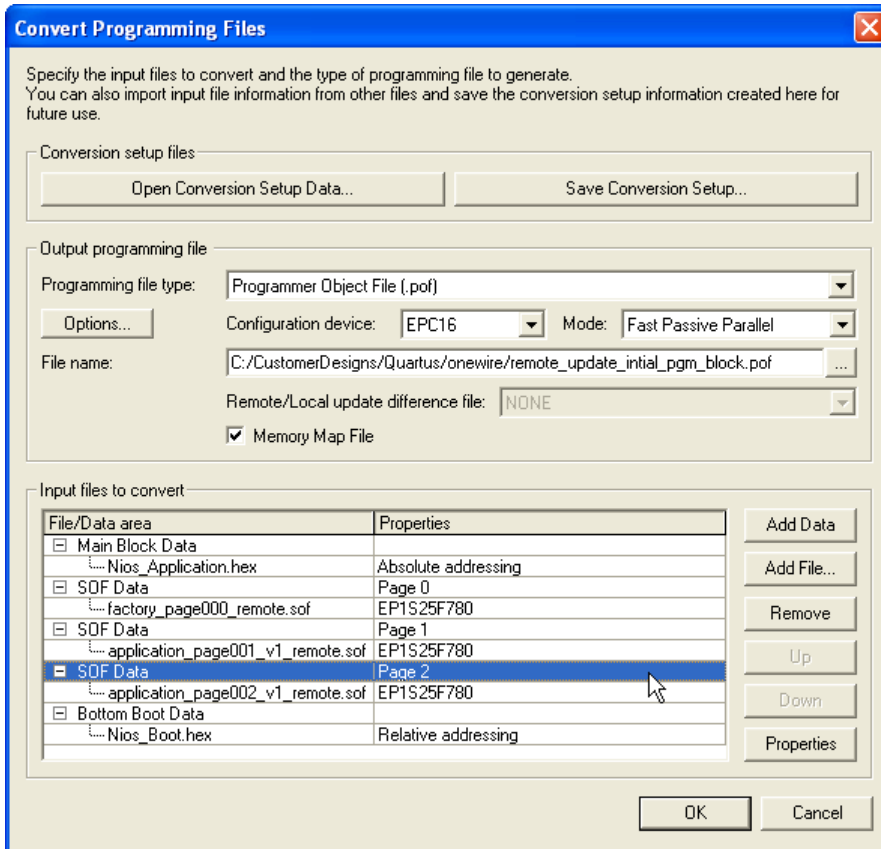
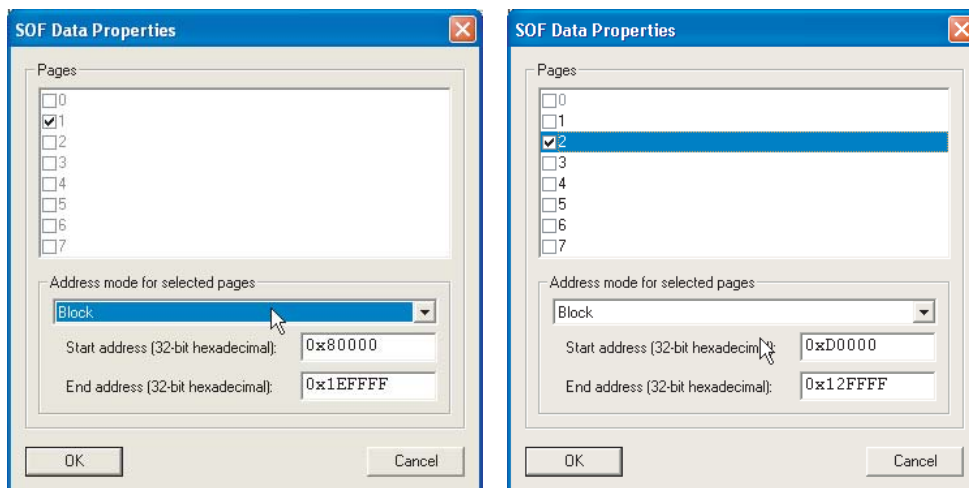


Figure 12–21. Specifying Block Addresses for an Application Configuration

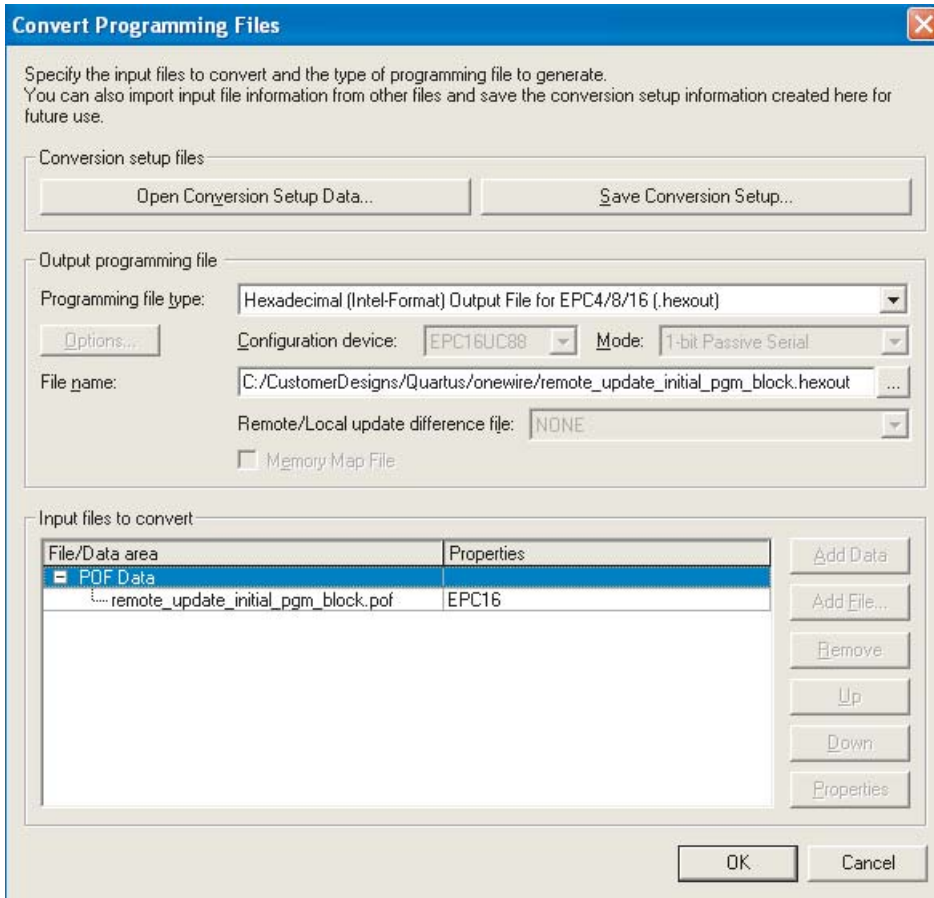
A sample memory map output file for the preceding example is shown below. Note that the allocated memory for page 1 is between 0x00070000 and 0x000BFFFF, while the actual region used by the current application configuration bitstream is between 0x0007B144 and 0x000BFFFF. The configuration data is top justified within the allocated SOF data region. Similarly, the allocated memory for page 2 is between 0x000D0000 and 0x0012FFFF, while the actual region used by the application configuration is between 0x000EB13E and 0x0012FFF9.

Block	Start Address	End Address
BOTTOM BOOT	0x00000000	0x000001FF
OPTION BITS	0x00010000	0x0001003F
PAGE 0	0x00010040	0x00054EFA
PAGE 1	0x0007B144	0x000BFFFF
PAGE 2	0x000EB13E	0x0012FFF9
TOP BOOT/MAIN	0x001F0000	0x001F01FF

Also note that the HEX data stored in the main data area uses absolute addressing. If relative addressing were to be used, the main data contents would be justified with the top (higher address locations) of the memory.

The initial POF can be converted to an Intel Hexadecimal format file (*.HEXOUT) using the Quartus II CPF utility. See [Figure 12–22](#).

Figure 12–22. Converting POF Programming File to Intel HEX Format



Partial Programming File Generation

In remote update mode, the Quartus II CPF utility allows an existing application configuration page to be replaced with new data, or a new application configuration to be added. Partial programming files are generated to perform such configuration data updates.

In order to generate a partial programming file, you have to input the initial POF and new configuration data (SOF) to the Quartus II CPF utility. In addition, you have to specify the addressing mode (auto or manual) that was used during initial POF creation. And if block addressing was used, you should specify the block start and end

addresses. With this information, Quartus II ensures that the partial POF only updates the flash region containing the application configuration. The factory configuration (page 0) and configuration option bits are left unaltered during this process. The only exception is when a new application configuration is added, the configuration options bits are updated to include start/end addresses for the new page. All existing page addresses and other configuration options bits remain unchanged.

Figure 12–23 and the following steps illustrate generation of a partial programming file to replace the most recent application configuration. In this example, the initial programming file contained one factory and two application configurations. Hence, the page 2 application configuration is being updated with new data.

1. Open the **Convert Programming Files** window from the **File** menu.
2. Select **Programmer Object File for Remote Update (*.pof)** from the drop-down list titled **Programming file type**, and specify an output file name.
3. In the **Input files to convert** box, highlight **POF Data** and click **Add File**. Select the initial programming POF file for this design and insert it.
4. In the **Input files to convert box**, highlight **SOF Data** and click **Add File**. Select the new application configuration bitstream (SOF) and insert it.
5. When using block addressing, select the **SOF Data** entry for **Page 2**, and click **Properties**. This opens the **SOF Data Properties** dialog box (see Figure 12–24 on page 12–42).
6. Pick **Block** from the **Address Mode** drop down selection, and enter 32-bit Hexadecimal byte address for block **Starting Address** and **Ending Address**. These addresses should be identical to those used to generate the initial programming file. Click **OK** to save SOF data properties.
7. Check the **Memory Map File** box to generate a memory map output file that specifies the start/end addresses of the new application configuration data in page 1.
8. Pick a remote update difference file from the **Remote/Local Update Difference File** drop-down menu. You can select between an Intel HEX, JAM, JBC, and POF output file types. The output file name is the same as the POF output file name with a **_dif** suffix.

9. Save the CPF setup (optionally), by selecting **Save Conversion Setup...** and specifying a name for the COF output file.
10. Click **OK** to generate initial programming and memory map files.

Figure 12–23. Remote Update Partial Programming File Generation

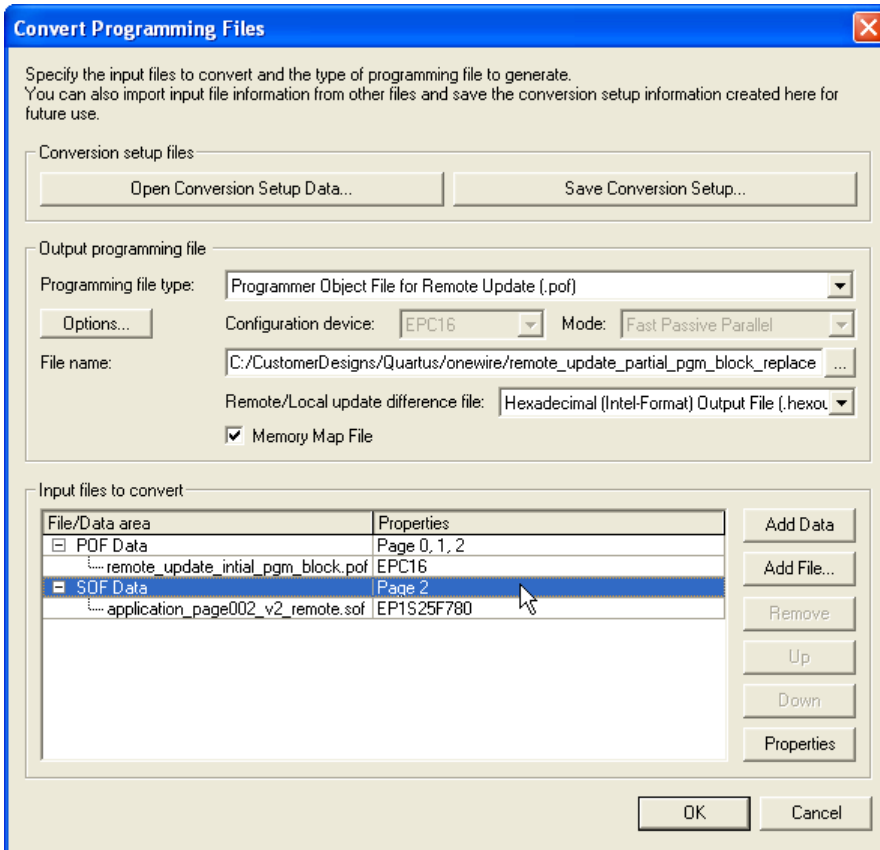
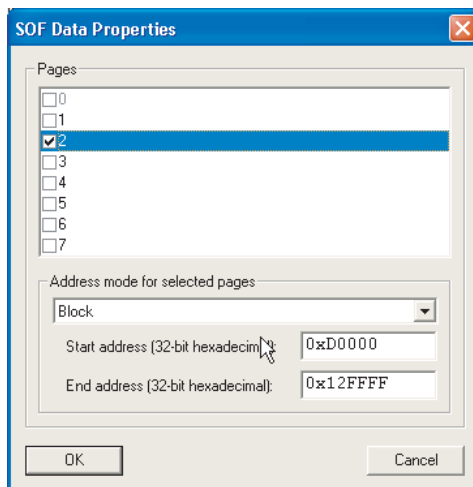


Figure 12–24. Specifying Block Addresses for Application Configuration



For adding a new application configuration, follow the steps listed above with one modification. In Step 5, select **SOF Data** and click on **Properties**. In the **SOF Data Properties** dialog box, select a new page (for example, page 3) and specify the addressing mode information. Continue with steps 7 through 10. When a new page is added, the memory map output file lists the start/end addresses for this page. A sample is shown below:

Block	Start Address	End Address
OPTION BITS	0x00010000	0x0001003F
PAGE 3	0x0012FFFA	0x00174EB4

Combining MAX Devices & Flash Memory

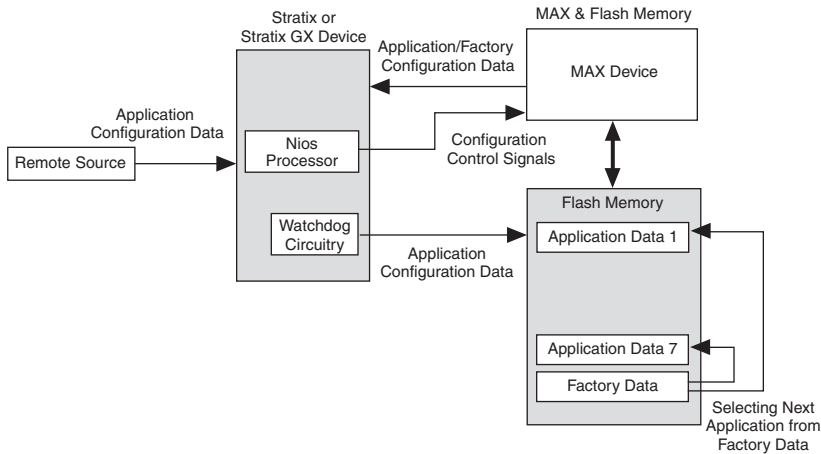
This section describes remote system configuration with the Stratix or Stratix GX device and the Nios embedded processor, using a combination of MAX® devices and flash memory.

You can use MAX 3000 or MAX 7000 devices and an industry-standard flash memory device instead of enhanced configuration devices. In this scheme, flash memory stores configuration data, and the MAX device controls reading and writing to the flash memory, keeping track of address locations.

The MAX device determines which address location and at what length to store configuration data in flash memory. The Nios embedded processor, running in the Stratix or Stratix GX device, receives the

incoming data from the remote source and writes it to the address location in flash memory. The Nios embedded processor initiates loading of factory configuration into the Stratix or Stratix GX device. Figure 12–25 shows remote system configuration using a MAX device and flash memory combination.

Figure 12–25. Remote System Configuration Using a MAX Device & Flash Memory



You can use both remote and local configuration modes in this scheme. You should specify a default page for factory configuration and make sure it is not altered or removed at any time. In remote system configuration mode, PS, FPP, and PPA modes are supported when configuring with MAX and flash devices.

Using an External Processor

This section describes remote system configuration with Stratix or Stratix GX devices and the Nios embedded processor, using an external processor and flash memory devices.

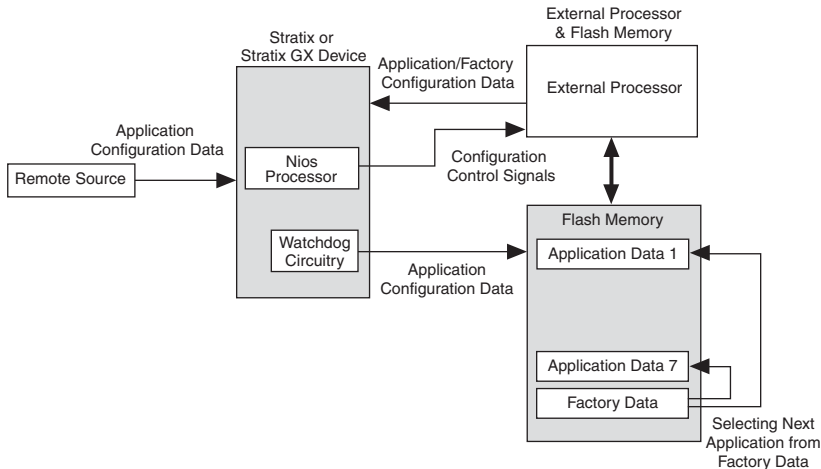
In this scheme, the external processor and flash memory device replace the enhanced configuration device. Flash memory stores configuration data, and the processor controls reading and writing to the flash memory and also keeps track of the address location. This type of remote system configuration supports PS, FPP, and PPA modes.

The processor determines at which address which length to store the configuration data in flash memory. The Nios embedded processor receives the incoming data from a remote source and writes it to the address location in the flash memory, and then initiates loading of factory

configuration data into the Stratix or Stratix GX device. Figure 12–26 shows the remote system configuration using a Nios embedded processor and flash memory.

You can use both remote and local configuration modes in this scheme. You should specify a default page for factory configuration and make sure it is not altered or removed at any time.

Figure 12–26. Remote System Configuration Using External Processor & Flash Memory



Conclusion

Stratix and Stratix GX devices are the first PLDs with dedicated support for remote system configuration. By allowing real-time system upgrades from a remote source, you can use Stratix and Stratix GX devices in a variety of applications that require automatic configuration updates. With the built-in watchdog timer circuitry, Stratix and Stratix GX devices avoid incorrect or erroneous states. Using Stratix and Stratix GX devices with remote system configuration enhances design flexibility and reduces time to market.