12. Remote System Configuration with Stratix & Stratix GX Devices

S52015-3.1

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

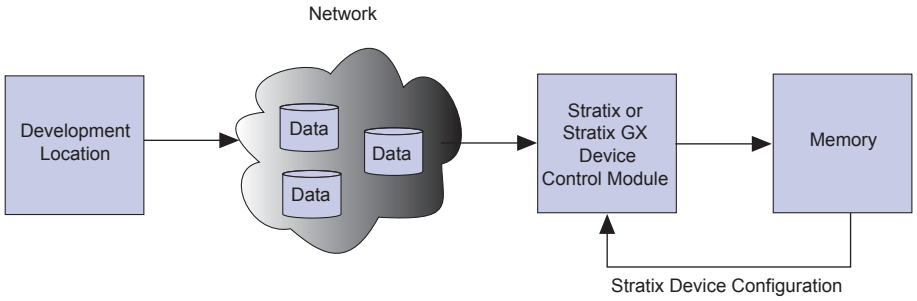
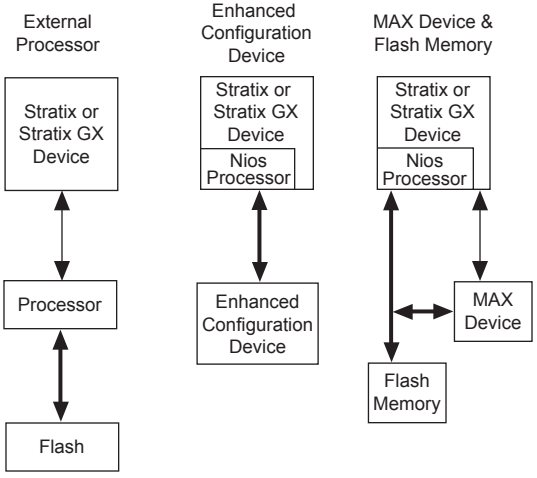


Figure 12–2. Different Options for Remote System Configuration



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 Note (1)

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.

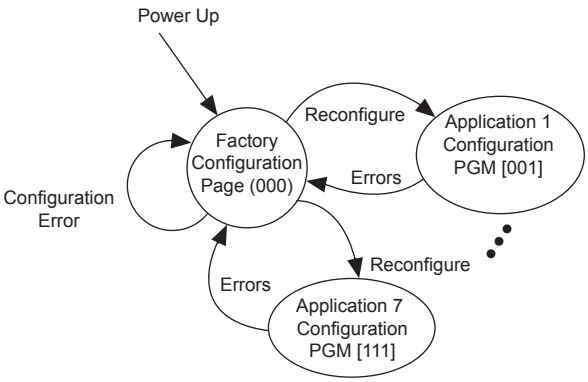
If more than seven files are sent to a system using remote configuration mode, previous files are overwritten.

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

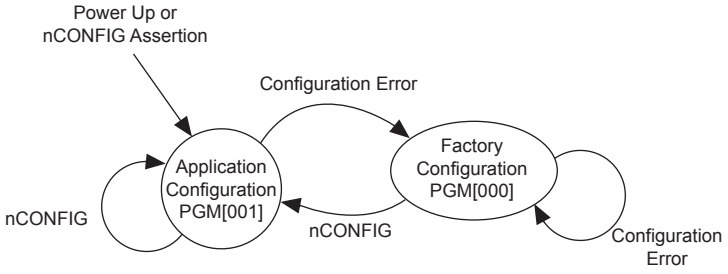
Local configuration mode—a simplified version of remote configuration mode—is suitable for systems that load an application immediately upon power-up. In this mode you can only use one application configuration, which you can update either remotely or locally.

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:

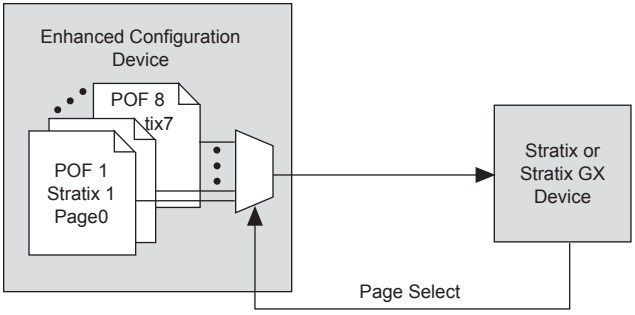
- Page mode feature
- Factory configuration
- Application configuration
- Watchdog timer
- Remote update sub-block
- Remote configuration registers

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.

Table 12–2. Remote Configuration Registers

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

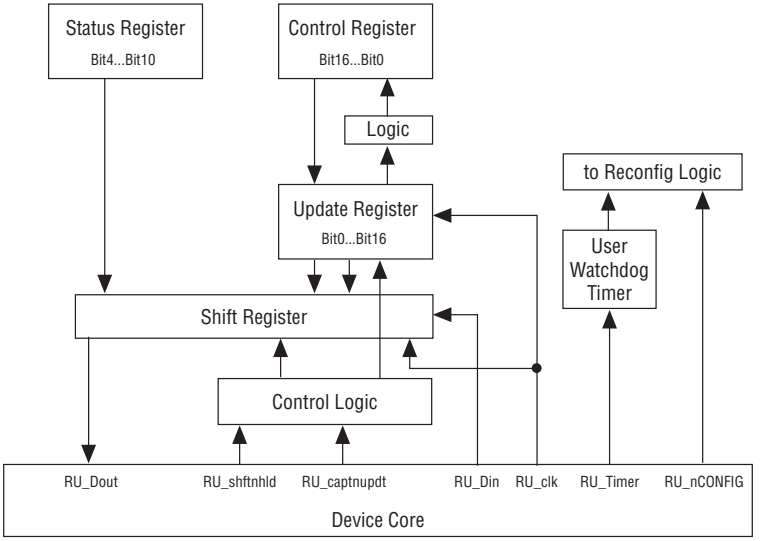


Table 12–3 describes the user configuration signals that are driven to/from the device logic array. The remote configuration logic has one input signal to the device logic array and six output signals from the device logic array.

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_shftnhld	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. It 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_shftnhld 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_shftnhld 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_shftnhld 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_shftnhld 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 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

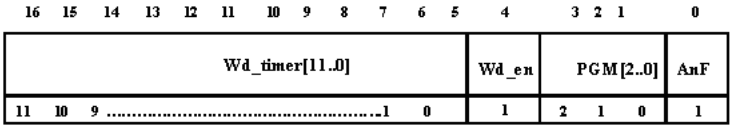


Table 12–4 shows the content of the control register upon POR.

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

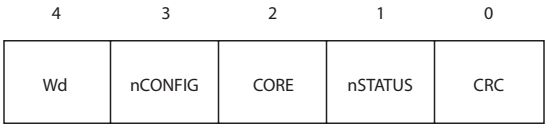


Table 12–5 shows the content of the status register upon POR.

Table 12–5. Status Register Contents

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

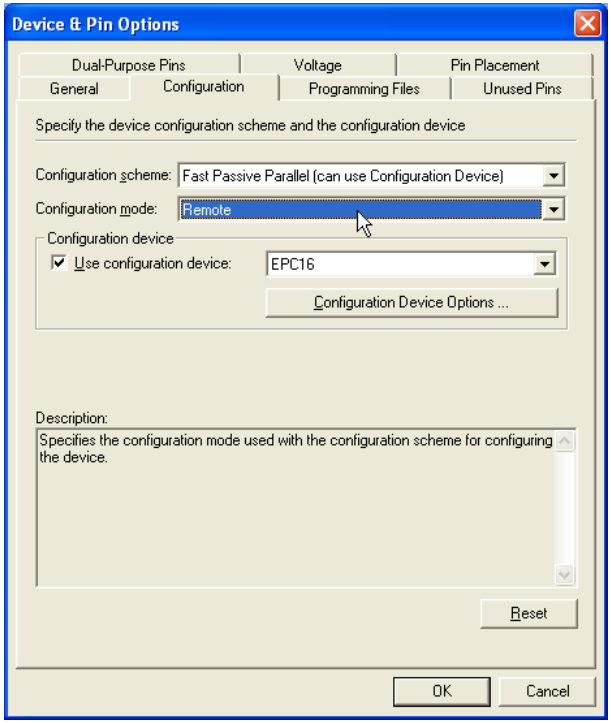
The Quartus II software supports implementation of both remote and local configuration modes in your Stratix or Stratix II device. To include the remote or local configuration feature to your design, select remote or local as the configuration mode under the **Device & Pin Options** compiler settings (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.

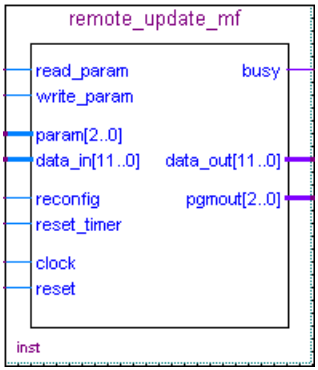
Figure 12–9. Device & Pin Options Dialog Box



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 be 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 `altremote_update` 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 `read_param` 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 `param[]` 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 `param[]` port should be written to the remote update block with the value on `data_in[]`. The number of valid bits on `data_in[]` 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 `write_param` 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 `param[]` and `data_in[]` are ignored. Once the busy signal is deactivated, the next parameter can be written. This signal is only valid when the `Current_Configuration` 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 `param[]` input and by driving the `write_param` 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 `Current_Configuration` 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 `data_in[]`, `param[]`, and `reconfig` inputs. This signal will go high when `read_param` or `write_param` 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 `param[]` input and by driving the `read_param` 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 `data_out[]` port is only valid after a read_param 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 `data_out[]` 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, `nSTATUS` being pulled low due to an error, the device core caused an error, `nCONFIG` 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 3 bit '000' - Remote configuration	Page mode selection. Writing of this parameter is only allowed when in the factory configuration.
Current configuration (AnF)	101	1	1 bit '0' - Factory 1 bit '1' - Application	Specifies whether the current configuration is factory or and application configuration. This parameter can only be read.
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.

Table 12–9. Remote Update Block Input & Output Ports

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.

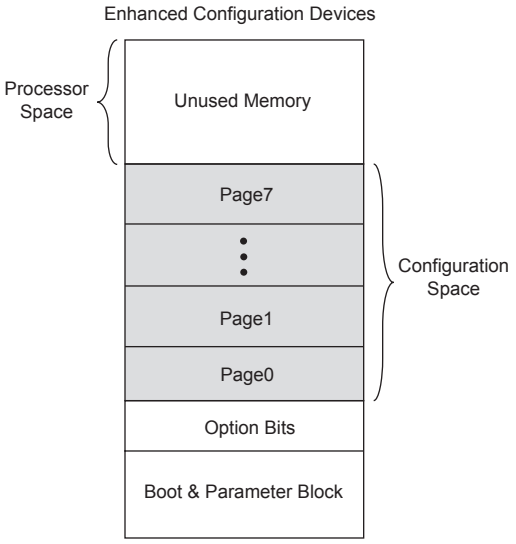
In remote system configuration, only PS and FPP modes are supported using an enhanced configuration device. A Stratix or Stratix GX device running a Nios embedded processor can receive data from a remote source through a network or any other appropriate media. A specific page of the enhanced configuration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

This scheme uses the page mode option in Stratix and Strati 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

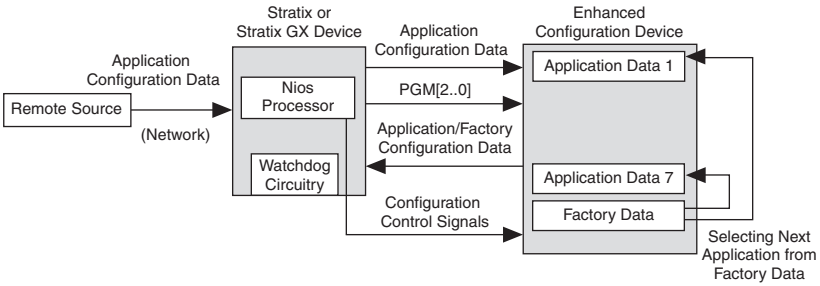


When the Stratix or Stratix GX device powers-up in remote configuration mode, the devices loads configuration data located at page address 000. You should always load the factory default configuration data at this location and make sure this information is not altered.

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.

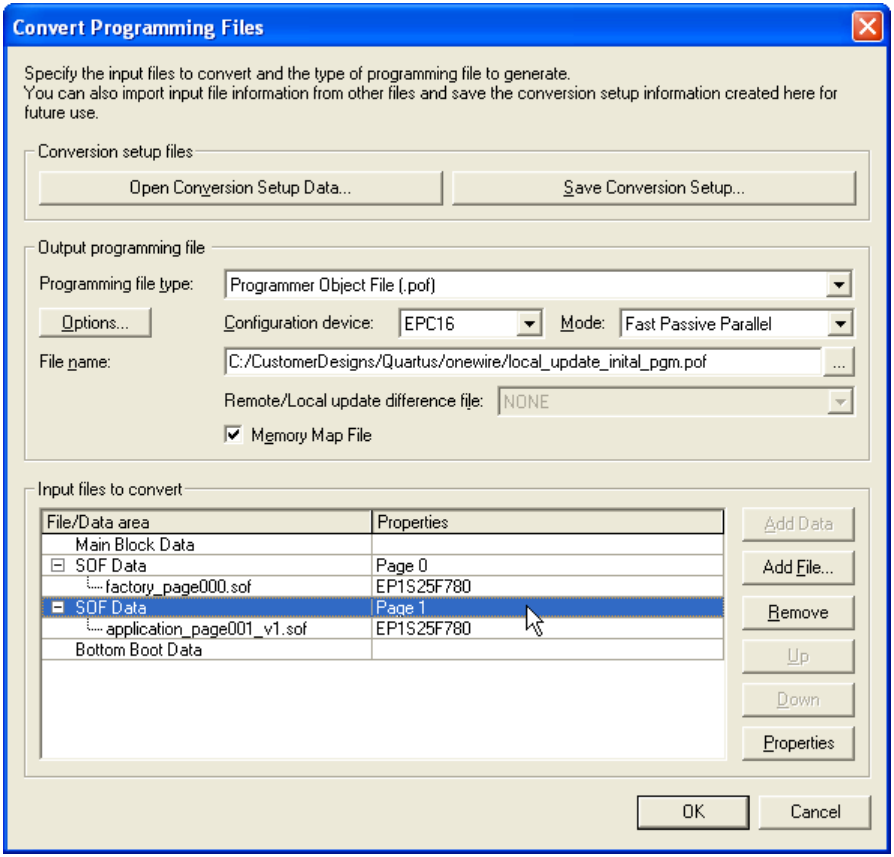
5. Repeat Step 4 for the **Page 1** application configuration page.

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)

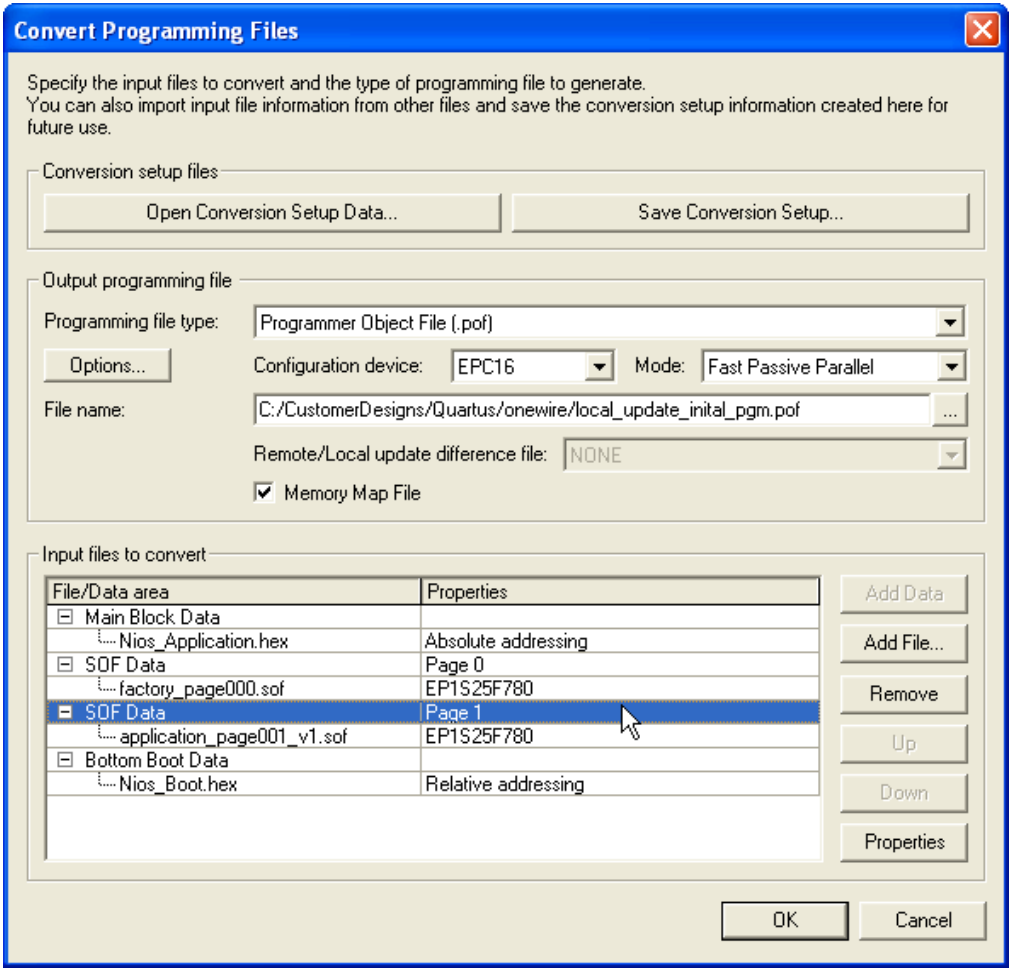
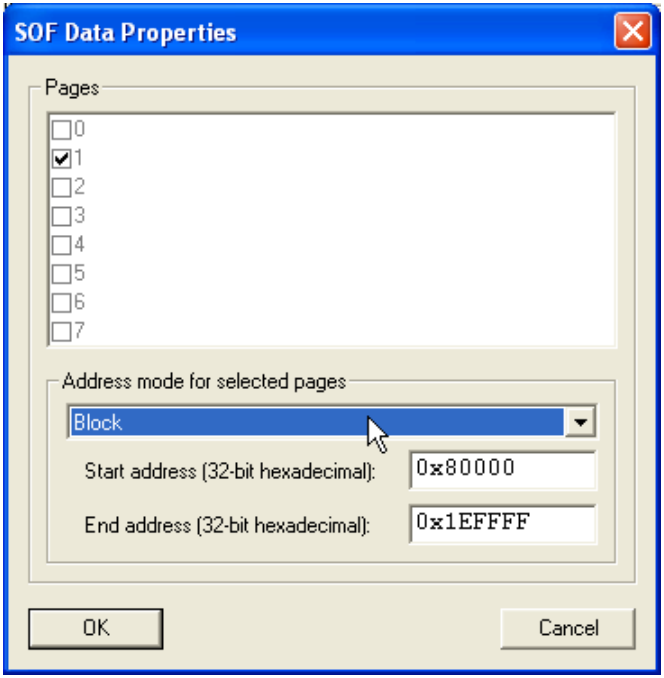


Figure 12–15. Specifying Block Addresses for Application Configuration



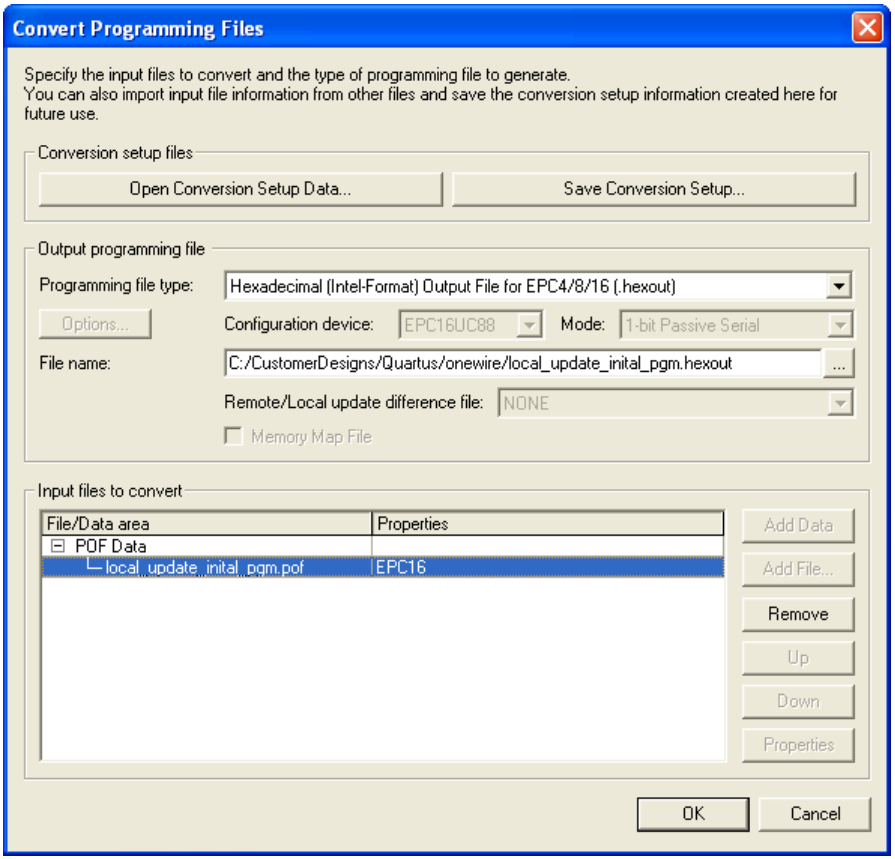
A sample memory map output file for the preceding example is shown below. Note that the allocated memory for page 1 is between 0x00080000 and 0x001EFFFF, while the actual region used by the current application configuration bitstream is between 0x001AB36C and 0x001EFFF7. The configuration data is top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

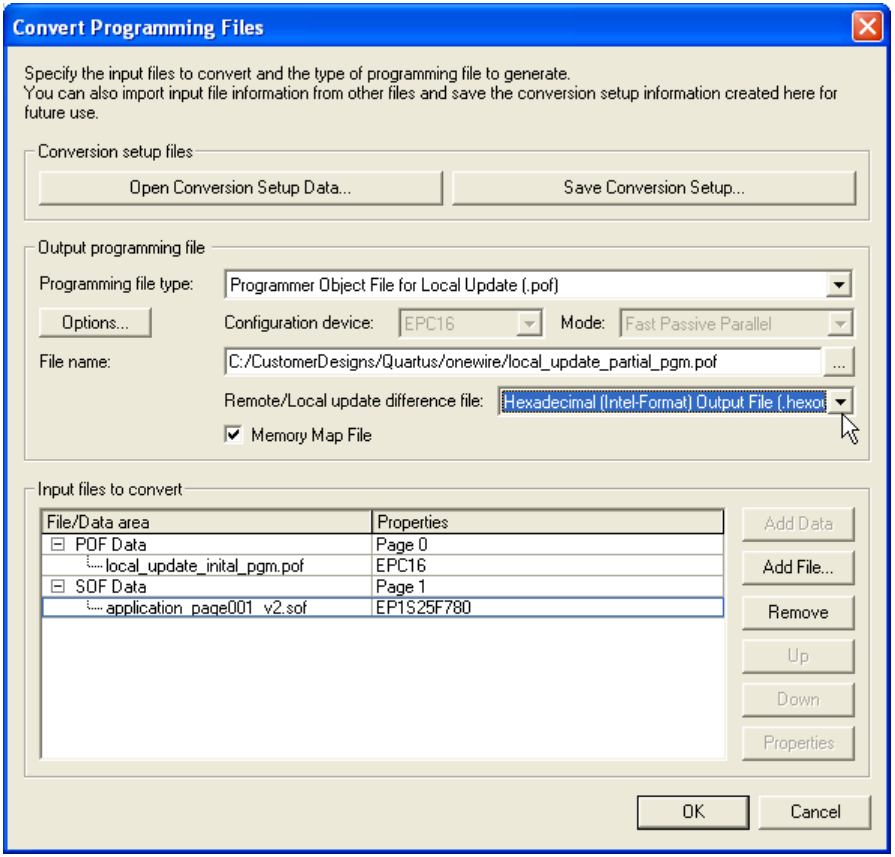
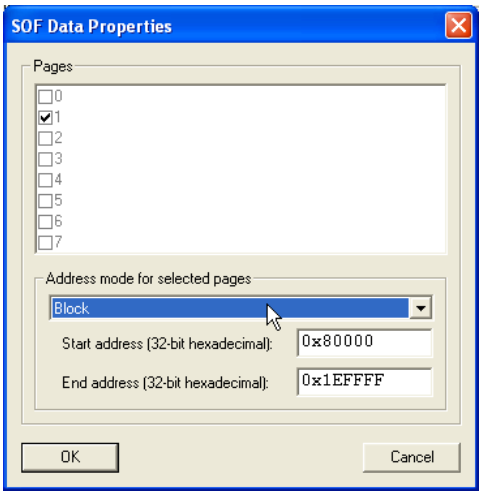


Figure 12–18. Specifying Block Addresses for Application Configuration



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.

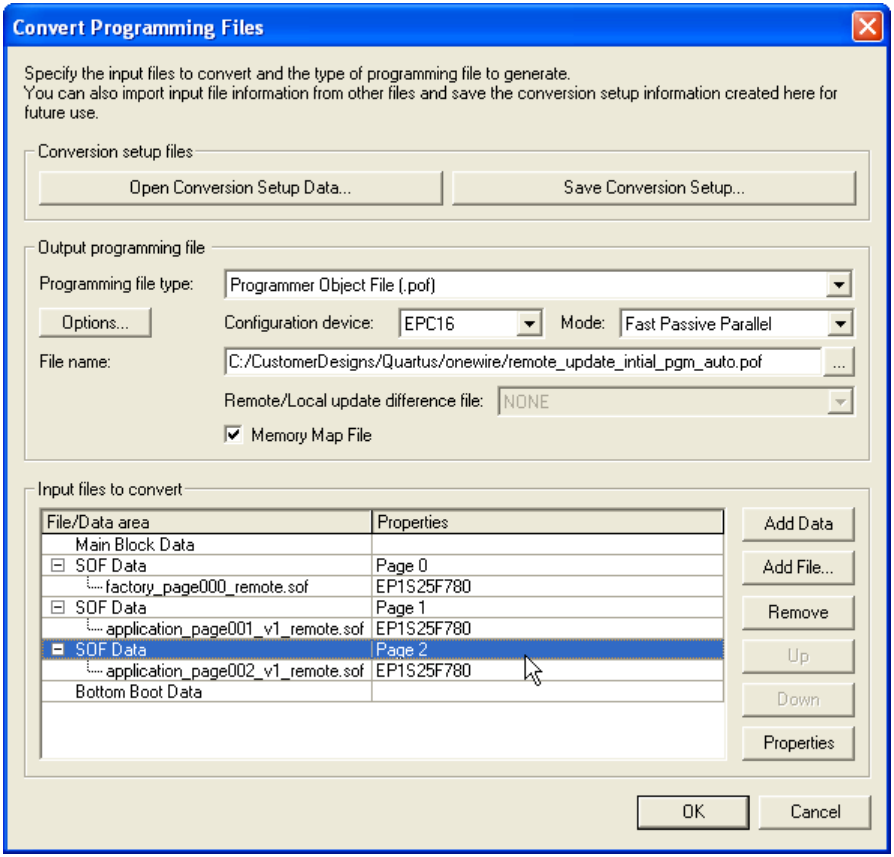
The following steps and screen shot (see Figure 12–19) describe initial programming file generation with auto addressing mode.

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)

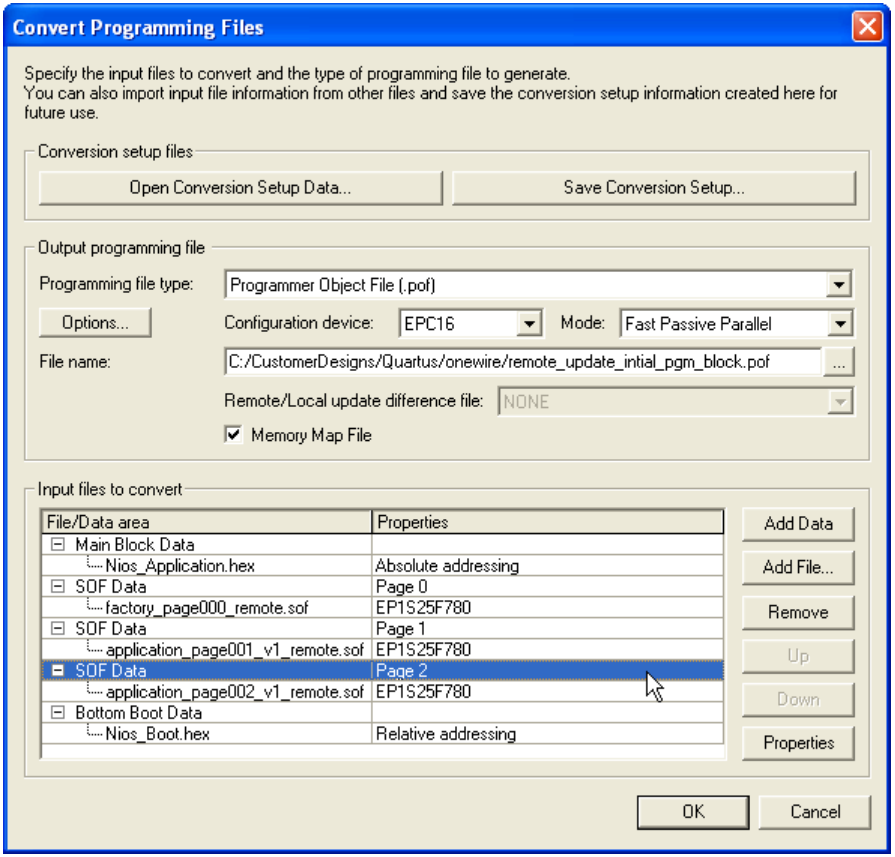
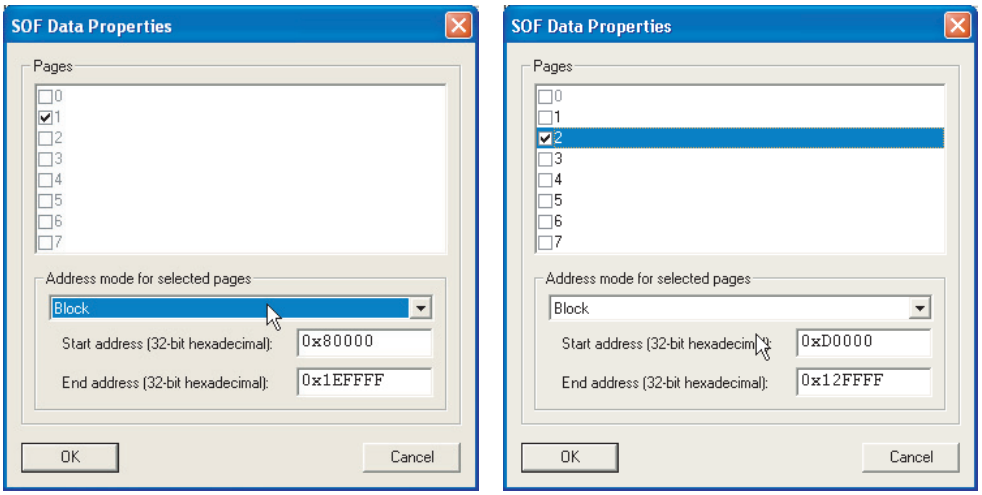


Figure 12–21. Specifying Block Addresses for an Application Configuration



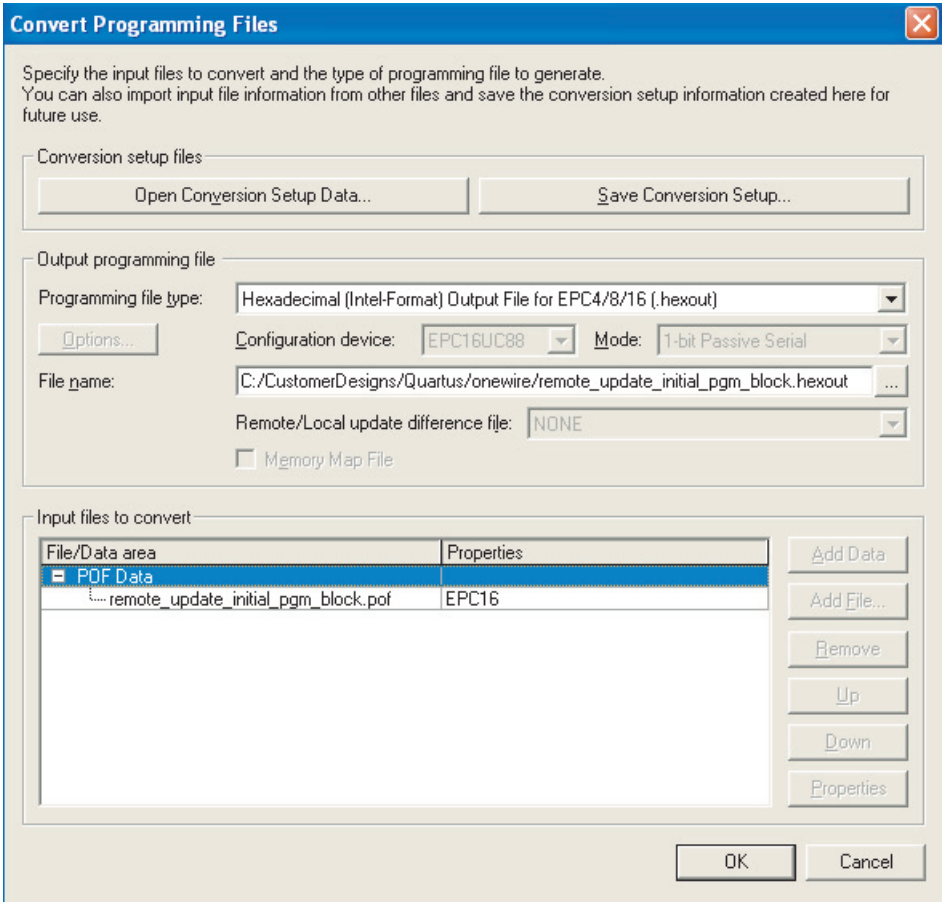
A sample memory map output file for the preceding example is shown below. Note that the allocated memory for page 1 is between `0x00070000` and `0x000BFFFF`, while the actual region used by the current application configuration bitstream is between `0x0007B144` and `0x000BFFFF`. The configuration data 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

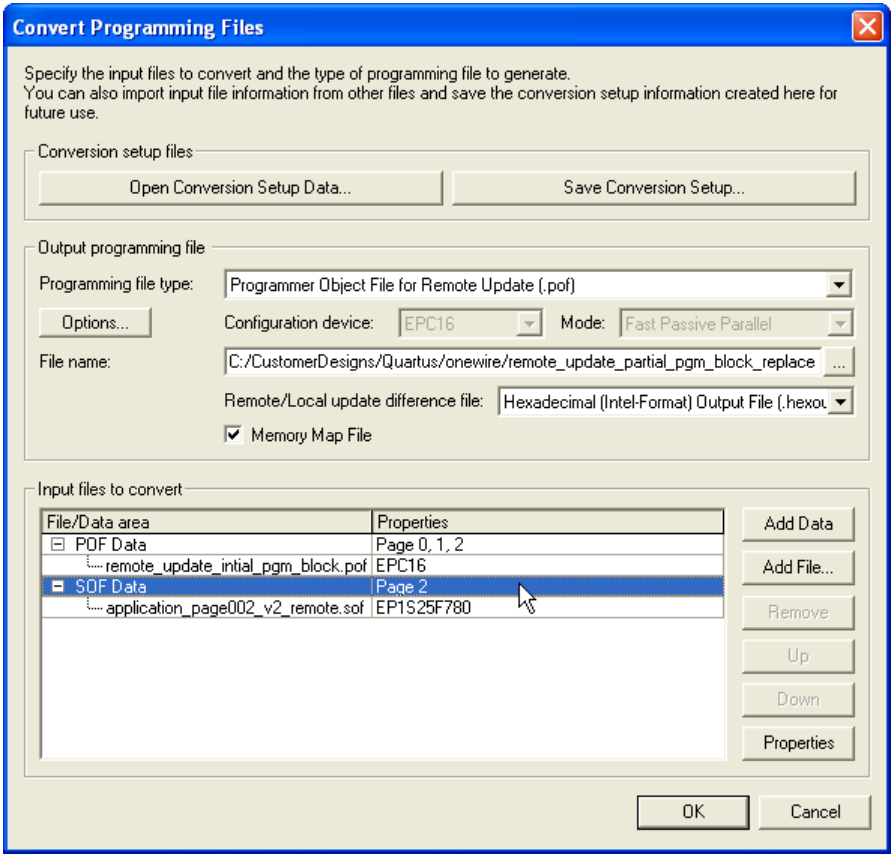
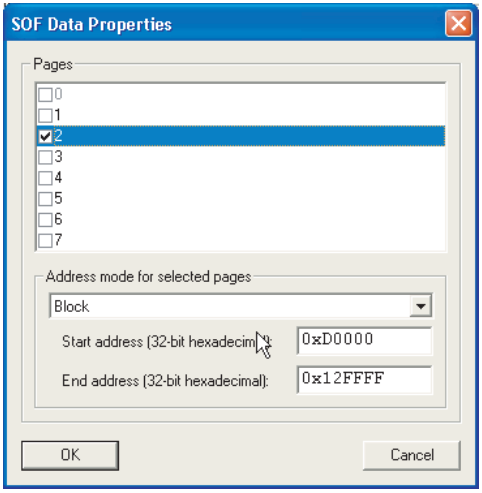


Figure 12–24. Specifying Block Addresses for Application Configuration



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

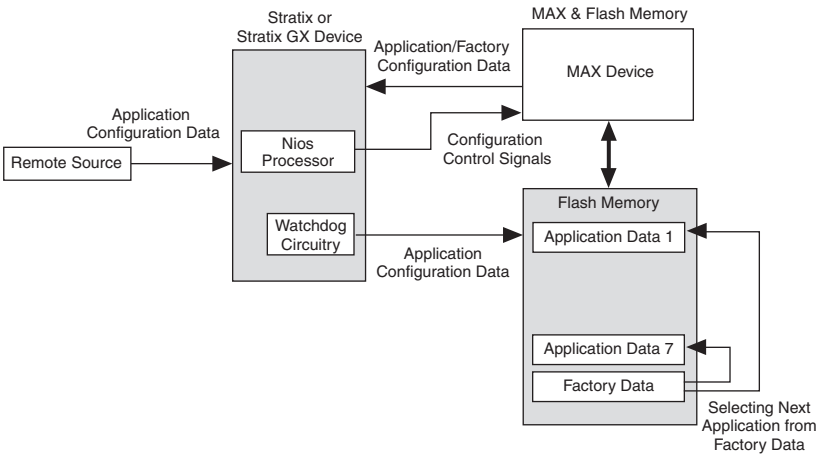
This section describes remote system configuration with the Stratix or Stratix GX device and the Nios embedded processor, using a combination of MAX® devices and flash memory.

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.

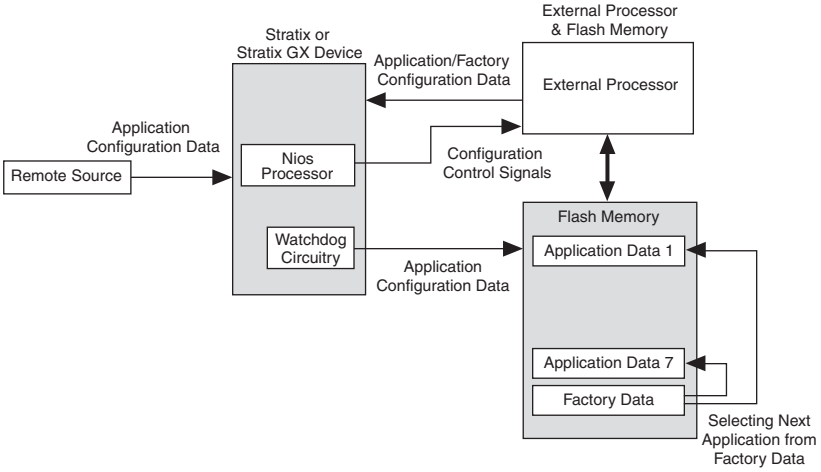
In this scheme, the external processor and flash memory device replace the enhanced configuration device. Flash memory stores configuration data, and the processor controls reading and writing to the flash memory and also keeps track of the address location. This type of remote system configuration supports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