Parallel Flash Loader Megafunction User Guide

UG-01082-2.0 User Guide

This user guide describes the parallel flash loader (PFL) megafunction and provides information about programming flash memory, configuring your FPGA from the flash memory, and instantiating the PFL megafunction in the Quartus® II software. As the density of FPGAs increases, larger configuration storage is required. If your system contains a flash memory device, you can use your flash memory as the FPGA configuration storage as well. You can use the PFL megafunction in Altera® CPLDs (MAX® II and MAX V devices) or FPGAs to program flash memory devices efficiently through the JTAG interface and to control configuration from the flash memory device to the Altera FPGA.

This user guide contains the following sections:

Features

You can use the PFL megafunction for the following functions:

- Programs Common Flash Interface (CFI) flash, quad Serial Peripheral Interface (SPI) flash, or NAND flash memory devices with the device JTAG interface.
- Controls Altera FPGA configuration from a CFI flash, quad SPI flash, or NAND flash memory device for ACEX® 1K, APEX™ 20K, APEX II, Arria series, Cyclone series, FLEX® 10K, and Stratix series FPGA devices.

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Subscribe

Device Support

This user guide focuses on implementing the PFL megafunction in an Altera CPLD. The PFL megafunction supports all Altera FPGAs. You can implement the PFL megafunction in an Arria®, Cyclone®, or Stratix® device family FPGA to program flash memory or to configure other FPGAs.

For more information about using an FPGA-based PFL to program a flash memory device, refer to *AN478: Using FPGA-Based Parallel Flash Loader with the Quartus II Software*.

Supported Flash Memory Devices

The Quartus II software generates the PFL megafunction logic for the flash programming bridge and FPGA configuration.

Table 1 lists the types of CFI flash memory devices that the PFL megafunction supports.

If your CFI device is not listed in Table 1, but is compatible with Intel or Spansion CFI flash, Altera recommends selecting **Define CFI Flash Device** in the Quartus II software.

Table 1. Flash Memory Device Supported by the CFI (Part 1 of 3)

Manufacturer	Device Name (1), (2)	Density (Megabit)	Data Width
Numonyx	28F800C3	8	16 bits
	28F160C3	16	
	28F320C3	32	
	28F640C3	64	
	28F320J3	32	8 or 16 bits
	28F640J3	64	
	28F128J3	128	

Table 1. Flash Memory Device Supported by the CFI (Part 2 of 3)

Manufacturer	Device Name (1), (2)	Density (Megabit)	Data Width
Numonyx	28F640P30	64	16 bits
	28F128P30	128	
	28F256P30	256	
	28F512P30	512	
	28F00AP30	1000	
	28F640P33	640	
	28F128P33	128	
	28F256P33	256	
	28F512P33	512	
	28F00AP33	1000	
	28F512M29EW	512	8 or 16 bits
	28F256M29EW	256	
	28F00AM29EW	1000	
	JS29F256J3	256	16 bits
	M29W256G		8 or 16 bits
	M29W640F	64	
	M28W160CT	16	8 or 16 bits
	M28W160CB		
	M29W160F7		
	M29W160FB		
	M29W320E	32	
	M29W320FT		
	M29W320FB		
	M29DW323DT		
	M29DW323DB		
	M29W640G	64	
	M29W128G	128	
	M58BW16FT	16	32 bits
	M58BW16FB		
	M58BW32FB	32	16 or 32 bits
	M58BW32FT		32 bits

Table 1. Flash Memory Device Supported by the CFI (Part 3 of 3)

Manufacturer	Device Name (1), (2)	Density (Megabit)	Data Width
Spansion	S29GL128P (3)	128	8 or 16 bits
	S29GL256P (3)	256	
	S29GL512P (3)	512	
	S29GL01GP	1024	
	S29AL016D	16	
	S29AL032D	32	
	S29AL016J	16	
	S29AL016M	16	
	229JL032H	32	
	229JL064H	64	
	S29WS128N	128	16 bits
Eon Silicon Solution	EN29LV160B	16	16 bits
	EN29LV320B	32	
	EN29GL128	128	
Macronix	MX29LV160D	16	16 bits
	MX29LV320D	32	
	MX29LV640D	64	
	MX29LV640E	64	
	MX29GL128E	128	
	MX29GL256E	256	

Notes to Table 1:

(1) The Spansion S29GL-N flash memory device family has been discontinued. Altera does not recommend using this flash memory device. For more information about an alternative recommendation, refer to the Spansion website (www.spansion.com).

(2) The PFL megafunction supports top and bottom boot block of the flash memory devices. For Numonyx flash memory devices, the PFL megafunction supports top, bottom, and symmetrical blocks of flash memory devices.

(3) Supports page mode.

Table 2 lists the types of quad SPI flash memory devices that the PFL megafunction supports.

Table 2. Types of Supported Quad SPI Flash Memory Device

Manufacturer	Device Name	Density (Megabit)
Macronix	MX25L8035E	8
	MX25L8036E	
	MX25U8035	
	MX25U8035E	
	MX25V8035	
Macronix	MX25L1635D	16
	MX25L1635E	
	MX25L1636D	
	MX25L1636E	
	MX25U1635E	
	MX25L3225D	32
	MX25L3235D	
	MX25L3235D	
	MX25L3236D	
	MX25L3237D	
	MX25U3235E	
	MX25L6436E	64
	MX25L6445E	64
	MX25L6465E	
	MX25U6435E	
	MX25L12836E	128
	MX25L12845E	
	MX25L12865E	
	MX25L25635E	256
	MX25L25735E	
Spansion	S25FL032P	32
	S25FL064P	64
	S25FL129P	128
Numonyx	N25Q128	128

Table 3 lists the types of NAND flash memory devices that the PFL megafunction supports.

Table 3. Types of Supported NAND Flash Memory Device

Manufacturer	Device Name	Density (Megabit)
Numonyx	NAND512W3A2	512
	NAND512R3A2	
Samsung	K9F1208R0C	
Hynix	HY27US0812(1/2)B	
Toshiba	TC58DVG02A1	1000

The PFL megafunction allows you to configure the FPGA in Passive Serial (PS) or Fast Passive Parallel (FPP) mode. The PFL megafunction supports configuration with FPGA on-chip data compression and data encryption. When you use compressed or encrypted configuration data for FPP configuration, the PFL megafunction holds one data byte for four `DCLK` cycles to ensure the `DCLK` frequency runs at ×4 data rate. The PFL megafunction checks if the compression or encryption feature is turned on in the configuration image before configuring in FPP mode. Hence, no additional setting is required in the PFL megafunction to specify whether the configuration file stored in the flash memory device is a compressed or uncompressed image.

☞ When you turn on the enhanced bitstream compression feature, data encryption is disabled.

You can program the Altera CPLDs and flash memory device in Programmer Object File (**.pof**), Jam™ Standard Test and Programming Language (STAPL) Format File (**.jam**), or JAM Byte Code File (**.jbc**) file format. The PFL megafunction does not support Raw Binary File (**.rbf**) format.

Logic element (LE) usage for the PFL megafunction varies with different PFL megafunction and Quartus II software settings. To determine the exact LE usage number, compile a PFL design with your settings using the Quartus II software.

Functional Description

The PFL megafunction allows you to program flash memory devices with Altera CPLDs through the JTAG interface and provide the logic to control configuration from the flash memory device to the Altera FPGA.

Programming Flash Memory

You can use the PFL megafunction to program the following flash memory devices with JTAG interface:

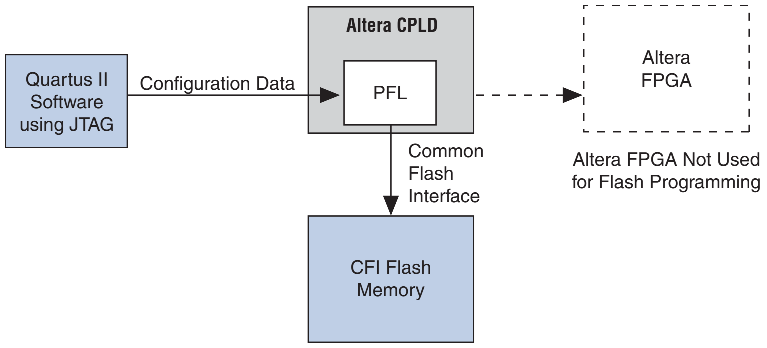
- Programming CFI Flash
- Programming Quad SPI Flash
- Programming NAND Flash

Programming CFI Flash

Altera configuration devices support programming through the JTAG interface to allow in-system programming and updates. However, standard flash memory devices do not support the JTAG interface. You can use the JTAG interface in Altera CPLDs to indirectly program the flash memory device. The Altera CPLD JTAG block interfaces directly with the logic array in a special JTAG mode. This mode brings the JTAG chain through the logic array instead of the Altera CPLD boundary-scan cells (BSCs). The PFL megafunction provides JTAG interface logic to convert the JTAG stream provided by the Quartus II software and to program the CFI flash memory devices connected to the CPLD I/O pins.

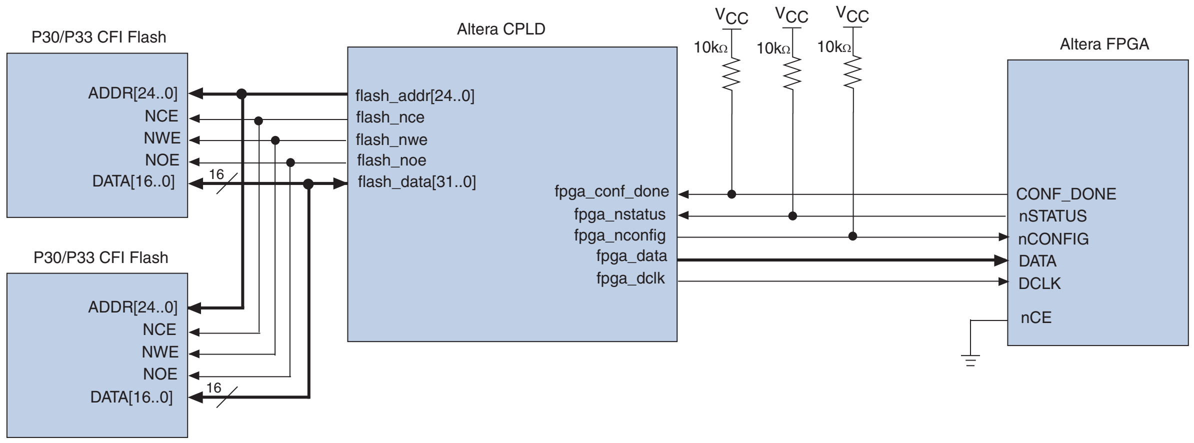
Figure 1 shows an Altera CPLD configured as a bridge to program the CFI flash memory device through the JTAG interface.

Figure 1. Programming the CFI Flash Memory With the JTAG Interface



To achieve faster configuration time, the PFL megafunction supports dual P30 or P33 CFI flash memory devices in burst read mode. In this solution, two identical P30 or P33 CFI flash memory devices are connected to the CPLD in parallel using the same data bus, clock, and control signals (Figure 2). During FPGA configuration, the FPGA `DCLK` frequency is four times as fast than the `flash_clk` frequency.

Figure 2. PFL Megafunction With Dual P30 or P33 CFI Flash Memory Devices



☞ Both flash memory devices used in the dual P30 or P33 CFI flash solution must be identical with the same memory density from the same device family and manufacturer. In the Quartus II software versions 9.1 SP1 and later, dual P30 or P33 flash support is available in the PFL megafunction.

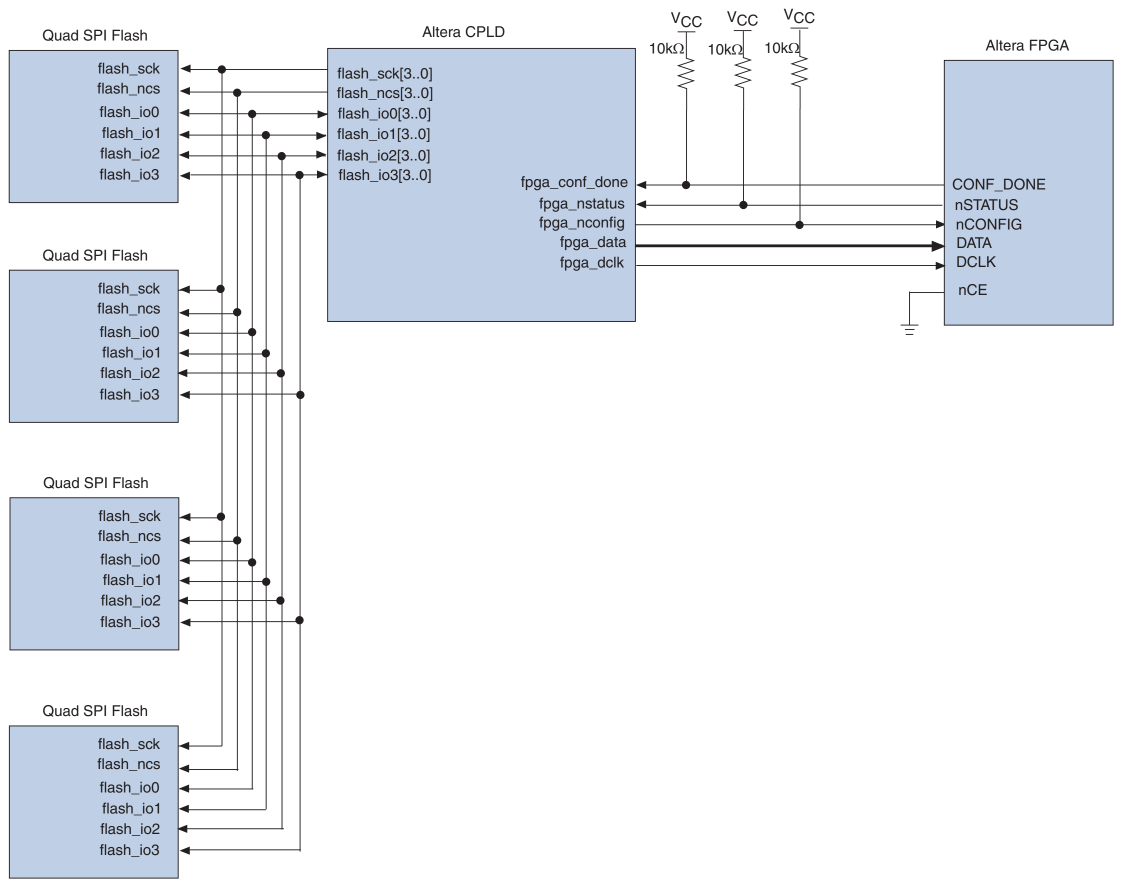
Programming Quad SPI Flash

You can also use the JTAG interface in Altera CPLDs to indirectly program a quad SPI flash memory device with the PFL megafunction. The PFL megafunction instantiated in the Altera CPLD acts as the bridge between the CPLD JTAG programming interface and the quad SPI flash memory device interface connected to the Altera CPLD I/O pins. You can connect as many as four identical quad SPI flashes in parallel to implement more configuration data storage.

☞ When connecting quad SPI flashes in parallel, you must use identical flash memory devices with the same memory density from the same device family and manufacturer. In the Quartus II software version 10.0 and later, quad SPI flash support is available in the PFL megafunction.

Figure 3 shows an Altera CPLD configured as a bridge to program the quad SPI flash memory device through the JTAG interface.

Figure 3. Programming Quad SPI Flash Memory Devices With the CPLD JTAG Interface



Note to Figure 3:

(1) The PFL megafunction supports multiple quad SPI flash programming of up to four devices. For a list of supported quad SPI flash, refer to Table 2 on page 5.

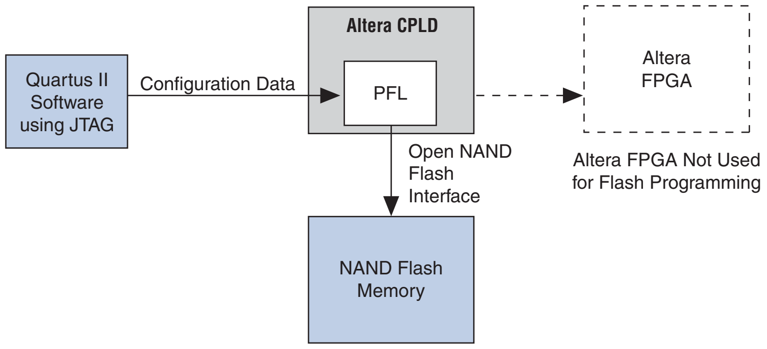
Programming NAND Flash

You can use the JTAG interface in Altera CPLDs to program the NAND flash memory device with the PFL megafunction. Compared to the CFI flash, the NAND flash memory device is a simpler device that has faster erase and write speed with higher memory density.

You can use the JTAG interface in Altera CPLDs to indirectly program the flash memory device. The CPLD JTAG block interfaces directly with the logic array in a special JTAG mode. This mode brings the JTAG chain through the logic array instead of the Altera CPLD BSCs. The PFL megafunction provides JTAG interface logic to convert the JTAG stream provided by the Quartus II software and to program the NAND flash memory device connected to the CPLD I/O pins.

Figure 4 shows an Altera CPLD configured as a bridge to program the NAND flash memory device through the JTAG interface.

Figure 4. Programming NAND Flash Memory Devices With the JTAG Interface

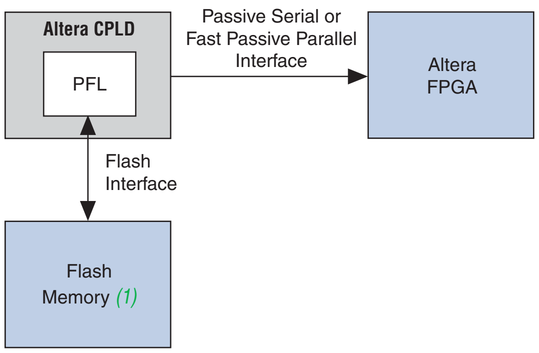

Altera CPLD
Quartus II Software using JTAG
Configuration Data
PFL
Altera FPGA
Open NAND Flash Interface
Altera FPGA Not Used for Flash Programming
NAND Flash Memory

Controlling Altera FPGA Configuration From Flash Memory

You can use the PFL in Altera CPLDs as a configuration controller for FPGA configuration. The PFL logic in the CPLD plays the configuration controller role, and determines when to start the configuration process, read the data from the flash memory device, and configure the Altera FPGA in PS or FPP configuration scheme.

Figure 5 shows the Altera CPLD as the configuration controller for the FPGA.

Figure 5. FPGA Configuration With Flash Memory Data



Note to Figure 5:

(1) CFI flash, quad SPI flash, or NAND flash.

For more information about multi-device FPGA configuration in PS or FPP mode, refer to the Configuration chapter of the respective device handbook.

You can use the PFL megafunction to either program the flash memory devices, configure your FPGA, or both. To perform both functions, create separate PFL functions if any of the following conditions apply to your design:

- You want to use fewer LEs.
- You modify the flash data infrequently.
- You have JTAG or In-System Programming (ISP) access to the Altera CPLD.

- You want to program the flash memory device with non-Altera data. For example, the flash memory device contains initialization storage for an ASSP. You can use the PFL megafunction to program the flash memory device with the initialization data and also create your own design source code to implement the read and initialization control with the CPLD logic.

To create separate PFL functions, follow these steps:

1. To create a PFL instantiation, select **Flash Programming Only** mode.
2. Assign the pins appropriately.
3. Compile and generate a **.pof** for the flash memory device. Ensure that you tri-state all unused I/O pins.
4. To create another PFL instantiation, select **Configuration Control Only** mode.
5. Instantiate this configuration controller into your production design.
6. Whenever you must program the flash memory device, program the CPLD with the flash memory device **.pof** and update the flash memory device contents.
7. Reprogram the CPLD with the production design **.pof** that includes the configuration controller.

☞ All unused pins are set to ground by default. When programming the configuration flash memory device through the CPLD JTAG pins, you must tri-state the FPGA configuration pins common to the CPLD and the configuration flash memory device. You can use the `pfl_flash_access_request` and `pfl_flash_access_granted` signals of the PFL block to tri-state the correct FPGA configuration pins.

The following sections contain important information about using the PFL megafunction:

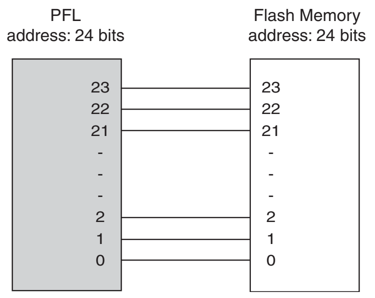
- "Mapping PFL and Flash Address" on page 11
- "Implementing Page in the Flash .pof" on page 13
- "Using Enhanced Bitstream Compression and Decompression" on page 16
- "Using Remote System Upgrade" on page 18

Mapping PFL and Flash Address

Figure 6 through Figure 9 show the address connections between the PFL megafunction and the flash memory device. The address connections vary depending on the flash memory device vendor and data bus width.

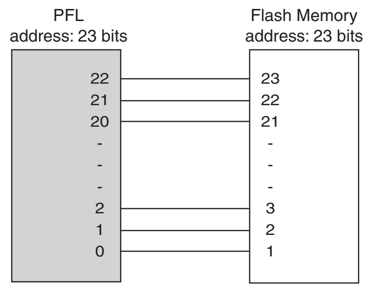
In Figure 6, address connection between the PFL megafunction and the flash memory device are the same.

Figure 6. Numonyx J3 Flash Memory in 8-Bit Mode



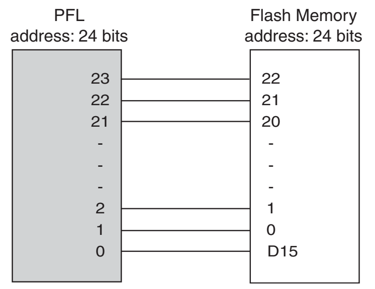
In Figure 7, flash memory addresses in Numonyx J3, P30, and P33 16-bit flash memory are shifted one bit down when compared with the flash addresses in the PFL megafunction. The flash address in the Numonyx J3, P30, and P33 flash memory starts from bit 1 instead of bit 0.

Figure 7. Numonyx J3, P30, and P33 Flash Memories in 16-Bit Mode



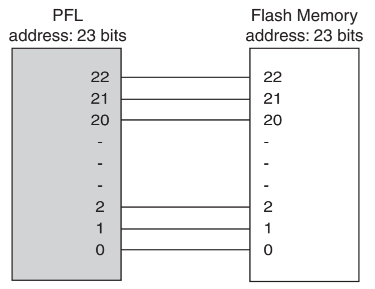
In Figure 8, flash memory addresses in Spansion 8-bit flash shifts one bit up. Address bit 0 of the PFL megafunction connects to data pin D15 of the flash memory.

Figure 8. Spansion and Numonyx M28, M29 Flash Memory in 8-Bit Mode



In Figure 9, address bit numbers in the PFL megafunction and the flash memory device are the same.

Figure 9. Spansion and Numonyx M28, M29 Flash Memory in 16-Bit Mode



Implementing Page in the Flash .pof

The PFL megafunction stores configuration data in a maximum of eight different pages in a flash memory block. Each page holds the configuration data for a single FPGA chain. A single FPGA chain can contain more than one FPGA. For an FPGA chain with multiple FPGAs, multiple SRAM Object File (**.sof)** files are stored in the same page.

The total number of pages allowed and the size of each page depends on the density of the flash. These pages allow you to store designs for different FPGA chains or different designs for the same FPGA chain in different pages.

Use the generated **.sof** files to create a flash memory device **.pof**. When converting these **.sof** files to a **.pof**, use the following address modes to determine the page address:

- Block mode—Allows you to specify the start and end addresses for the page.
- Start mode—Allows you to specify only the start address. The start address for each page is located on an 8-KB boundary. If the first valid start address is 0×000000, the next valid start address is an increment of 0×2000.
- Auto mode—Allows the Quartus II software to automatically determine the start address of the page. The Quartus II software aligns the pages on a 128-KB boundary; for example, if the first valid start address is 0×000000, the next valid start address is an increment of 0×20000.

If you are programming NAND flash, you must specify the NAND flash memory device reserved block start address and the start address to ensure the files reside within a 128-KB boundary.

Storing Option Bits

The PFL megafunction requires you to allocate space in the flash memory device for option bits. The option bits sector contains information about the start address for each page, the **.pof** version used for flash programming, and the Page-Valid bits. You must specify the options bits sector address in the flash memory device when converting the **.sof** files to a **.pof** and creating a PFL design.

Table 4 lists the option bits sector format. Offset addresses 0×00 to 0×1F of the option bits sector store the start addresses for Page 0 to Page 7. Offset address 0×80 stores the **.pof** version required for programming flash memory. This **.pof** version applies to all 8 pages of the configuration data. The PFL megafunction requires the **.pof** version to perform a successful FPGA configuration process.

Table 4. Option Bits Sector Format

Sector Offset	Value
0×00–0×03	Page 0 start address
0×04–0×07	Page 1 start address
0×08–0×0B	Page 2 start address
0×0C–0×0F	Page 3 start address
0×10–0×13	Page 4 start address
0×14–0×17	Page 5 start address
0×18–0×1B	Page 6 start address
0×1C–0×1F	Page 7 start address
0×20–0×7F	Reserved
0×80 *(1)*	**.pof** version
0×81–0×FF	Reserved

Note to Table 4:

(1) **.pof** version occupies only one byte in the option bits sector.

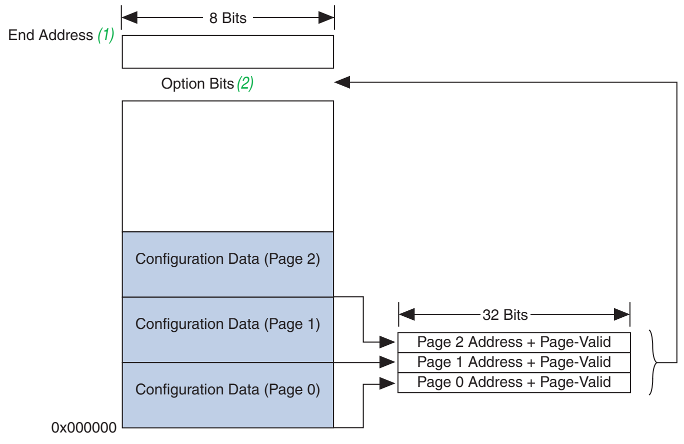
The Quartus II Convert Programming File tool generates the information for the **.pof** version when you convert the **.sof** files to **.pof** files.

The value for the **.pof** version generated by the Quartus II software version 7.1 or later is 0×03. However, if you turn on the enhanced bitstream-compression feature, the value for the **.pof** version is 0×04.

Do not overwrite any information in the option bits sector to prevent the PFL megafunction from malfunctioning, and always store the option bits in unused addresses in the flash memory device.

Figure 10 shows the implementation of page mode and option bits in the CFI flash memory device.

Figure 10. Implementing Page Mode and Option Bits in the CFI Flash Memory Device


8 Bits
End Address (1)
Option Bits (2)
Configuration Data (Page 2)
Configuration Data (Page 1)
Configuration Data (Page 0)
0x000000
32 Bits
Page 2 Address + Page-Valid
Page 1 Address + Page-Valid
Page 0 Address + Page-Valid

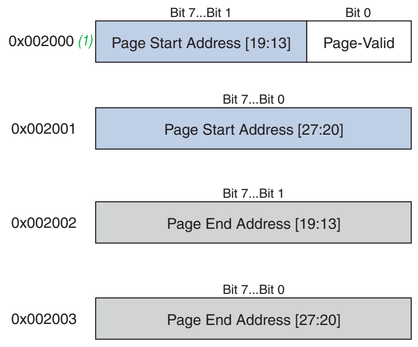
Notes to Figure 10:

(1) The end address depends on the density of the flash memory device. For the address range for devices with different densities, refer to Table 5.

(2) You must specify the byte address for the option bits sector.

Figure 11 shows how the start address and Page-Valid bit for each page are stored in the option bits sector.

Figure 11. Page Start Address, End Address, and Page-Valid Bit Stored as Option Bits



Note to Figure 11:

(1) For flash byte addressing mode.

Bits 0 to 12 for the page start address are set to zero and are not stored as option bits. The Page-Valid bits indicate whether each page is successfully programmed. The PFL megafunction programs the Page-Valid bits after successfully programming the pages.

Table 5 lists the byte address range for CFI flash memory devices with different densities.

Table 5. Byte Address Range

CFI Device (Megabit)	Address Range
8	0×0000000–0×00FFFFF
16	0×0000000–0×01FFFFF
32	0×0000000–0×03FFFFF
64	0×0000000–0×07FFFFF
128	0×0000000–0×0FFFFFF
256	0×0000000–0×1FFFFFF
512	0×0000000–0×3FFFFFF
1024	0×0000000–0×7FFFFFF

Using Enhanced Bitstream Compression and Decompression

The enhanced bitstream compression and decompression feature in the PFL megafunction reduces the size of the configuration file stored in the flash memory device. On average, you can reduce the file size by as much as 50% based on the designs used. When you turn on the enhanced bitstream compression feature, the PFL megafunction disables data encryption.

Table 6 lists the preliminary compressed data size reduction and the configuration time comparison between typical, enhanced, and double bitstream compression results.

Table 6. Comparison Between Typical, Enhanced, and Double Compression

FPGA Configuration	Typical Bitstream Compression Feature	Enhanced Bitstream Compression Feature	Double Compression Technique
FPGA on-chip bitstream decompression enabled	Yes	No	Yes
PFL enhanced bitstream decompression enabled	No	Yes	Yes
Typical configuration file size reduction	35%–55%	45%–75%	40%–60%
PS configuration time	Moderate (1)	Slow	Moderate (1)
FPP configuration time	Fast (2)	Very fast (3)	Not supported

Notes to Table 6:

(1) The FPGA receives compressed bitstream which decreases the time required to transmit the bitstream to the FPGA.

(2) For FPP with on-chip bitstream decompression enabled, the DCLK frequency is ×2, ×4, or ×8 the data rate, depending on the device used. You can check the relationship of the DCLK and data rate in the FPP Configuration section in the configuration chapter of the respective device handbook.

(3) For FPP with enhanced bitstream decompression enabled, the DCLK frequency is ×1 the data rate.

When using the PFL with compression, set the device MSEL pins set for compression or decompression. When generating or converting a programming file, you can enable compression. In the first few bytes during the generation of the programming file (with compression enabled), there is a bit set to notify the PFL that the incoming files is a compressed file. The 4× DCLK-to-data are handled automatically within the PFL.

For the FPP configuration scheme, the enhanced bitstream compression feature helps achieve higher configuration data compression ratio and faster configuration time. For the PS configuration scheme, the double compression technique helps achieve higher configuration data compression ratio and moderate configuration time. To enable the double compression technique, turn on both the typical bitstream compression feature and the enhanced bitstream compression feature in the PFL parameter editor.

Figure 12 shows the configuration data flow when the enhanced bitstream compression feature is used for PS or FPP configuration scheme.

Figure 12. FPGA Configuration With Enhanced Bitstream Compression Feature Enabled

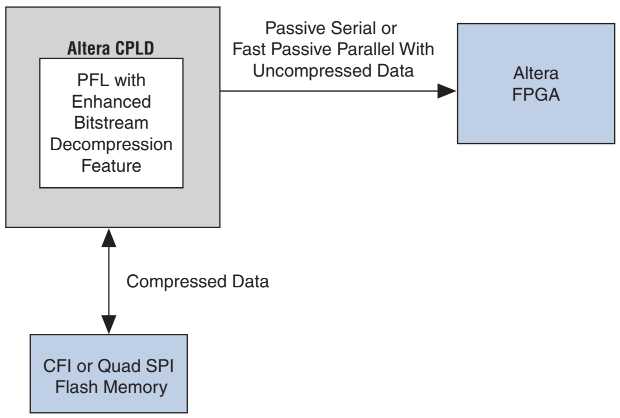
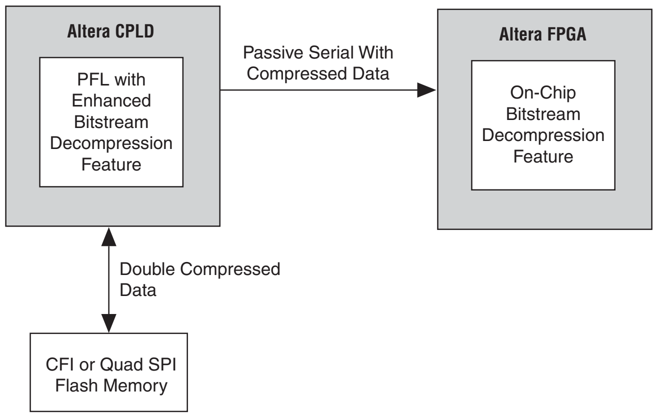

Altera CPLD
PFL with Enhanced Bitstream Decompression Feature
Passive Serial or Fast Passive Parallel With Uncompressed Data
Altera FPGA
Compressed Data
CFI or Quad SPI Flash Memory

Figure 13 shows the configuration data flow when the double compression technique is used for PS configuration scheme.

Figure 13. FPGA Configuration With Double Compression Technique


Altera CPLD
PFL with Enhanced Bitstream Decompression Feature
Passive Serial With Compressed Data
Altera FPGA
On-Chip Bitstream Decompression Feature
Double Compressed Data
CFI or Quad SPI Flash Memory

☞ The enhanced bitstream compression and decompression feature is available in the PFL megafunction in the Quartus II software version 10.0 and later.

For more information about the typical data compression feature, refer to the *Configuration Data Decompression* section in the configuration chapter of the relevant device handbook.

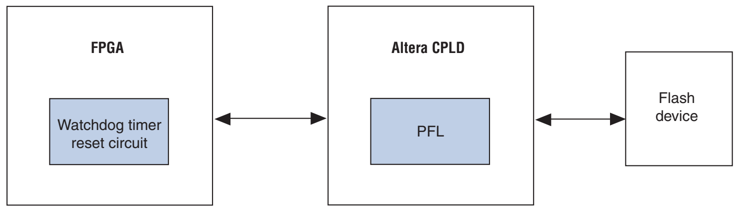
Using Remote System Upgrade

When you instantiate the PFL megafunction in the Altera CPLD for FPP or PS configuration, you can use the features in the PFL megafunction to perform remote system upgrade. You can download a new configuration image from a remote location, store it in the flash memory device, and direct the PFL megafunction to trigger an FPGA reconfiguration to load the new configuration image. You must store each configuration image as a new page in the flash memory device. The PFL megafunction supports a maximum of eight pages.

When using remote system upgrade, the configuration images are classified as a factory image or as application images. A factory image is a user-defined fall-back or safe configuration that performs system recovery when unintended errors occur during or after application image configuration. The factory image is written to the flash memory device only once by the system manufacturer and you must not modify or overwrite the factory image. Application images implement user-defined functionality in the target FPGA and you can remotely update in the system.

Figure 14 shows the block diagrams for implementing a remote system upgrade feature with the PFL megafunction in FPP or PS configuration mode.

Figure 14. Remote System Upgrade With the PFL Megafunction

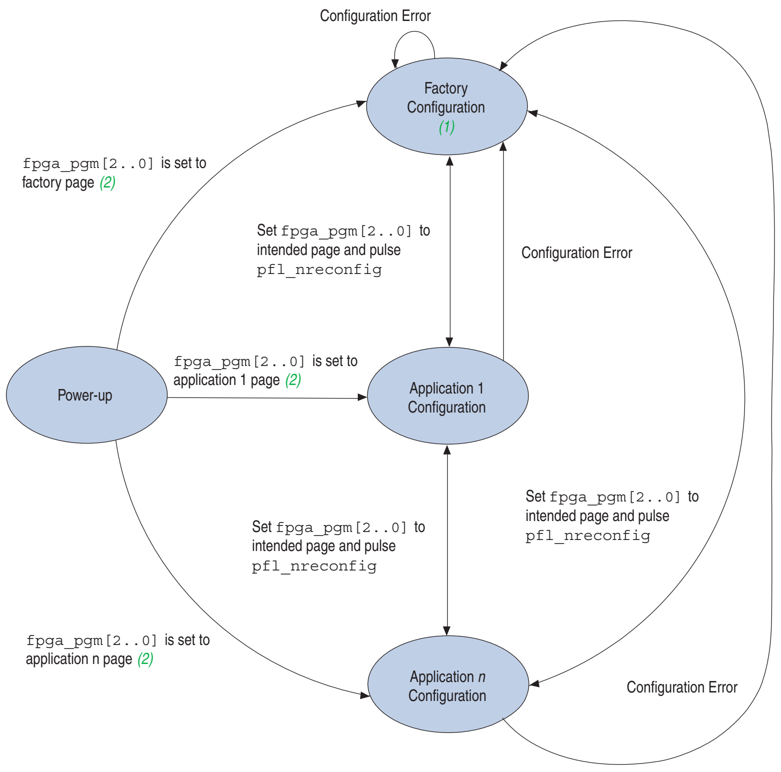


Remote System Upgrade State Machine in the PFL Megafunction

After FPGA powers up, you have the flexibility to determine whether a factory image or any application image is to be loaded by setting the `fpga_pgm[2..0]` input pin to the page in which the intended configuration image is stored. If an error occurs while loading the configuration image, the PFL megafunction triggers a reconfiguration to automatically load the factory image. After the FPGA successfully loads the configuration image, the FPGA enters user mode. After the FPGA enters user mode, you can initiate a reconfiguration to a new page by setting the `fpga_pgm[2..0]` input pin and pulsing the `pfl_nreconfigure` input pin low.

Figure 15 shows the transition between different configurations.

Figure 15. Transitions Between Different Configurations in Remote System Upgrade


Configuration Error
Factory Configuration (1)
fpga_pgm[2..0] is set to factory page (2)
Set fpga_pgm[2..0] to intended page and pulse pfl_nreconfig
Configuration Error
Power-up
fpga_pgm[2..0] is set to application 1 page (2)
Application 1 Configuration
Set fpga_pgm[2..0] to intended page and pulse pfl_nreconfig
Set fpga_pgm[2..0] to intended page and pulse pfl_nreconfig
fpga_pgm[2..0] is set to application n page (2)
Application n Configuration
Configuration Error

Notes to Figure 15:

(1) The remote system upgrade feature in the PFL megafunction does not restrict the factory image to page 0, but allows the factory image to be located on other pages in the flash.

(2) You can load the FPGA with either a factory image or any application image upon power-up, depending on the `fpga_pgm[2..0]` setting.

The PFL megafunction can implement a Last Revision First programming order. The application image is updated with remote system upgrade capabilities. If a flash programming error causes the FPGA configuration to fail, the FPGA is reconfigured from the factory image address. A system shipped from the factory has the same configuration file at the application image address and the factory image address. Altera recommends that you write-protect the factory image blocks in the flash memory device.

Implementing Remote System Upgrade With the PFL Megafunction

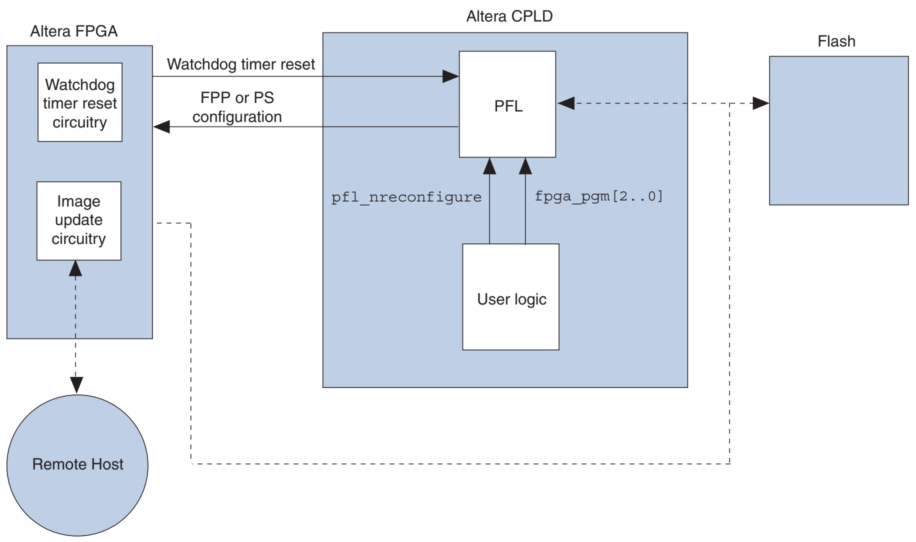
You can achieve the remote system upgrade capabilities with the PFL megafunction by controlling the `fpga_pgm[2..0]` and the `pfl_nreconfigure` ports. To control the `fpga_pgm[2..0]` and the `pfl_nreconfigure` ports, user-defined logic must perform the following capabilities:

- After FPGA power up, user logic sets the `fpga_pgm[2..0]` ports to specify which page of configuration image is to be loaded from the flash.
- After the remote host completes the new image update to the flash, user logic triggers a reconfiguration by pulling the `pfl_nreconfigure` pin low and setting the `fpga_pgm[2..0]` to the page in which the new image is located.

- If you have enabled the user watchdog timer, user logic can monitor the `pfl_watchdog_error` port to detect any occurrence of watchdog time-out error. If the `pfl_watchdog_error` pin is asserted high, this indicates watchdog time-out error. You can use the user logic to set the `fpga_pgm[2..0]` and pull the `pfl_nreconfigure` port low to initiate FPGA reconfiguration. The recovery page to be loaded from the flash memory device after watchdog timer error depends on the `fpga_pgm[2..0]` setting.

Figure 16 shows the implementation of remote system upgrade with the PFL megafunction.

Figure 16. Implementation of Remote System Upgrade With the PFL Megafunction



User Watchdog Timer

The user watchdog timer prevents faulty configuration from stalling the device indefinitely. The system uses the timer to detect functional errors after a configuration image is successfully loaded into the FPGA.

The user watchdog timer is a time counter that runs at the `pfl_clk` frequency. The timer begins counting after the FPGA enters user mode and continues until it reaches the user-defined watchdog time out period. You must periodically reset this timer by asserting the `pfl_reset_watchdog` pin before it reaches the watchdog time-out period. If the timer does not reset before it reaches the watchdog time-out period, the PFL megafunction detects watchdog time-out error and initiates a reconfiguration to load the factory image.

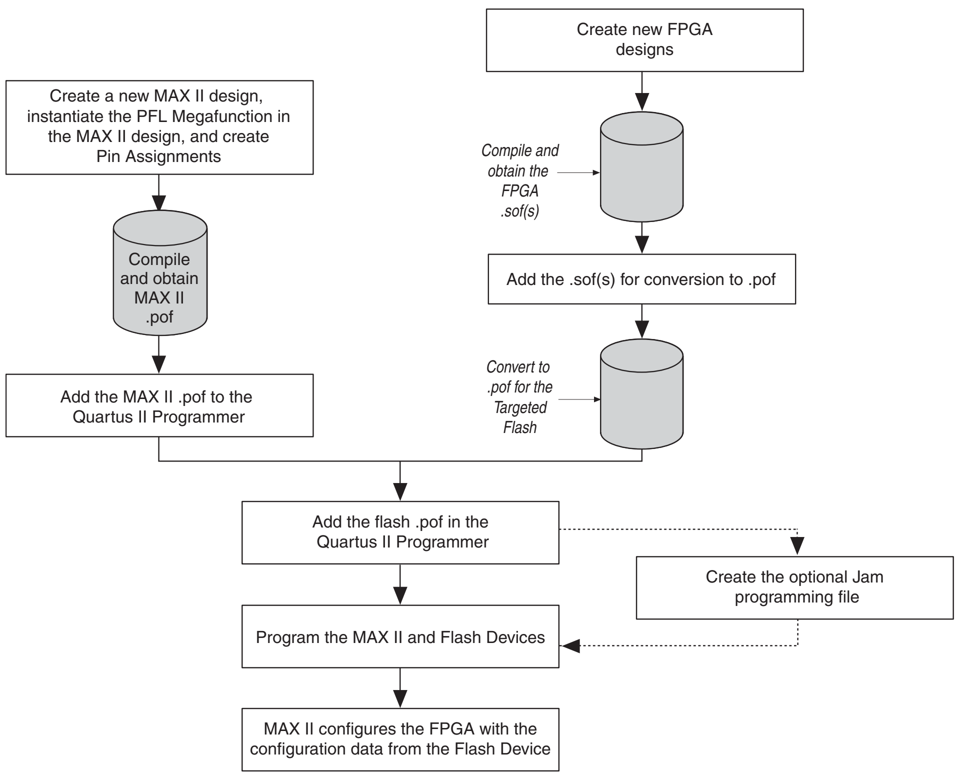
Instantiate the watchdog timer reset circuitry in the configuration image loaded into the FPGA, as shown in Figure 16 on page 20. Connect one output signal from the reset circuitry to the `pfl_reset_watchdog` pin of the PFL in the CPLD to periodically send a reset signal to the user watchdog timer. To reset the watchdog timer correctly, hold the `pfl_reset_watchdog` pin high or low for at least two `pfl_clk` cycles.

The user watchdog timer feature for remote system upgrade is available in the PFL megafunction in the Quartus II software version 10.0 and later.

Using the PFL Megafunction

This section describes how to use the PFL megafunction. Figure 17 shows the process for using the PFL megafunction, using MAX II as an example.

Figure 17. Process for Using the PFL Megafunction


Create new FPGA designs
Create a new MAX II design, instantiate the PFL Megafunction in the MAX II design, and create Pin Assignments
Compile and obtain the FPGA .sof(s)
Compile and obtain MAX II .pof
Add the .sof(s) for conversion to .pof
Add the MAX II .pof to the Quartus II Programmer
Convert to .pof for the Targeted Flash
Add the flash .pof in the Quartus II Programmer
Create the optional Jam programming file
Program the MAX II and Flash Devices
MAX II configures the FPGA with the configuration data from the Flash Device

The following sections describes these procedures:

- Instantiating the PFL Megafunction
- Constraining PFL Timing
- Simulating PFL Design
- Programming Altera CPLDs and Flash Memory Devices
- Defining New CFI Flash Device
- Programming Multiple Flash Memory Devices
- Creating Jam Files for Altera CPLDs and Flash Memory Device Programming

For more information about using the FPGA-based PFL megafunction to program a flash memory device, refer to *AN478: Using FPGA-Based Parallel Flash Loader with the Quartus II Software*.

Instantiating the PFL Megafunction

To instantiate the PFL megafunction, follow these steps:

1. In the Quartus II software, on the Tools menu, select **MegaWizard Plug-In Manager**.
2. Select **Create a new custom megafunction variation** and click **Next**.
3. For **Which device family will you be using**, select the **MAX II** device. If you are using a MAX V device, select the **MAX V** device.

   ☞ You can also instantiate the PFL megafunction in other FPGA device families.

4. Under **JTAG-accessible Extensions**, select **Parallel Flash Loader**.
5. Select the Hardware Description Language (HDL) output file type. For this example, select **Verilog HDL**.
6. Click **Next**.
7. Specify the directory and output filename.

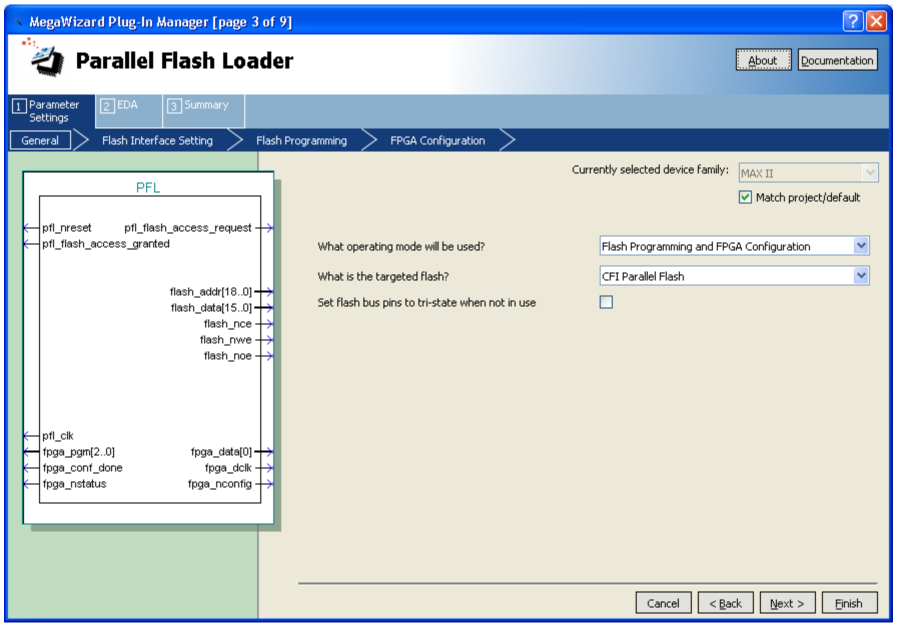
   ☞ Altera recommends instantiating the megafunction in your CPLD top-level design.

7. Click **Next** to display the **Parameter Settings** page.
7. Specify the parameter settings.

   ☞ For more information about the parameters and the allowed values, refer to Table 13 on page 46 .

Figure 18 shows the **Parameter Settings** page in the PFL megafunction parameter editor.

Figure 18. PFL Megafunction Settings (Page 3 of 9)



Currently selected device family: MAX II

☑ Match project/default

What operating mode will be used? Flash Programming and FPGA Configuration

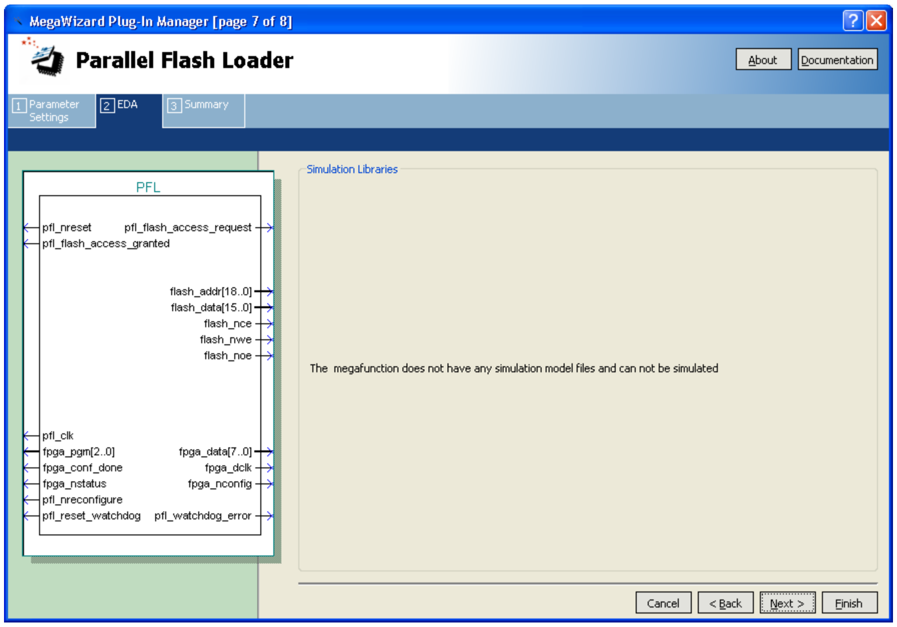
What is the targeted flash? CFI Parallel Flash

Set flash bus pins to tri-state when not in use ☐

8. Click **Next** or click the **EDA** tab to display the **EDA** page. Figure 19 shows that the PFL megafunction does not have simulation files and cannot be simulated.

The Quartus II software does not support simulation of the JTAG pins or the programming process of the Altera CPLD or flash memory device. However, simulation is possible for the FPGA configuration, on the condition that your FPGA is configured with the proper flash vectors and FPGA responses. Examples of flash vectors are `flash_addr` and `flash_data`; examples of FPGA responses are `fpga_conf_done` and `fpga_nstatus`.

Figure 19. List of Simulation Files Required (Page 7 of 8)



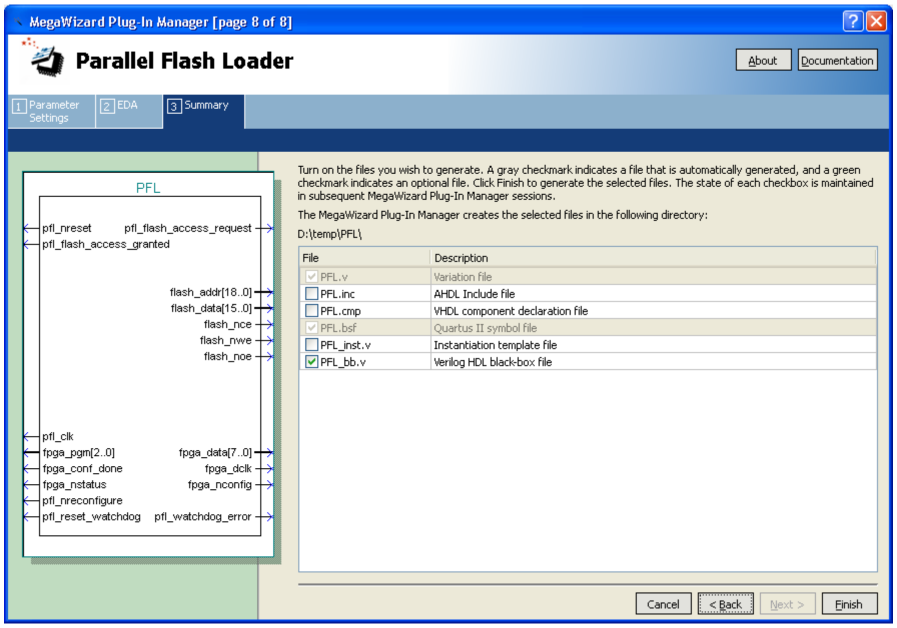
Simulation Libraries

The megafunction does not have any simulation model files and can not be simulated

For more information about the signals, refer to Table 14 on page 51.

9. Click **Next** or click the **Summary** tab to display the **Summary** page. Figure 20 shows the files that are created for the megafunction.

Figure 20. Selecting the Output File Type for the PFL Megafunction (Page 8 of 8)



Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

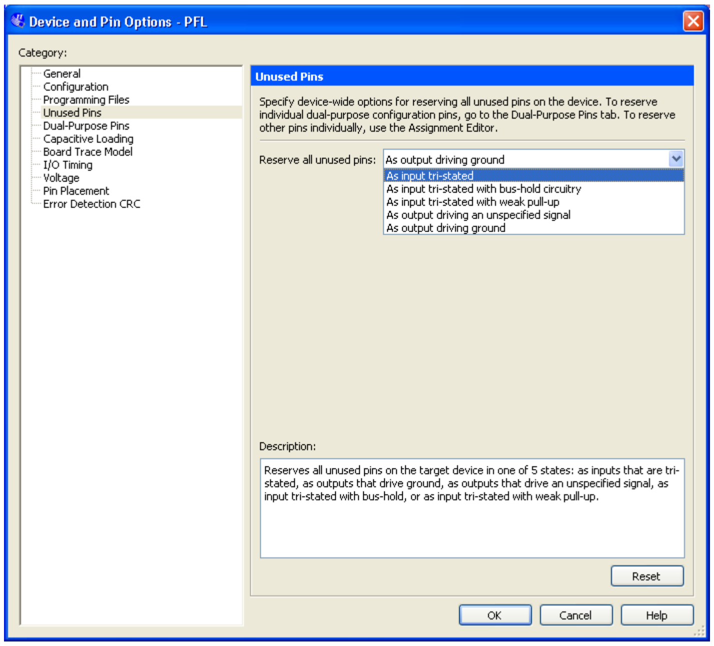
D:\temp\PFL\

File	Description
✓ PFL.v	Variation file
☐ PFL.inc	AHDL Include file
☐ PFL.cmp	VHDL component declaration file
✓ PFL.bsf	Quartus II symbol file
☐ PFL_inst.v	Instantiation template file
✓ PFL_bb.v	Verilog HDL black-box file

10. Select any additional file types that you want to create and click **Finish**. The Quartus II software generates the PFL megafunction files you select.

By default, all unused pins are tied to ground. Altera recommends setting all unused pins to tri-state because doing otherwise might cause interference. To set all unused pins to tri-state, in the Quartus II software, click **Assignments**>**Device**>**Device and Pin Options**>**Unused Pins** and select an item from the **Reserve all unused pins** list (Figure 21).

Figure 21. Reserve All Unused Pins



Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As output driving ground
- As input tri-stated
- As input tri-stated with bus-hold circuitry
- As input tri-stated with weak pull-up
- As output driving an unspecified signal
- As output driving ground

Description:

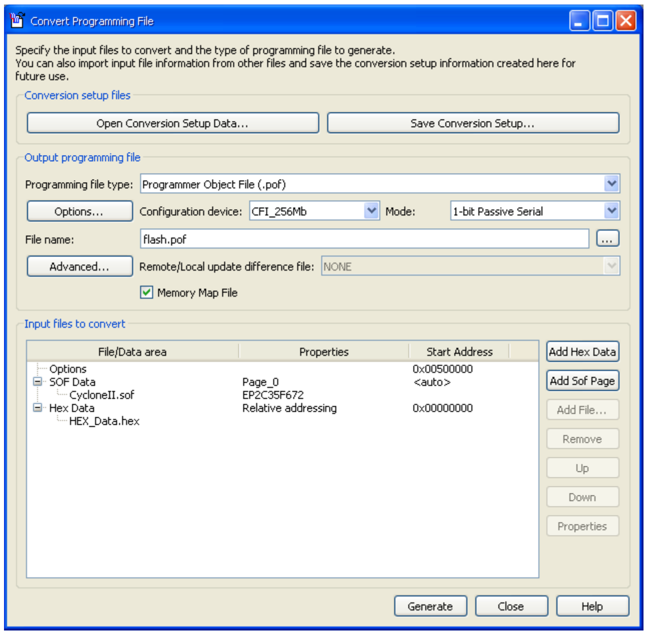
Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

Converting .sof Files to a .pof

To generate a programming file with different compression features, you must convert the **.sof** files to a **.pof**. To convert the **.sof** files to a **.pof**, follow these steps:

1. On the File menu, click **Convert Programming Files**.
2. For **Programming file type**, specify **Programmer Object File (.pof)** and name the file (refer to Figure 22).

Figure 22. Convert Programming File Tab



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

Programming file type: Programmer Object File (.pof)
Configuration device: CFI_256Mb Mode: 1-bit Passive Serial
File name: flash.pof
Remote/Local update difference file: NONE
☑ Memory Map File

File/Data area	Properties	Start Address
Options		0x00500000
SOF Data	Page_0	<auto>
CycloneII.sof	EP2C35F672	
Hex Data	Relative addressing	0x00000000
HEX_Data.hex		

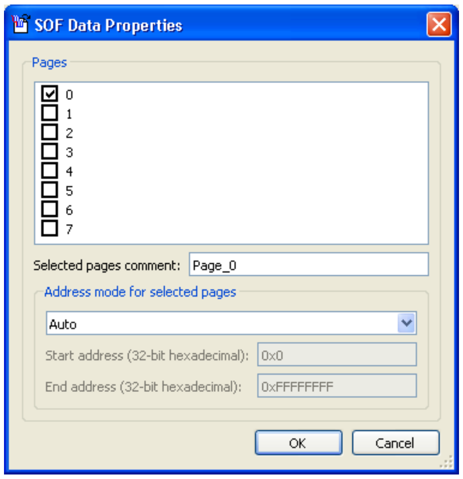
3. For **Configuration device**, select the CFI or NAND flash memory device with the correct density. For example, **CFI_32Mb** is a CFI device with 32-Megabit (Mb) capacity.
4. To add the configuration data, under **Input files to convert**, select **SOF Data**.
5. Click **Add File** and browse to the **.sof** files you want to add.

   You can place more than one **.sof** in the same page if you intend to configure a chain of FPGAs. The order of the **.sof** files must follow the order of the devices in the chain.

   If you want to store the data from other **.sof** files in a different page, click **Add SOF page**. Add the **.sof** files to the new page.

6. Select **SOF Data** and click **Properties** to set the page number and name. Figure 23 shows the **SOF Data Properties** dialog box.

Figure 23. SOF Data Properties



Selected pages comment: Page_0

Auto

Start address (32-bit hexadecimal): 0x0

End address (32-bit hexadecimal): 0xFFFFFFFF

7. Under **Address mode for selected pages**, select **Auto** to let the Quartus II software automatically set the start address for that page.
8. Select **Block** to specify the start and end addresses, or select **Start** to specify the start address only.
9. Click **OK**.
10. You can also store Hexadecimal (Intel-Format) File (**.hex**) user data in the flash memory device:

    a. In the **Input files to convert** sub-window of the Convert Programming Files window, select **Add Hex Data**.

    b. In the **Add Hex Data** dialog box, select either absolute or relative addressing mode.

    - If you select absolute addressing mode, the data in the **.hex** is programmed in the flash memory device at the exact same address location listed in the **.hex**.
    - If you select relative addressing mode, you are allowed to specify a start address.

    The data in the **.hex** is programmed into the flash memory device with the specific start address, and the differences between the addresses are kept. If no address is specified, the Quartus II software selects an address.

    You can also add other non-configuration data to the **.pof** by selecting the **.hex** that contains your data when creating the flash memory device **.pof**.

11. Click **Options** to specify the start address to store the option bits. This start address must be identical to the address you specify when creating the PFL megafunction. Ensure that the option bits sector does not overlap with the configuration data pages and that the start address resides on an 8-KB boundary.

12. If you are using a NAND flash memory device, specify the reserved block start address and the start address (including the option bits) within a 128-KB boundary. To specify the address, in the **File/Data area** column, select **NAND flash Reserved Block** and click **Properties**.
13. To generate programming files with either the typical or enhanced bitstream compression feature, or both, perform one of the following steps:
    - Typical bitstream compression feature
        - Select **.sof** under **SOF Data**.
        - Click **Properties**, and then turn on the **Compression** option.
        - Click **OK**.
    - Enhanced bitstream compression feature
        - In the **Options** dialog box, turn on the **Enable enhanced bitstream-compression when available** option.
        - Click **OK**.
    - Double compression technique

      Perform all the steps for the typical bitstream compression and enhanced bitstream compression features listed above.

    For more information about the compression feature in the PFL megafunction, refer to "Using Enhanced Bitstream Compression and Decompression" on page 16.
14. To generate programming files with encrypted data, select **.sof** under **SOF Data** and click **Properties**. Turn on the **Generate encrypted bitstream** check box.
15. Click **OK** to create the **.pof**.

Constraining PFL Timing

The PFL megafunction supports the Quartus II TimeQuest Timing Analyzer for accurate timing analysis on the Altera IP cores. To perform timing analysis, you must define the clock characteristics, external path delays, and timing exceptions for the PFL input and output ports. This section provides guidelines for defining this information for PFL input and output ports for use by the TimeQuest analyzer.

The TimeQuest analyzer is a timing analysis tool that validates the timing performance of the logic in the design using industry-standard constraint, analysis, and reporting methodology. For more information about the TimeQuest analyzer, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

After you specify the timing constraint settings for the clock signal and for the asynchronous and synchronous input and output ports in the TimeQuest analyzer, on the Constraints menu, click **Write SDC File** to write all the constraints to a specific System Design Constraints File (**.sdc**). After the **.sdc** is written, run full compilation for the PFL design.

Constraining Clock Signal

At any given time, one of the following two clock sources clocks the blocks and modules of the PFL megafunction:

- Clock signals from the `pfl_clk` ports of the PFL during FPGA configuration
- `TCK` pins of the JTAG programming interface during flash programming

The clock signal on the `TCK` pins is internally constrained to the maximum frequency supported by the selected JTAG programming hardware. You are not required to manually constrain the clock signal.

You can constrain `pfl_clk` up to the maximum frequency supported by the PFL megafunction. You can use the `create_clock` command or the **Create Clock** dialog box to specify the period and duty cycle of the clock constraint.

To constrain the `pfl_clk` signal in the TimeQuest analyzer, follow these steps:

1. Run full compilation for the PFL design. Ensure that the timing analysis tool is set to **TimeQuest Timing Analyzer**.
2. After full compilation completes, on the Tools menu, select **TimeQuest Timing Analyzer** to launch the TimeQuest analyzer window.
3. In the **Tasks** list, under **Diagnostic**, click **Report Unconstrained Paths** to view the list of unconstrained parts and ports of the PFL design.
4. In the **Report** list, under **Unconstrained Paths**, click **Clock Summary** to view the clock that requires constraints. The default setting for all unconstrained clocks is 1 GHz. To constrain the clock signal, right-click the clock name and select **Edit Clock Constraint**.
5. In the **Create Clock** dialog box, set the period and the duty cycle of the clock constraint.
6. Click **Run**.

Constraining Synchronous Input and Output Ports

The setup and hold time of synchronous input and output ports is critical to the system designer. To avoid setup and hold time violations, you can specify the signal delay from the FPGA or the flash memory device to the synchronous input and output ports of the PFL megafunction. The Quartus II Fitter places and routes the input and output registers of the PFL megafunction to meet the specified timing constraints.

For more information about the synchronous input and output ports of the PFL megafunction, refer to Table 7.

The signal delay from FPGA or flash memory device to the PFL synchronous input port is specified by `set_input_delay`. The calculation of the delay is:

Input delay value = Board delay from FPGA or flash output port to the PFL input port + T_{CO} of the FPGA or flash memory device

The signal delay from PFL synchronous output port to FPGA or flash memory device is specified by `set_output_delay`. The calculation of the delay is as follows:

Output delay value = Board delay from the PFL output port to the FPGA or flash input port + T_{CO} of the Altera CPLD

T_{CO} is the clock-to-output time which is obtained from the timing specification in the FPGA, CPLD or flash datasheet.

To constrain the synchronous input and output signals in the TimeQuest analyzer, follow these steps:

1. Follow steps 1 through 3 as specified in "Constraining Clock Signal" on page 29.
2. In the **Report** list, under the **Unconstrained Paths** category, select **Setup Analysis**, and then click on **Unconstrained Input Port Paths**.
3. Right-click each synchronous input or output port in the **From** list or **To** list and select **set_input_delay** for the input port or **set_output_delay** for the output port, then specify the input delay or output delay value.

Constraining Asynchronous Input and Output Ports

You can exclude Asynchronous input and output ports from the timing analysis of the PFL IP core because the signals on these ports are not synchronous to a megafunction clock source. The internal structure of the PFL megafunction handles the metastability of these asynchronous signals.

For more information about the asynchronous input and output ports of the PFL megafunction, refer to Table 7.

To exclude asynchronous input and output ports from the timing analysis, use the `set_false_path` command to ignore these ports during timing analysis.

After you specify all timing constraint settings for the clock signal, on the Constraints menu, click **Write SDC File** to write all the constraints to a specific **.sdc**. Then, run full compilation for the PFL design again.

Summary of PFL Timing Constraints

Table 7 lists the PFL timing constraints.

Table 7. PFL Timing Constraints (Part 1 of 2)

Type	Port	Constraint Type	Delay Value
Input clock	`pfl_clk`	`create_clock`	Can be constrained up to the maximum frequency supported by the PFL megafunction.
Input asynchronous	`pfl_nreset`	`set_false_path`	—
	`fpga_pgm`	`set_false_path`	—
	`fpga_conf_done`	`set_false_path`	—
	`fpga_nstatus`	`set_false_path`	—
	`pfl_flash_access_granted`	`set_false_path`	—

Table 7. PFL Timing Constraints (Part 2 of 2)

Type	Port	Constraint Type	Delay Value
Output asynchronous	fpga_nconfig	set_false_path	—
	pfl_flash_access_request	set_false_path	—
	flash_nce	set_false_path	—
	flash_nwe	set_false_path	—
	flash_noe	set_false_path	—
	flash_addr	set_false_path	—
Bidirectional synchronous	flash_data	■ Normal read mode: set_false_path ■ Burst read mode: set_input_delay	Burst read mode: Board delay from fpga_dclk pin of the CPLD to DCLK pin of the FPGA
Output synchronous	fpga_data	set_output_delay	Board delay + T_{CO} of the CPLD
	fpga_dclk	set_output_delay	Board delay from fpga_dclk pin of the CPLD to DCLK pin of the FPGA

Simulating PFL Design

You can simulate the behavior of the PFL megafunction with the ModelSim®-Altera software as it configures an FPGA. This section provides guidelines on the PFL simulation for FPGA configuration.

PFL simulation is based on functional netlist, and does not support gate-level simulation. PFL simulation does not reflect the true behavior of the hardware. Altera certifies the PFL megafunction based on actual hardware testing, and not through PFL simulation. The PFL simulation only provides primitive behavioral simulation.

For more information about simulation setup in ModelSim-Altera software, refer to the ModelSim-Altera Software Support page of the Altera website. For issues related to PFL simulation, check the known issues on the Knowledge Database page of the Altera website.

Table 8 lists the files required to simulate the PFL megafunction in the ModelSim-Altera software.

Table 8. Files Required for PFL Simulation in the ModelSim-Altera Software (Part 1 of 2)

File/Library	Description
.vo or **.vho**	The Verilog HDL or VHDL output file of the PFL megafunction.
.sdo	The Standard Delay Format Output file (**.sdo**) of the PFL megafunction.
Simulation libraries: ■ altera ■ altera_mf ■ maxii ■ maxv	The precompiled library files for Altera megafunction primitives and Altera CPLDs in the ModelSim-Altera software.

Table 8. Files Required for PFL Simulation in the ModelSim-Altera Software (Part 2 of 2)

File/Library	Description
Test bench	Test bench file to establish the interface between the PFL megafunction and the flash memory device.
Flash simulation model files	The simulation model files for the flash memory devices used in the PS or FPP configuration. For the flash simulation model file for each flash memory device, refer to the respective flash memory device manufacturer.

For more information about obtaining the **.vo** or **.vho**, **.sdo**, and simulation libraries in the ModelSim-Altera software, refer to About Using the ModelSim Software With the Quartus II Software in Quartus II Help.

Creating a Test Bench File for PFL Simulation

You can use a test bench file to establish the interface between the PFL megafunction and the flash memory device. You must map the input and output ports of the PFL megafunction to the appropriate data or address bus, and to the control signals of the flash. To perform the signal mapping, you must include the PFL primitive block and the flash primitive block in the test bench. The primitive blocks contain the input and output ports of the device. You can obtain the flash primitive blocks from the simulation model files provided by the flash memory device manufacturer.

For more information about the flash simulation model files, contact the flash memory device manufacturer.

Example 1 shows the primitive block for the PFL megafunction.

Example 1. PFL Primitive Block

```
pfl pfl_inst (
                  .fpga_pgm(<fpga_pgm source>),
                  .pfl_clk(<pfl clock source>),
                  .pfl_flash_access_granted(<pfl_flash_access_granted source>),
                  .pfl_flash_access_request(<pfl_flash_access_granted destination>),
                  .pfl_nreconfigure(<pfl_nreconfigure source>),
                  .pfl_nreset(<pfl_nreset source>),
                  .flash_addr(<flash address bus destination>),
                  .flash_data(<flash_data bus destination>),
                  .flash_nce(<flash_nce destination>),
                  .flash_noe(<flash_noe destination>),
                  .flash_nreset(<flash_nreset destination>),
                  .flash_nwe(<flash_nwe destination>),
                  .fpga_conf_done(<fpga_conf_done source>),
                  .fpga_nstatus(<fpga_nstatus source>),
                  .fpga_data(<fpga_data destination>),
                  .fpga_dclk(<fpga_dclk destination>),
                  .fpga_nconfig(<fpga_nconfig destination>),
          );
```

To establish the connection between the PFL megafunction and the flash memory device, you must connect the flash data bus, the flash address bus, and the flash control signals from the PFL primitive block to the appropriate ports of the flash primitive block.

Performing PFL Simulation in the ModelSim-Altera Software

To perform PFL simulation in the ModelSim-Altera software, you must specify the **.sdo** or load the ModelSim precompiled libraries listed in Table 8.

For more information about how to perform functional simulation with the ModelSim-Altera interface or command-line commands, refer to About Using the ModelSim-Altera Software With the Quartus II Software in Quartus II Help.

Performing PFL Simulation for FPGA Configuration

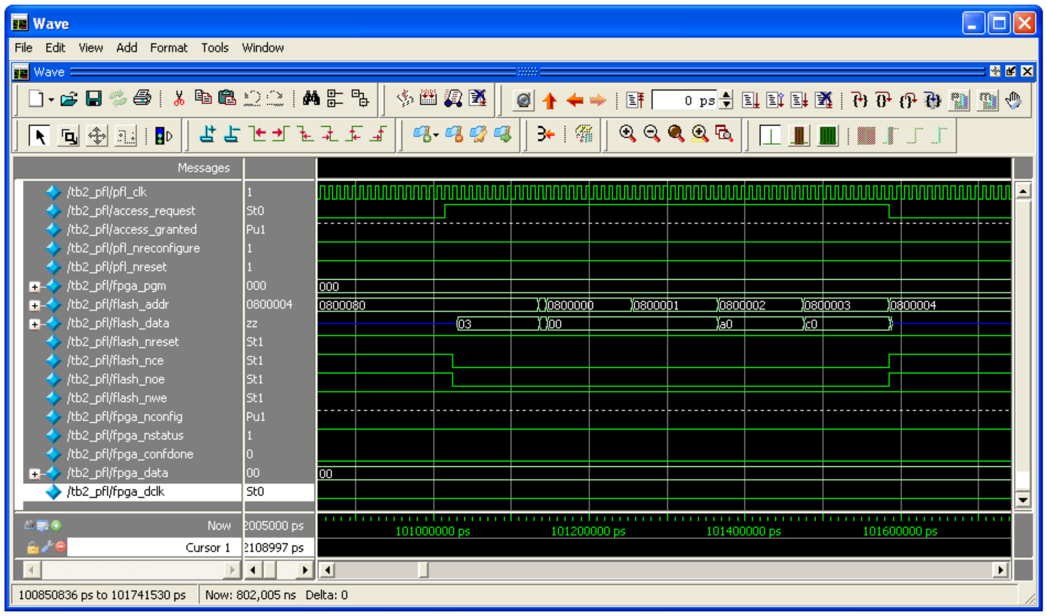
Before beginning the FPGA configuration, the PFL megafunction reads the option bits stored in the option bits sector to obtain information about the **.pof** version used for flash programming, the start and end address of each page of the configuration image stored in the flash, and the Page-Valid bit. In this simulation example, the start and end addresses of the option bits sector are 0×800000 and 0×800080, respectively. The PFL megafunction first reads from the final address, which is 0×800080, to obtain the **.pof** version information. Because `fpga_pgm[2..0]` is set to 000, the PFL megafunction reads from address 0×800000 to address 0×800003 to get the start and end address of page 0 and the Page-Valid bit. The LSB in address 0×800000 is the Page-Valid bit.

The Page-Valid bit must be 0 for the PFL megafunction to proceed with FPGA configuration. While the PFL megafunction reads from the flash, it asserts the active-low `flash_nce` and `flash_noe` signals, and asserts the active-high `pfl_flash_access_request` signal.

Figure 24 shows the simulation when the PFL megafunction reads the option bits from the flash memory device before configuration starts.

Before you perform the device configuration simulation, ensure that the PFL megafunction receives the correct option bits address and associated values to guarantee correct simulation output.

Figure 24. Simulation Before Configuration

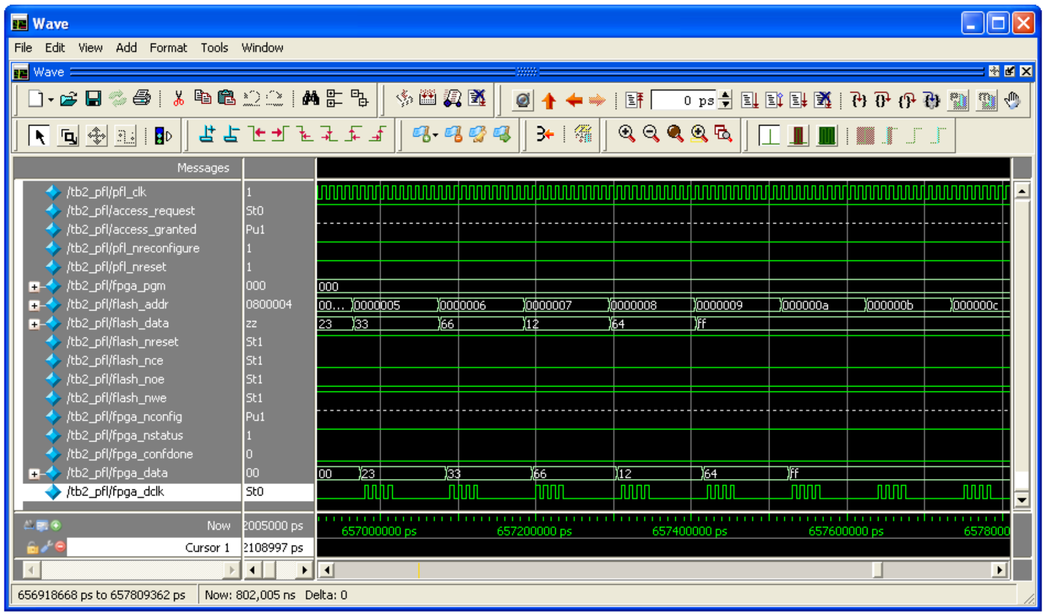


```
/tb2_pfl/pfl_clk             1
/tb2_pfl/access_request      St0
/tb2_pfl/access_granted      Pu1
/tb2_pfl/pfl_nreconfigure    1
/tb2_pfl/pfl_nreset          1
/tb2_pfl/fpga_pgm            000
/tb2_pfl/flash_addr          0800004
/tb2_pfl/flash_data          zz
/tb2_pfl/flash_nreset        St1
/tb2_pfl/flash_nce           St1
/tb2_pfl/flash_noe           St1
/tb2_pfl/flash_nwe           St1
/tb2_pfl/fpga_nconfig        Pu1
/tb2_pfl/fpga_nstatus        1
/tb2_pfl/fpga_confdone       0
/tb2_pfl/fpga_data           00
/tb2_pfl/fpga_dclk           St0
Now                          2005000 ps
Cursor 1                     2108997 ps
100850836 ps to 101741530 ps    Now: 802,005 ns   Delta: 0
```

After reading the option bits for page 0, the PFL megafunction waits for a period of time before the configuration starts. The `flash_data` remains at 0×ZZ within this period. Configuration starts when the `fpga_dclk` starts to toggle. During configuration, the PFL megafunction asserts the `flash_nce` and `flash_noe` signals low, and the `pfl_flash_access_request` signal high.

Figure 25 shows the simulation when the FPGA configuration starts.

Figure 25. Simulation When FPGA Configuration Starts



```
/tb2_pfl/pfl_clk              1
/tb2_pfl/access_request       St0
/tb2_pfl/access_granted       Pu1
/tb2_pfl/pfl_nreconfigure     1
/tb2_pfl/pfl_nreset           1
/tb2_pfl/fpga_pgm             000       000
/tb2_pfl/flash_addr           0800004   00... 0000005 0000006 0000007 0000008 0000009 000000a 000000b 000000c
/tb2_pfl/flash_data           zz        23 33 66 12 64 ff
/tb2_pfl/flash_nreset         St1
/tb2_pfl/flash_nce            St1
/tb2_pfl/flash_noe            St1
/tb2_pfl/flash_nwe            St1
/tb2_pfl/fpga_nconfig         Pu1
/tb2_pfl/fpga_nstatus         1
/tb2_pfl/fpga_confdone        0
/tb2_pfl/fpga_data            00        00 23 33 66 12 64 ff
/tb2_pfl/fpga_dclk            St0

Now        2005000 ps    657000000 ps   657200000 ps   657400000 ps   657600000 ps   6578000
Cursor 1   2108997 ps

656918668 ps to 657809362 ps    Now: 802,005 ns   Delta: 0
```

The FPGA configuration continues until the `fpga_conf_done` signal is asserted high, which indicates the configuration is complete. After the configuration process completes, the PFL megafunction pulls the `flash_nce` and `flash_noe` signals high and the `pfl_flash_access_request` signal low to indicate the configuration data is no longer being read from the flash memory device.

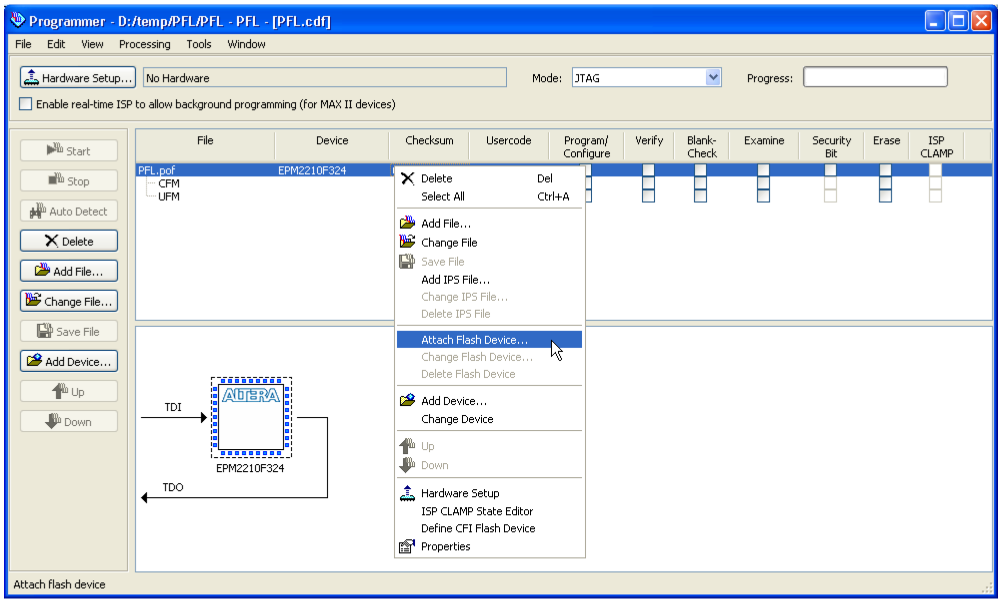
Programming Altera CPLDs and Flash Memory Devices

Using the Quartus II Programmer, you can program Altera CPLDs and flash memory device in a single step or separate steps.

To program both in a single step, first program the CPLD, then the flash memory device. Follow these steps:

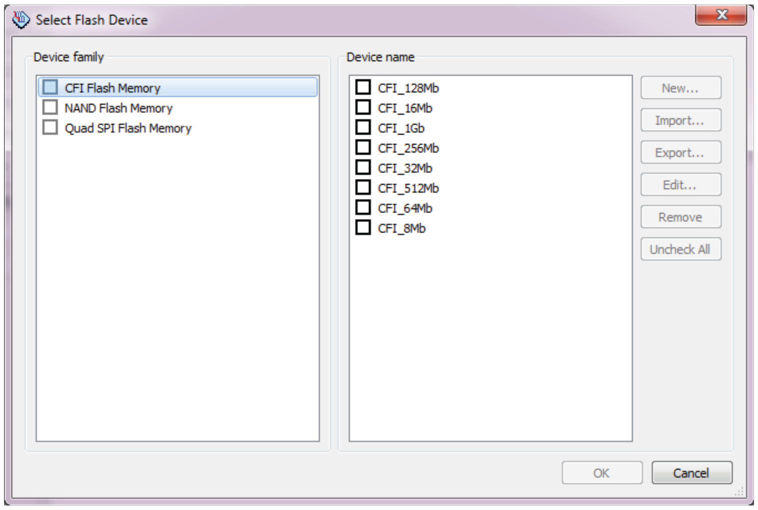
1. Open the Quartus II Programmer window and click **Add File** to add the **.pof** for the CPLD.
2. Right-click the CPLD **.pof** and click **Attach Flash Device**, as shown in Figure 26. The Flash Device menu appears.

Figure 26. Attaching Flash Device



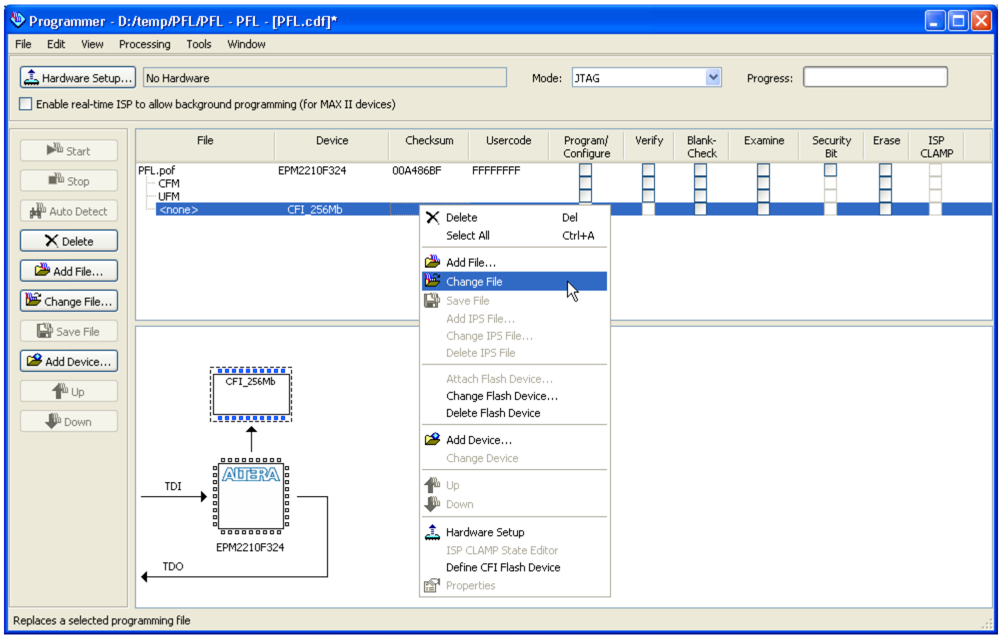
3. In the Flash Device menu, select the density of the flash memory device to be programmed, as shown in Figure 27.

Figure 27. Flash Device Selection



4. Right-click the desired flash memory device density and click **Change File**, as shown in Figure 28.

Figure 28. Attaching the Flash .pof



5. Select the **.pof** generated for the flash memory device. The **.pof** for the flash memory device is attached to the **.pof** of the CPLD.
6. Add other programming files if your chain has other devices.
7. Check all the boxes in the **Program/Configure** column for the new **.pof** and click **Start** to program the CPLD and flash memory device.

The Quartus II Programmer allows you to program, verify, erase, blank-check, or examine the configuration data page, the user data page, and the option bits sector separately, provided the CPLD contains the PFL megafunction.

The Quartus II Programmer erases the entire flash memory device if the flash memory device **.pof** is selected before programming. To prevent the Quartus II Programmer from erasing other sectors in the flash memory device, select only the pages, **.hex** data, and option bits.

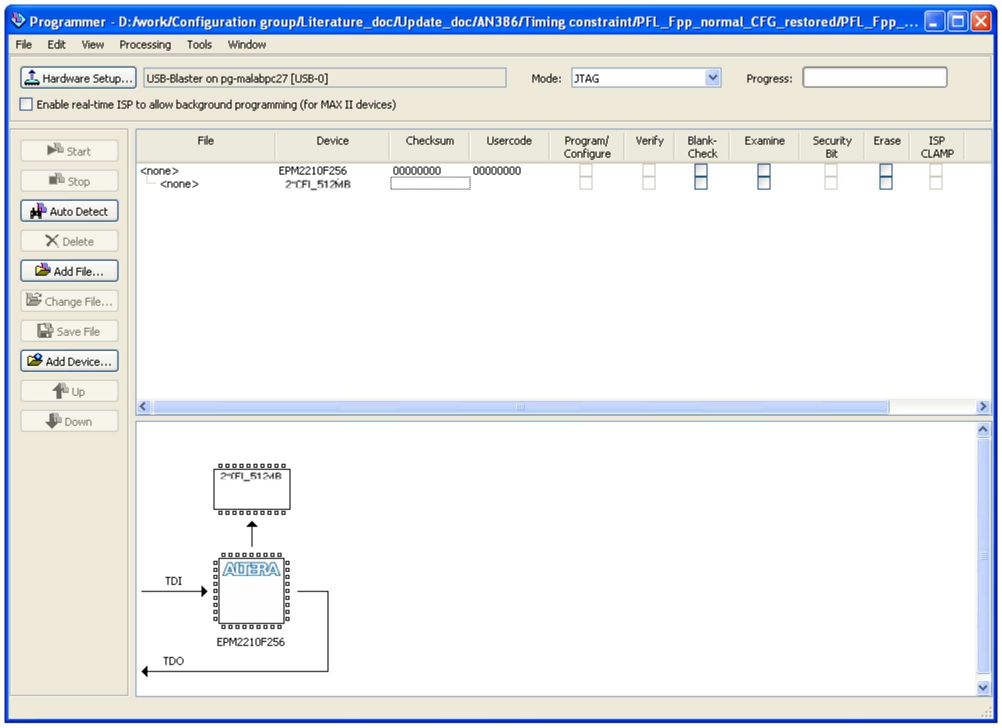
If you intend to use the flash memory device to store user data only, hold the `pfl_nreset` pin low at all times to prevent FPGA configuration.

To program the CPLD and the flash memory devices separately, follow these steps:

1. Open the Quartus II Programmer window.
2. Click **Add File**. The **Add Programming File Window** dialog box appears.
3. Add the targeted **.pof**, and click **OK**.
4. Check the boxes under the **Program/Configure** column of the **.pof**.
5. Click **Start** to program the CPLD.

6. After the programming progress bar reaches 100%, click **Auto Detect**. For example, if you are using dual P30 or P33, the programmer window shows a dual P30 or P33 chain in your setup (refer to Figure 29).

Figure 29. Quartus II Programmer



Programmer - D:/work/Configuration group/Literature_doc/Update_doc/AN386/Timing constraint/PFL_Fpp_normal_CFG_restored/PFL_Fpp_...

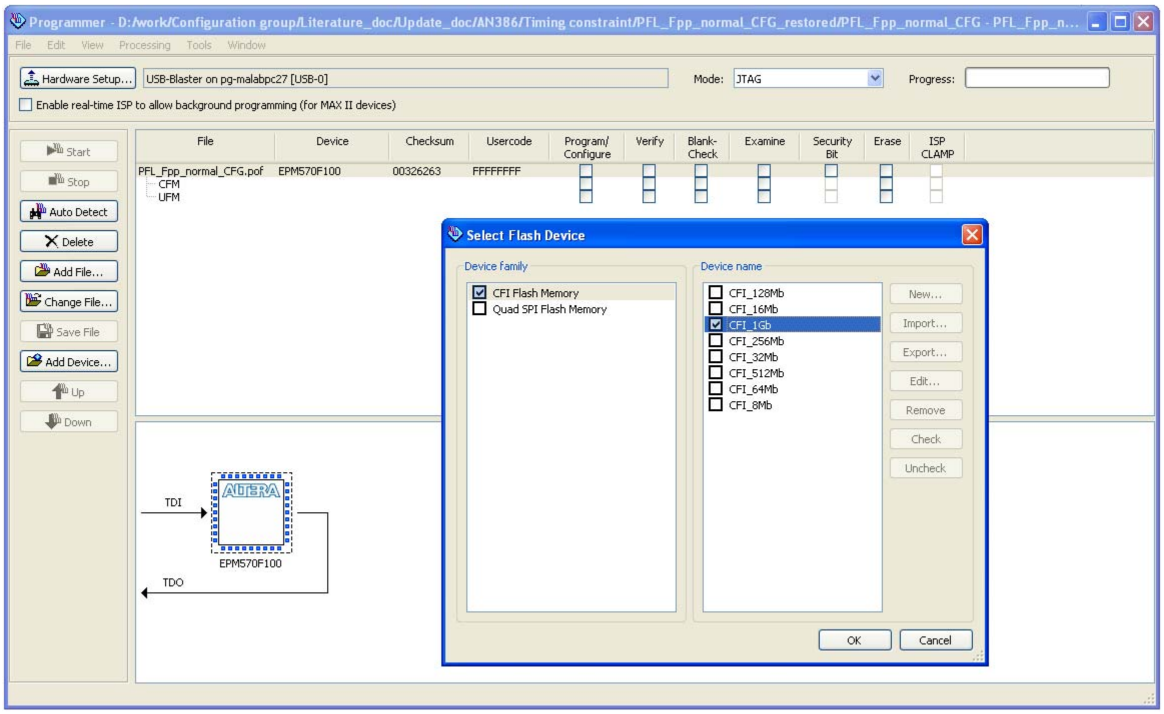
Hardware Setup... USB-Blaster on pg-malabpc27 [USB-0]   Mode: JTAG   Progress:

☐ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	Erase	ISP CLAMP
<none>	EPM2210F256	00000000	00000000	☐	☐	☐	☐	☐	☐	☐
<none>	2-CFI_512MB			☐	☐	☐	☐	☐	☐	☐

Alternatively, you can add the flash memory device to the programmer manually. Right-click the CPLD **.pof** and click **Select Flash Device**. In the **Select Flash Device** dialog box, select the device of your choice.

Figure 30. Attaching Flash Devices to the Quartus II Programmer



Programmer - D:/work/Configuration group/Literature_doc/Update_doc/AN386/Timing constraint/PFL_Fpp_normal_CFG_restored/PFL_Fpp_normal_CFG - PFL_Fpp_n...

Hardware Setup... USB-Blaster on pg-malabpc27 [USB-0] Mode: JTAG Progress:

Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	Erase	ISP CLAMP
PFL_Fpp_normal_CFG.pof	EPM570F100	00326263	FFFFFFFF							
CFM										
UFM										

Select Flash Device

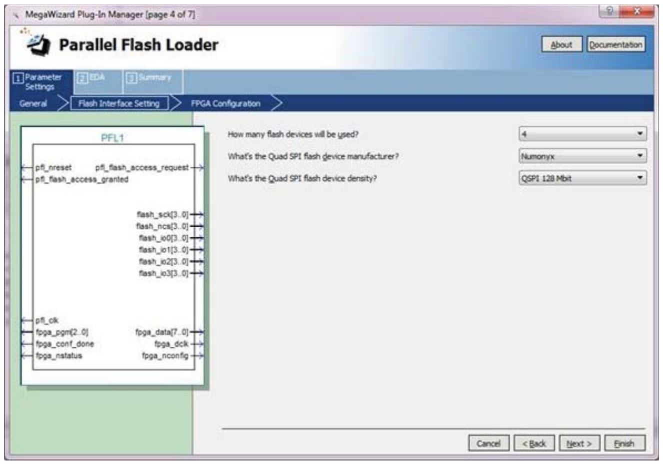
Device family
- [✓] CFI Flash Memory
- [ ] Quad SPI Flash Memory

Device name
- [ ] CFI_128Mb
- [ ] CFI_16Mb
- [✓] CFI_1Gb
- [ ] CFI_256Mb
- [ ] CFI_32Mb
- [ ] CFI_512Mb
- [ ] CFI_64Mb
- [ ] CFI_8Mb

7. Right-click the desired flash memory device density and click **Change File**.

You must select the density that is equivalent to the sum of the density of two CFI or NAND flash memory devices. For example, if you require two 512-Mb CFI flash memory devices, then select **CFI 1 Gbit**. For more than one quad SPI flash memory device, select the density that is equivalent to the sum of all the density of the quad SPI flash memory devices. For example, a four quad SPI flash memory devices (128 Mb for each device), the total density is equivalent to 512 Mb. A **.pof** with 512-Mb flash density is required to program these quad SPI flash devices. The PFL megafunction handles the 512-Mb **.pof** programming to the four quad SPI flash memory devices, as shown in Figure 31.

Figure 31. Selecting Density for CFI or NAND Flash Memory Devices



How many flash devices will be used? 4

What's the Quad SPI flash device manufacturer? Numonyx

What's the Quad SPI flash device density? QSPI 128 Mbit

8. Select the **.pof** generated for the flash memory device. The **.pof** for the flash memory device is attached to the **.pof** of the CPLD.
9. Check the boxes under the **Program/Configure** column for the added **.pof** and click **Start** to program the flash memory devices.

Defining New CFI Flash Device

The PFL megafunction supports Intel-compatible and AMD-compatible flash memory devices. In addition to the supported flash memory devices listed in Table 1 on page 2, you can define the new Intel- or AMD-compatible CFI flash memory device in the PFL-supported flash database using the **Define new CFI flash memory device** feature.

To add a new CFI flash memory device to the database or update a CFI flash device in the database, follow these steps:

1. In the Programmer window, on the Edit menu, select **Define New CFI Flash Device**. The Define CFI Flash Device window appears (refer to Figure 32). Table 9 lists the three functions available in the Define CFI Flash Device window.

Figure 32. Define CFI Flash Device

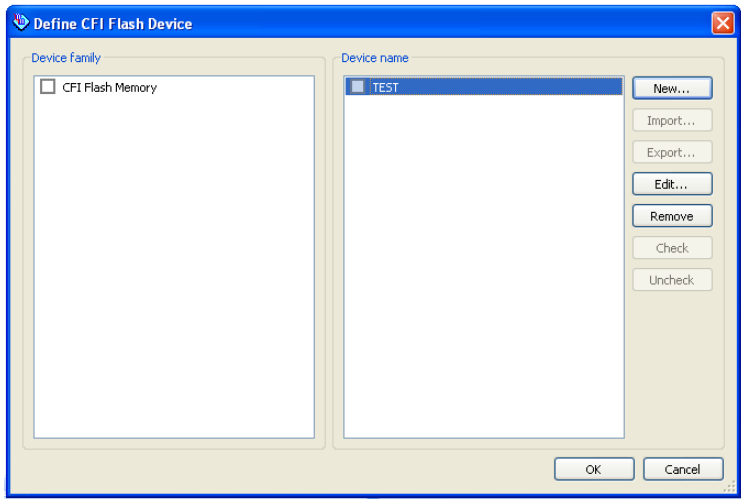
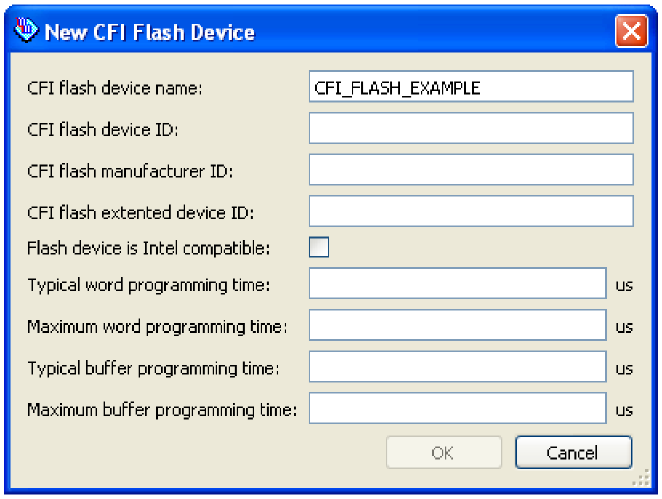


Table 9. Functions of the Define CFI Flash Device Feature

Function	Description
New	Add new Intel- or AMD-compatible CFI flash memory device into the PFL-supported flash database.
Edit	Edit the parameters of the newly added Intel- or AMD-compatible CFI flash memory device in the PFL-supported flash database.
Remove	Remove the newly added Intel- or AMD-compatible CFI flash memory device from the PFL-supported flash database.

2. To add a new CFI flash memory device or edit the parameters of the newly added CFI flash memory device, select **New** or **Edit**. The **New CFI Flash Device** dialog box appears (refer to Figure 33).
3. In the **New CFI Flash Device** dialog box, specify or update the parameters of the new flash memory device (refer to Table 10). You can obtain the values for these parameters from the datasheet of the flash memory device manufacturer.

Figure 33. New CFI Flash Device



CFI flash device name: CFI_FLASH_EXAMPLE
CFI flash device ID:
CFI flash manufacturer ID:
CFI flash extented device ID:
Flash device is Intel compatible:
Typical word programming time: us
Maximum word programming time: us
Typical buffer programming time: us
Maximum buffer programming time: us

Table 10. Parameter Settings for New CFI Flash Device

Parameter	Description
CFI flash device name	Define the CFI flash name
CFI flash device ID	Specify the CFI flash identifier code
CFI flash manufacturer ID	Specify the CFI flash manufacturer identification number
CFI flash extended device ID	Specify the CFI flash extended device identifier, only applicable for AMD-compatible CFI flash memory device
Flash device is Intel compatible	Check the checkbox if the CFI flash is Intel compatible
Typical word programming time	Typical word programming time value in µs unit
Maximum word programming time	Maximum word programming time value in µs unit
Typical buffer programming time	Typical buffer programming time value in µs unit
Maximum buffer programming time	Maximum buffer programming time value in µs unit

You must specify either the word programming time parameters, buffer programming time parameters, or both. Do not leave both programming time parameters at the default value of zero.

4. Click **OK** to save the parameter settings.
5. After you add, update, or remove the new CFI flash memory device, click **OK.**

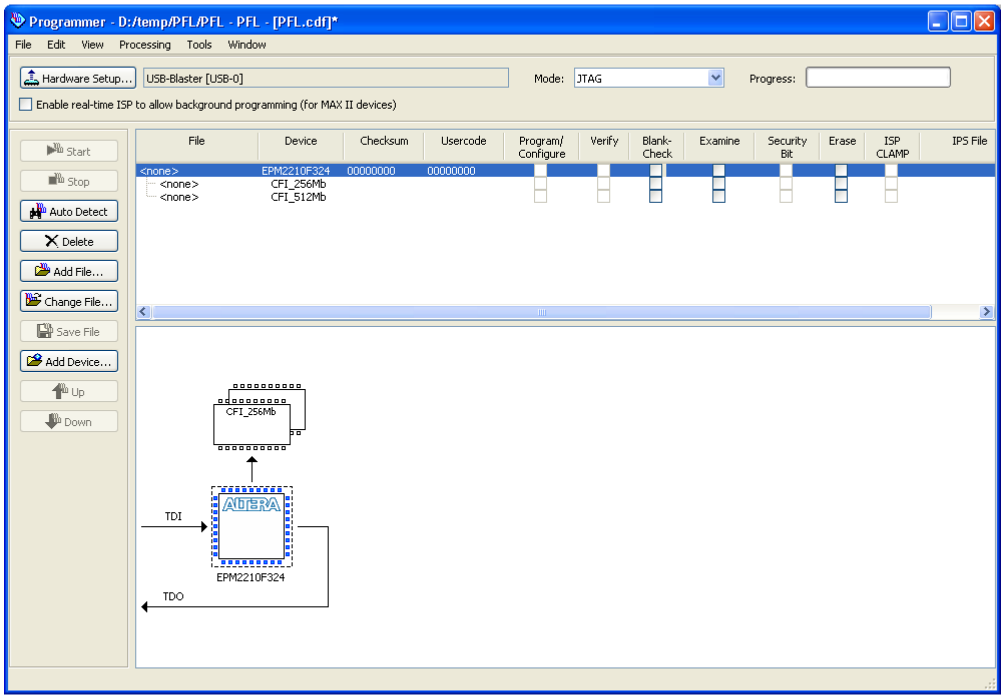
Programming Multiple Flash Memory Devices

The PFL megafunction supports multiple-flash programming of as many as 16 flash memory devices. This feature allows the PFL megafunction to connect to multiple flash memory devices to perform flash programming sequentially. PFL multiple-flash programming supports both speed and area mode flash programming. For FPGA configuration, the content in the flash memory device connected to the `nCE[0]` pin is used as configuration data.

To use the multiple flash programming feature, complete the following steps:

1. Select the number of flash memory devices connected to the CPLD in the PFL megafunction parameter editor.
2. Connect the `nCE` pins of the PFL to the `nCE` pins of the flash memory device in the block diagram. Compile the design.
3. Click **Auto Detect** in the Quartus II programmer. The CPLD appears as the main item, followed by a list of CFI flash memory devices detected as secondary items in the device tree (refer to Figure 34).

Figure 34. Auto Detect Flash Device



Hardware Setup... USB-Blaster [USB-0]   Mode: JTAG   Progress:

☐ Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP	IPS File
<none>	EPM2210F324	00000000	00000000								
<none>	CFI_256Mb										
<none>	CFI_512Mb										

4. Attach the flash memory device **.pof** to each flash memory device.
5. Check the boxes in the Quartus II Programmer for the necessary operation and click **Start**.

Creating Jam Files for Altera CPLDs and Flash Memory Device Programming

To use **.jam** files to program the CPLD and flash memory device, follow these steps:

1. Open the Quartus II Programmer and add the CPLD **.pof** and flash memory device **.pof**, by performing steps 1 through 5 in "Programming Altera CPLDs and Flash Memory Devices" on page 34.
2. On the File menu, point to **Create/Update** and click **Create JAM, JBC, SVF, or ISF File**.
3. Enter a name and select the file format (**.jam**).
4. Click **OK.**

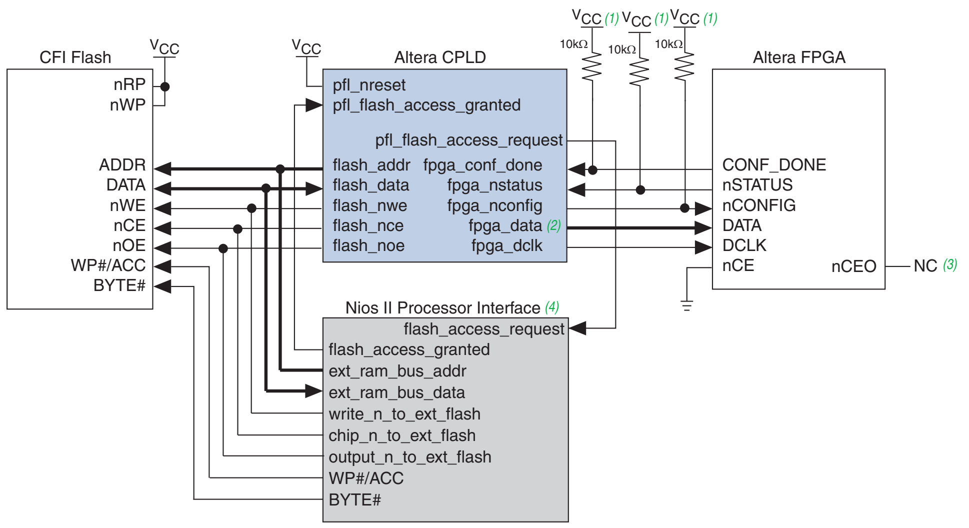
Use the **.jam** files with the Quartus II Programmer or `quartus_jli` executable file.

For more information about the `quartus_jli` executable, refer to *AN425: Using the Command-Line Jam STAPL Solution for Device Programming*.

PFL Megafunction In Embedded Systems

The PFL megafunction allows processors, such as the Nios® II processor, to access the flash memory device while programming flash and configuring an FPGA. Figure 35 shows how you can use the PFL megafunction to program the flash memory device and to configure the FPGA with a Nios II processor. The configured Nios II processor uses the non-configuration data stored in the same flash memory device.

Figure 35. Single-Device Configuration Using the PFL With the Controller



Notes to Figure 35:

(1) You must connect the pull-up resistor to a supply that provides an acceptable input signal for the devices. V_{CC} must be high enough to meet the V_{IH} specification of the I/O on both devices. For example, the Stratix II V_{IH} specification ranges from 1.7 to 3.3 V; therefore, the supply for the pull-up resistor, V_{CC}, must be within 1.7 to 3.3 V to meet the V_{IH} specification.

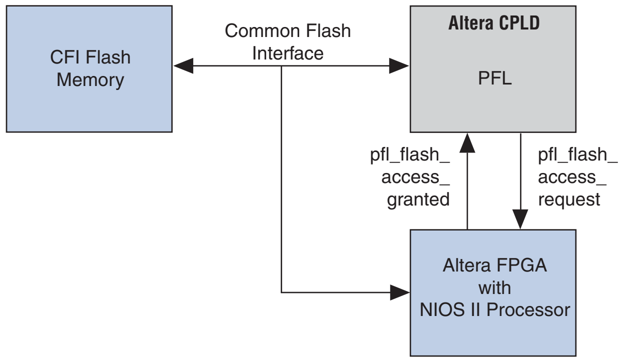
(2) For PS configuration mode, this is a 1-bit data line. For FPP configuration mode, this is an 8-bit data bus.

(3) Do not connect anything to the NC pin (the no connect pin), not even V_{CC} or GND.

(4) The Nios II processor can be implemented in any other Altera FPGA except the FPGA that is being configured.

Figure 36 shows the relationship between PFL megafunction, the CFI flash memory device, and the Nios II processor.

Figure 36. Relationship Between the Four Sections in the Design Example


CFI Flash Memory
Common Flash Interface
Altera CPLD
PFL
pfl_flash_ access_ granted
pfl_flash_ access_ request
Altera FPGA with NIOS II Processor

You must configure the Altera FPGA with the Nios II processor when you power up the board. You can store the Nios II processor image in the flash memory device and use the PFL megafunction to configure the image to the Altera FPGA. If you store the Nios II processor image in the same flash memory device you intend to program, make sure not to overwrite the Nios II processor image when you program the flash memory device with other user data.

If you do not want to store the image in the flash memory device, you can store the Nios II image in a different storage device, such as an enhanced configuration (EPC) device or an erasable programmable configurable serial (EPCS) memory.

In Figure 36, the Nios II processor and the PFL megafunction share the same bus line to the flash memory device. However, to avoid data contention, the processor and the megafunction cannot access or program the flash memory device at the same time. To ensure that only one controller (the processor or the megafunction), is accessing the flash memory device at any given time, you must tri-state all output pins that are connected to the flash memory device from one controller, while the other controller is accessing the flash memory device using the `pfl_flash_access_request` and `pfl_flash_access_granted` pins in the PFL megafunction.

Table 11 lists the functions of the `pfl_flash_access_request` and `pfl_flash_access_granted` pins.

Table 11. PFL Flash Access Pins

Pin	Description
`pfl_flash_access_request`	The PFL megafunction drives this pin high to request access to the flash memory device.
`pfl_flash_access_granted`	The PFL megafunction enables the access to the flash memory device whenever the PFL megafunction receives a high input signal at this pin.

Table 12 lists the methods to use the `pfl_flash_access_request` and `pfl_flash_access_granted` pins to ensure both processors are not accessing the flash memory device at the same time.

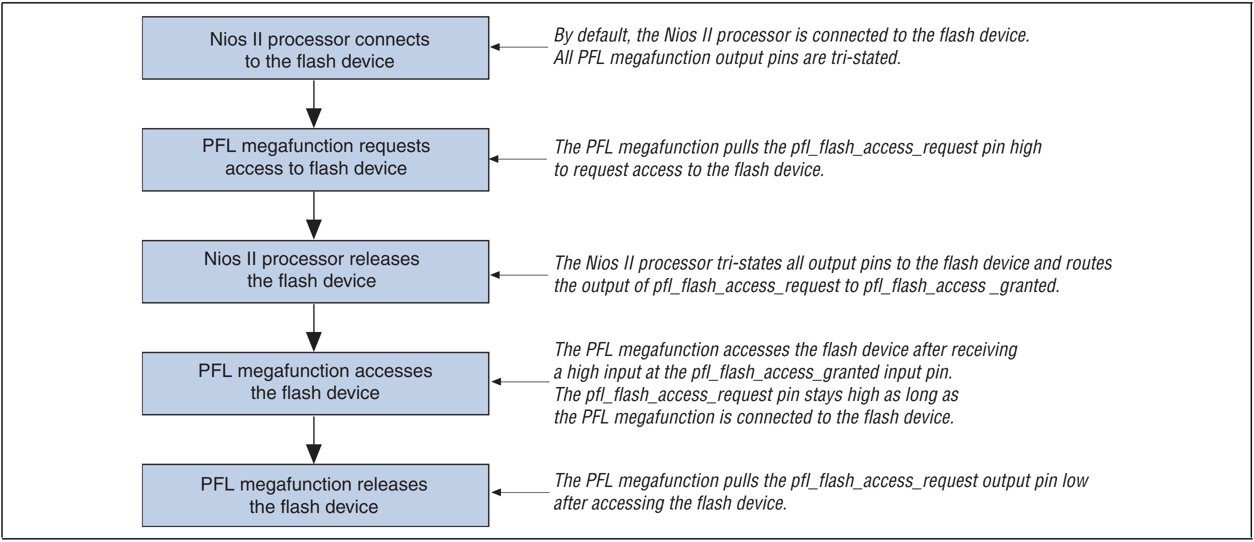
Table 12. pfl_flash_access_request and pfl_flash_access_granted Pins With the Nios II and PFL Megafunction

Signal	Nios II Processor	PFL Megafunction
High output signal at `pfl_flash_access_request`	Tri-state all output pins to the flash memory device.	Connect all input and output pins to the flash memory device when the `pfl_flash_access_granted` pin receives a high input.
Low output signal at `pfl_flash_access_request`	Reconnect all pins to the flash memory device.	Tri-state all output pins to the flash memory device when the `pfl_flash_access_granted` pin receives a low input.

The **Set bus pins to tri-state when not in use** option for the PFL megafunction disables the PFL megafunction whenever the `pfl_flash_access_granted` pin is pulled low.

Figure 37 shows the sequence of accesses to the flash memory device.

Figure 37. Nios II Processor and PFL Megafunction Accessing the Flash Memory Device


Nios II processor connects to the flash device
By default, the Nios II processor is connected to the flash device. All PFL megafunction output pins are tri-stated.
PFL megafunction requests access to flash device
The PFL megafunction pulls the pfl_flash_access_request pin high to request access to the flash device.
Nios II processor releases the flash device
The Nios II processor tri-states all output pins to the flash device and routes the output of pfl_flash_access_request to pfl_flash_access _granted.
PFL megafunction accesses the flash device
The PFL megafunction accesses the flash device after receiving a high input at the pfl_flash_access_granted input pin. The pfl_flash_access_request pin stays high as long as the PFL megafunction is connected to the flash device.
PFL megafunction releases the flash device
The PFL megafunction pulls the pfl_flash_access_request output pin low after accessing the flash device.

Altera recommends that you enable the safe state machine setting to prevent the PFL megafunction from entering an undefined state. To set this option, on the Assignments menu, click **Settings**. In the **Settings** dialog box, on the **Analysis and Synthesis** page, click **More Settings,** and select **safe state machine**.

The Altera CPLD and Nios II processor can each program the CFI flash memory device individually. To prevent both processors from accessing the CFI flash memory device at the same time, the `flash_access_granted` and `flash_access_request` pins of the CPLD and Nios II processor are connected together.

For more information about FPGA configuration, refer to the *Configuration Handbook*, and for more information about the Nios II processor, refer to the *Nios II Processor Reference Handbook*.

To use other processors or controllers in place of the Nios II processor, ensure that the `pfl_flash_access_granted` and `pfl_flash_access_request` pins of the PFL megafunction connect to your processor using the method described in Table 12 on page 45.

You must also specify the flash memory device read or write access time for your processor or controller. To avoid data contention when the PFL megafunction is accessing the flash memory device, ensure that the output pins from your processor are tri-stated when the `pfl_flash_access_request` signal is high.

Parameters

This section contains information about the GUI parameters of the PFL megafunction.

Table 13 lists the options available in the PFL megafunction parameter editor.

Table 13. PFL Megafunction Parameters (Part 1 of 5)

Megafunction Options	Value	Description
General		
Operating mode	Flash Programming and FPGA Configuration, Flash Programming, or FPGA Configuration	Specifies the operating mode of flash programming and FPGA configuration control in one megafunction or separate these functions into individual blocks and functionality.
Targeted flash device	CFI Parallel Flash, Quad SPI Flash, or NAND Flash	Specifies the flash memory device connected to the PFL megafunction.
Tri-state flash bus	On or Off	Turn on to allow the PFL megafunction to tri-state all pins interfacing with the flash memory device when the PFL megafunction does not require to access the flash memory.
Flash Interface Setting		
Number of flash devices used	CFI Parallel Flash: 1–16 Quad SPI Flash: 1,2,4 NAND Flash: 1	Specifies the number of flash memory devices connected to the PFL megafunction. The maximum number of flash memory devices allowed is four.
Largest flash density	CFI Parallel Flash: 8 Mbit–1 Gbit NAND Flash: 512 Mbit–1 Gbit	Specifies the density of the flash memory device to be programmed or used for FPGA configuration. If you have more than one flash memory device connected to the PFL megafunction, specify the largest flash memory device density. For CFI flash, select the density that is equivalent to the sum of the density of two CFI flashes. For example, if you use two 512-Mb CFI flashes, you must select **CFI 1 Gbit**. (Available only if you select **CFI Parallel Flash** or **NAND Flash**.)

Table 13. PFL Megafunction Parameters (Part 2 of 5)

Megafunction Options	Value	Description
Flash interface data width	CFI Parallel Flash: 8, 16, or 32 bits NAND Flash: 8 bits	Specifies the flash data width in bits. The flash data width depends on the flash memory device you use. For multiple flash memory device support, the data width must be the same for all connected flash memory devices. For CFI flash, select the flash data width that is equivalent to the sum of the data width of two CFI flashes. For example, if you are targeting dual P30 or P33 solution, you must select **32 bits** because each CFI flash data width is 16 bits. (Available only if you select **CFI Parallel Flash** or **NAND Flash**.)
User control `flash_nreset` pin	On or Off	Turn on to create a `flash_nreset` pin in the PFL megafunction to connect to the `reset` pin of the flash memory device. A low signal resets the flash memory device. In burst mode, this pin is available by default. (Available only if you select **CFI Parallel Flash**.)
Quad SPI flash device manufacturer	Macronix, Numonyx, Spansion	Specifies the device manufacturer of the quad SPI flash. (Available only if you select **Quad SPI Flash**.)
Quad SPI flash device density	8 Mbit–256 Mbit	Specifies the density of the quad SPI flash to be programmed or used for FPGA configuration. (Available only if you select **Quad SPI Flash**.)
Byte address for reserved block area	—	Specifies the start address of the reserved block area for bad block management. NAND flash memory may contain bad blocks that contain one or more invalid bits. The reserve blocks replace any bad blocks that the PFL megafunction encounters. Altera recommends that you reserve a minimum of 2% of the total block. (Available only if you select **NAND Flash**.)
On-die ECC support	On or Off	Turn on to enable the support for on-die ECC. Certain NAND flash memory devices has on-die ECC. Turning on this option allows the PFL megafunction to use the flash memory device's on-die ECC. Turning off this option allows the PFL megafunction to generate its own ECC engine. (Available only if you select **NAND Flash**.)
Flash Programming		
Flash programming IP optimization	Area, Speed	Specifies the flash programming IP optimization. If you optimize the PFL IP core for speed, the flash programming time is shorter but the megafunction uses more LEs. If you optimize the PFL IP core for area, the megafunction uses less LEs, but the flash programming time is longer. (Available only if you select **CFI Parallel Flash**.)

Table 13. PFL Megafunction Parameters (Part 3 of 5)

Megafunction Options	Value	Description
FIFO size	—	Specifies the FIFO size if you select **Speed** for flash programming IP optimization. The PFL megafunction uses additional LEs to implement FIFO as temporary storage for programming data during flash programming. With a larger FIFO size, programming time is shorter. (Available only if you select **CFI Parallel Flash**.)
Add Block-CRC verification acceleration support	On or Off	Turn on to add a block to accelerate verification. (Available only if you select **CFI Parallel Flash**.)
FPGA Configuration		
External clock frequency	—	Specifies the user-supplied clock frequency used by the megafunction to configure the FPGA. The clock frequency must not exceed two times the maximum clock (`DCLK`) frequency acceptable by the FPGA for configuration. The PFL megafunction can divide the frequency of the input clock maximum by two.
Flash access time	—	Specifies the access time of the flash. You can get the maximum access time required by a flash memory device from the flash datasheet. Altera recommends specifying a flash access time that is the same as or longer than the required time. For CFI parallel flash, the unit is in ns and for NAND flash, the unit is in us. NAND flash uses page instead of byte, and require more access time. This option is disabled for quad SPI flash.
Option bits byte address	—	Specifies the start address in which the option bits are stored in the flash memory. The start address must reside on an 8-KB boundary. For more information about option bits, refer to "Storing Option Bits" on page 13.
FPGA configuration scheme	PS (passive serial), FPP (fast passive parallel)	Select the FPGA configuration scheme. The default FPP is FPP x8. If you are using Stratix V devices, two additional FPP mode is available: FPP ×16 and FPP ×32.
Configuration failure response options	Halt, Retry same page, or Retry from fixed address	Configuration behavior after configuration failure. If you select **Halt**, the FPGA configuration stops completely after failure. If you select **Retry same page**, after failure, the PFL megafunction reconfigures the FPGA with data from the same page in which the failure occurred. If you select **Retry from fixed address**, the PFL megafunction reconfigures the FPGA with data from a fixed address specified in the next option field after failure.

Table 13. PFL Megafunction Parameters (Part 4 of 5)

Megafunction Options	Value	Description
Byte address to retry from on configuration failure	—	If you select **Retry from fixed address** for configuration failure option, this option specifies the flash address for the PFL megafunction to read from the reconfiguration in the case of a configuration failure.
Include input to force reconfiguration	On or Off	Turn on to include an optional reconfiguration input pin (`pfl_nreconfigure`) to enable a reconfiguration of the FPGA.
Watchdog timer	On or Off	Turn on to enable a watchdog timer for remote system upgrade support. Turning on this option, enables the `pfl_reset_watchdog` input pin, `pfl_watchdog_error` output pin, and specifies the time period before the watchdog timer times out. This watchdog timer is a time counter which runs at the `pfl_clk` frequency.
Time period before the watchdog timer times out	—	Specifies the time out period of the watchdog timer. The default time out period is 100 ms.
Ratio between input clock and DCLK output clock	1, 2, 4, or 8	Specifies the ratio between the input clock and `DCLK`. ■ Ratio 8 means every eight external clocks to `pfl_clk` generates one `fpga_dclk`. ■ Ratio 4 means every four external clocks to `pfl_clk` generates one `fpga_dclk`. ■ Ratio 2 means every two external clocks to `pfl_clk` generates one `fpga_dclk`. ■ Ratio 1 means every one external clock to `pfl_clk` generates one `fpga_dclk`.

Table 13. PFL Megafunction Parameters (Part 5 of 5)

Megafunction Options	Value	Description
Use advance read mode	Normal Mode, Intel Burst Mode (P30 or P33), Spansion Page Mode (GL), or Numonyx Burst Mode (M58BW)	An option to improve the overall flash access time for the read process during the FPGA configuration. ■ Normal mode—Applicable for all flash memory ■ Intel Burst mode—Applicable for Numonyx P30 and P33 flash memory only. Reduce sequential read access time ■ Spansion page mode—Applicable for Spansion GL flash memory only ■ Numonyx burst mode—Applicable for Numonyx M58BW flash memory only For more information about the read-access modes of the flash memory device, refer to the respective flash memory data sheet.
Enhanced bitstream decompression	None, Area, or Speed	Select to enable or disable the enhanced bitstream decompression block. If you select **None**, the core disables the enhanced bitstream decompression block. If you select **Area**, the core optimizes the logic resources used by the enhanced bitstream decompression block in the PFL megafunction. If you select **Speed**, the core optimizes the speed of the data decompression. You can only optimize speed if you select **FPP** as the FPGA configuration scheme. For Stratix V devices, this option is not available if you choose FPP ×16 or FPP ×32 as your configuration scheme.

Signals

This section contains information about the input and output signals of the PFL megafunction.

Table 14 lists the functions of the PFL signals and specifies the external pull-up resistor required for the configuration pins.

For more pull-up information about configuring pins for specific Altera FPGA families, refer to the *Configuration Handbook*.

Table 14. PFL Signals (1) (Part 1 of 4)

Pin	Description	Weak Pull-Up	Function
`pfl_nreset`	Input	—	Asynchronous reset for the PFL megafunction. Pull high to enable FPGA configuration. Otherwise, to prevent FPGA configuration, pull low at all times when you do not use the PFL megafunction. This pin does not affect the flash programming functionality of the PFL megafunction.
`pfl_flash_access_granted`	Input	—	Used for system-level synchronization. This pin is driven by a processor or any arbitrator that controls access to the flash. This active-high pin is connected permanently high if you want the PFL megafunction to function as the flash master. Pulling the `pfl_flash_access_granted` pin low prevents the JTAG interface from accessing the flash and FPGA configuration.
`pfl_clk` (2)	Input	—	User input clock for the device. Frequency must match the frequency specified in the megafunction and must not be higher than the maximum `DCLK` frequency specified for the specific FPGA during configuration.
`fpga_pgm[]` (2)	Input	—	Determines the page used for the configuration.
`fpga_conf_done` (2)	Input	10-kΩ Pull-Up Resistor	Connects to the `CONF_DONE` pin of the FPGA. The FPGA releases the pin high if the configuration is successful. During FPGA configuration, this pin remains low.
`fpga_nstatus` (2)	Input	10-kΩ Pull-Up Resistor	Connects to the `nSTATUS` pin of the FPGA. This pin must be released high before the FPGA configuration and must stay high throughout FPGA configuration. If a configuration error occurs, the FPGA pulls this pin low and the PFL megafunction stops reading the data from the flash memory device.
`pfl_nreconfigure` (2)	Input	—	A low signal at this pin initiates FPGA reconfiguration. You may reconnect this pin to a switch for more flexibility to set this input pin high or low to control FPGA reconfiguration. When FPGA reconfiguration is initiated, the `fpga_nconfig` pin is pulled low to reset the FPGA device.
`pfl_flash_access_request`	Output	—	Used for system-level synchronization. If required, this pin is connected to a processor or arbitrator. The PFL megafunction drives this pin high when the JTAG interface accesses the flash or the PFL megafunction configures the FPGA. This output pin works in conjunction with the `flash_noe` and `flash_nwe` pins.

Table 14. PFL Signals (1) (Part 2 of 4)

Pin	Description	Weak Pull-Up	Function
`flash_addr[]` (5)	Output	—	Address inputs for memory addresses. The width of the address bus line depends on the density of the flash memory device and the width of the `flash_data` bus.
`flash_data[]` (5)	Input or Output (bidirectional pin)	—	Data bus to transmit or receive 8- or 16-bit data to or from the flash memory in parallel. (3)
`flash_nce[]`	Output	—	Connects to the `nCE` pin of the flash memory device. A low signal enables the flash memory device. Use this pin for multiple flash memory device support. The `flash_nce` pin is connected to each `nCE` pin of all the connected flash memory devices. The width of this port depends on the number of flash memory devices in the chain.
`flash_nwe`	Output	—	Connects to the `nWE` pin of the flash memory device. A low signal enables write operation to the flash memory device.
`flash_noe`	Output	—	Connects to the `nOE` pin of the flash memory device. A low signal enables the outputs of the flash memory device during a read operation.
`flash_clk` (4)	Output	—	Used for burst mode. Connects to the `CLK` input pin of the flash memory device. The active edges of `CLK` increment the flash memory device internal address counter. The `flash_clk` frequency is half of the `pfl_clk` frequency in burst mode for single CFI flash. In dual P30 or P33 CFI flash solution, the `flash_clk` frequency runs at a quarter of the `pfl_clk` frequency.
`flash_nadv` (4)	Output	—	Used for burst mode. Connects to the address valid input pin of the flash memory device. Use this signal for latching the start address.
`flash_nreset`	Output	—	Connects to the reset pin of the flash memory device. A low signal resets the flash memory device.
`fpga_data[]` (2)	Output	—	Data output from the flash to the FPGA device during configuration. For PS mode, this is a 1-bit bus `fpga_data[0]` data line. For FPP mode, this is an 8-bit `fpga_data[7..0]` data bus.
`fpga_dclk` (2)	Output	—	Connects to the `DCLK` pin of the FPGA. Clock input data to the FPGA device during configuration.
`fpga_nconfig` (2)	Open Drain Output	10-kΩ Pull-Up Resistor	Connects to the `nCONFIG` pin of the FPGA. A low pulse resets the FPGA and initiates configuration. (3)

Table 14. PFL Signals (1) (Part 3 of 4)

Pin	Description	Weak Pull-Up	Function
`flash_sck[]`	Output	—	Clock source used for flash data read operation. Connects to the `CLK` input pin of the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the `CLK` input of all the quad SPI flashes. The width of the port is equivalent to the number of quad SPI flash in the chain.
`flash_ncs[]`	Output	—	Connects to the `ncs` pin of the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the `ncs` pin of all the quad SPI flashes. The width of this port is equivalent to the number of quad SPI flashes in the chain.
`flash_io0[]`	Output	—	The first bit of the data bus to or from the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the first bit of the data bus of all the quad SPI flashes. The width of this port is equivalent to the number of quad SPI flashes in the chain.
`flash_io1[]`	Output	—	The second bit of the data bus to or from the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the second bit of the data bus of all the quad SPI flashes. The width of this port is equivalent to the number of quad SPI flashes in the chain.
`flash_io2[]`	Output	—	The third bit of the data bus to or from the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the third bit of the data bus of all the quad SPI flashes. The width of this port is equivalent to the number of quad SPI flashes in the chain.
`flash_io3[]`	Output	—	The fourth bit of the data bus to or from the quad SPI flash. If more than one quad SPI flash is used, connect this pin to the fourth bit of the data bus of all the quad SPI flashes. The width of this port is equivalent to the number of quad SPI flashes in the chain.
`pfl_reset_watchdog`	Input	—	A toggle signal to reset the watchdog timer before the watchdog timer times out. Hold the signal high or low for at least two clock cycles of the `pfl_clk` frequency to correctly reset the watchdog timer.

Table 14. PFL Signals (1) (Part 4 of 4)

Pin	Description	Weak Pull-Up	Function
`pfl_watchdog_error`	Output	—	A high signal indicates an error to the watchdog timer.

Notes to Table 14:

(1) For maximum FPGA configuration `DCLK` frequencies, refer to the *Configuration Handbook*.

(2) These pins are not available for the flash programming option in the PFL megafunction.

(3) Altera recommends not inserting logic between the PFL pins and the CPLD I/O pins, especially on the `flash_data` and `fpga_nconfig` pins.

(4) Use the `flash_clk` and `flash_nadv` pins for burst mode only. Do not connect these pins from the flash memory device to the CPLD device if you are not using burst mode.

(5) When the PFL is not accessing the flash memory device, the output of the `flash_addr` and `flash_data` pins depends on the setting of the unused pins if you did not select the PFL interface tri-state option.

Specifications

This section provides the equations required to estimate the time required to configure the FPGA with the PFL megafunction.

The equations in Table 15 assume the following definitions:

- C_{flash} is the number of clock cycles required to read from flash memory.
- C_{cfg} is the number of input clock cycles to clock out the data (producing between one and 16 `DCLK` cycles, depending on the choice of flash data bus width and FPP or PS mode). The processes of reading from the flash and clocking out the data for configuration are performed in parallel; therefore, only the larger number between C_{flash} and C_{cfg} is important.
- F_{clk} is the input clock frequency to the PFL megafunction.
- T_{access} is the flash access time.
- C_{access} is the number of clock cycles required before the data from the flash is ready.
- T_{page_access} is the page read time for Spansion flash memory devices and is only applicable for page mode access. T_{page_access} is set to 30 ns in the PFL megafunction.
- *N* is the number of bytes to be clocked out. This value is obtained from the Raw Binary File (**.rbf)** for the specific FPGA.

Table 15. Equations for the PFL Megafunction (Part 1 of 2)

Flash Access Mode	Configuration Data Option	Flash Data Width	FPP Mode		PS Mode	
			DCLK Ratio = 1	DCLK Ratio = 2	DCLK Ratio = 1	DCLK Ratio = 2
Normal Mode	Normal	8 bits	$C_{flash} = C_{access}$ $C_{cfg} = 2$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 3$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 16$ $C_{overhead} = 5*C_{access}$
		16 bits	$C_{flash} = C_{access}/2$ $C_{cfg} = 1.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 2.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 16$ $C_{overhead} = 3*C_{access}$
	Compressed and/or encrypted	8 bits	$C_{flash} = C_{access}$ $C_{cfg} = 5$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 16$ $C_{overhead} = 5*C_{access}$
		16 bits	$C_{flash} = C_{access}/2$ $C_{cfg} = 4.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ $C_{cfg} = 16$ $C_{overhead} = 3*C_{access}$
Burst Mode	Normal	4 bits	$C_{flash} = 4$ $C_{cfg} = 1$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 2$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 8$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 16$ $C_{overhead} = 48$
		8 bits	$C_{flash} = 2$ $C_{cfg} = 1$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 2$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 8$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 16$ $C_{overhead} = 22*C_{access} + 8$
		16 bits	$C_{flash} = 1$ $C_{cfg} = 1$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 2$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 8$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 16$ $C_{overhead} = 20*C_{access} + 8$
	Compressed and/or encrypted	4 bits	$C_{flash} = 4$ $C_{cfg} = 4$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 8$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 8$ $C_{overhead} = 48$	$C_{flash} = 4$ $C_{cfg} = 16$ $C_{overhead} = 48$
		8 bits	$C_{flash} = 2$ $C_{cfg} = 4$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 8$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 8$ $C_{overhead} = 22*C_{access} + 8$	$C_{flash} = 2$ $C_{cfg} = 16$ $C_{overhead} = 22*C_{access} + 8$
		16 bits	$C_{flash} = 1$ $C_{cfg} = 4$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 8$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 8$ $C_{overhead} = 20*C_{access} + 8$	$C_{flash} = 1$ $C_{cfg} = 16$ $C_{overhead} = 20*C_{access} + 8$

Table 15. Equations for the PFL Megafunction (Part 2 of 2)

Flash Access Mode	Configuration Data Option	Flash Data Width	FPP Mode		PS Mode	
			DCLK Ratio = 1	DCLK Ratio = 2	DCLK Ratio = 1	DCLK Ratio = 2
Page Mode Access (1)	Normal	8 bits	$C_{flash} = C_{access}$ $C_{cfg} = 2$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 3$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 16$ $C_{overhead} = 5*C_{access}$
	Normal	16 bits	$C_{flash} = C_{access} / 2$ $C_{cfg} = 1.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 2.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 16$ $C_{overhead} = 3*C_{access}$
	Compressed	8 bits	$C_{flash} = C_{access}$ $C_{cfg} = 5$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 8$ $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ $C_{cfg} = 16$ $C_{overhead} = 5*C_{access}$
	Compressed	16 bits	$C_{flash} = C_{access} / 2$ $C_{cfg} = 4.5$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 8$ $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access} / 2$ $C_{cfg} = 16$ $C_{overhead} = 3*C_{access}$

- For Normal Mode and Burst Mode:

  $C_{access} = T_{access}*F_{clk}+1$

  Total Clock Cycles (from `nRESET` asserted high to N bytes of data clocked out)

  $= C_{overhead} + \max(C_{flash}, C_{cfg})*N$

  Total Configuration Time = Total Clock Cycle/ PFL Input Clock

- For Page Mode:

  $C_{access} = [(T_{access}*F_{clk}+1) + (T_{page\_access}*F_{clk}*15)]/16$

  Total Clock Cycles (from `nRESET` asserted high to N bytes of data clocked out)

  $= C_{overhead} + \max(C_{flash}, C_{cfg})*N$

  Total Configuration Time = Total Clock Cycle/ PFL Input Clock

Note to Table 15:

(1) Spansion page mode support is only available in the Quartus II software versions 8.0 and later.

The following are the configuration time calculation examples for normal mode, page mode, and burst mode:

Any reference to the core clock speed of 100 MHz is used simply to illustrate configuration time calculation and not a recommendaiton of the actual clock.

- Normal mode configuration time calculation:

  .rbf size for EP2S15 = 577KB = 590,848 Bytes
  Configuration mode = FPP without data compression or encryption
  Flash access mode = Normal Mode
  Flash data bus width = 16 bits
  Flash access time = 100 ns
  PFL input Clock = 100 MHz
  DCLK ratio = 2

The following formulas are used in this calculation:

C_{access} = T_{access}*F_{clk}+1
C_{flash} for Normal Mode = C_{access} / 2
C_{cfg} = 2.5
$C_{overhead}$ = 3*C_{access}
Total Clock Cycles = $C_{overhead}$ + max (C_{flash}, C_{cfg})*N
Total Configuration Time = Total Clock Cycle/ PFL Input Clock

Substitute these values in the following formulas:

C_{access} = (100ns * 100MHz) + 1 = 11
C_{flash} = 11/2 = 5.5
C_{cfg} = 2.5
$C_{overhead}$ = 3*11 = 33
Total Clock Cycles = 33 + 5.5 * 590848 = 3249697
Total Configuration Time at 100 MHz = 9453571/ 100 × 10^6 = 32.5ms

- Page mode configuration time calculation:

.rbf size for EP2S15 = 577 KB = 590,848 Bytes
Configuration mode = FPP without data compression or encryption
Flash access mode = Page Mode
Flash data bus width = 16 bits
Flash access time = 100 ns
PFL input Clock = 100 MHz
DCLK ratio = 2

The following formulas are used in this calculation:

T_{page_access} = 30 ns
C_{access} = [(T_{access}*F_{clk}+1) + (T_{page_access}*F_{clk}*15)]/16
C_{flash} for Page Mode = C_{access} / 2
C_{cfg} = 2.5
$C_{overhead}$ = 3* C_{access}
Total Clock Cycles = $C_{overhead}$ + max (C_{flash}, C_{cfg})*N
Total Configuration Time = Total Clock Cycle/ PFL Input Clock

Substitute these values in the following formulas:

C_{access} = [((100ns * 100 MHz) + 1) + (30ns*100 MHz*15)]/16 = 3.5
C_{flash} for Page Mode = 3.5/ 2 = 1.75 = 2
C_{cfg} = 2.5
$C_{overhead}$ = 3*3.5 = 10.5
Total Clock Cycles = 10.5 + 2.5*590848 = 1477130.5
Total Configuration Time at 100 MHz = 1477130.5 / 100 × 10^6 = 14.77 ms

- Burst mode configuration time calculation:

  .rbf size for EP2S15 = 577KB = 590,848 Bytes
  Configuration mode = FPP without data compression or encryption
  Flash access mode = Burst Mode
  Flash data bus width = 16 bits
  Flash access time = 100 ns
  PFL input Clock = 100 MHz
  DCLK ratio = 2

  The following formulas are used in this calculation:

  $C_{access} = T_{access} * F_{clk} + 1$
  C_{flash} for Burst Mode = 1
  $C_{cfg} = 2$
  $C_{overhead} = 20 * C_{access} + 8$
  Total Clock Cycles = $C_{overhead}$ + max (C_{flash}, C_{cfg})*N
  Total Configuration Time = Total Clock Cycle/ PFL Input Clock

  Substitute these values in the following formulas:

  C_{access} = (100ns * 100 MHz) + 1 = 11
  $C_{flash} = 1$
  $C_{cfg} = 2$
  $C_{overhead} = (20*11)+8 = 228$
  Total Clock Cycles = 228 + 2 * 590848 = 1181924
  Total Configuration Time at 100 MHz = 1181924 / 100 × 10^6 = 11.82 ms

  The following are the configuration time calculation examples for single or cascaded quad SPI flash:

- Single quad SPI flash:

  .rbf size for EP2S15 = 577KB = 590,848 Bytes
  Configuration mode = FPP without data compression or encryption
  Flash access mode = Burst Mode
  Flash data bus width = 4 bits (only one quad SPI flash is used)
  Flash access time = 100 ns
  PFL input Clock = 100 MHz
  DCLK ratio = 2

  The following formulas are used in this calculation:

  $C_{flash} = 4$
  $C_{cfg} = 2$
  $C_{overhead} = 48$
  Total Clock Cycles = $C_{overhead}$ + max (C_{flash}, C_{cfg})*N
  Total Configuration Time = Total Clock Cycle/ PFL Input Clock

  Substitute these values in the following formulas:

  $C_{flash} = 4$
  $C_{cfg} = 2$
  $C_{overhead} = 48$
  Total Clock Cycles = 48 + 4 * 590848 = 2363440
  Total Configuration Time at 100 MHz = 2363440 / 100 × 10^6 = 23.63 ms

- Four cascaded quad SPI flashes:

  .rbf size for EP2S15 = 577KB = 590,848 Bytes
  Configuration mode = FPP without data compression or encryption
  Flash access mode = Burst Mode
  Flash data bus width = 16 bits (total bus width for four quad SPI flashes)
  Flash access time = 100 ns
  PFL input Clock = 100 MHz
  DCLK ratio = 2

  The configuration time calculation for four cascaded quad SPI flash is identical to the configuration time calculation for CFI flash with 16 bit flash data width.

Document Revision History

Table 16 lists the revision history for this document.

Table 16. Document Revision History

Date	Version	Changes
December 2011	2.0	■ Updated "Using Enhanced Bitstream Compression and Decompression" to include reference. ■ UpdatedTable 1 to include Eon Silicon CFI device EN29GL128 and to remove S29GL-N devices. ■ Updated Table 2 to include Numonyx QSPI device N25Q128. ■ Updated "Specifications" ■ Updated Table 13 to include FPP x16 and FPP x32 configuration scheme for Stratix V devices. ■ Updated Figure 27. ■ Added Figure 31. ■ Minor text edits.
February 2011	1.1	■ Restructured the user guide. ■ Added information about the new feature for the Quartus II software 10.1 release: Support for NAND flash.
July 2010	1.0	Converted from *AN386: Using the Parallel Flash Loader With the Quartus II Software*.