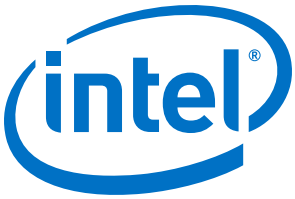


Intel® FPGA Parallel Flash Loader IP Core User Guide

Updated for Intel® Quartus® Prime Design Suite: **17.1**

UG-01082 | 2017.11.06
Latest document on the web: **PDF** | **HTML**

Subscribe
Send Feedback

Contents

1 Intel® FPGA Parallel Flash Loader IP Core User Guide

This document describes how to instantiate the Parallel Flash Loader (PFL) IP core in your design, programming flash memory, and configuring your FPGA from the flash memory.

FPGAs' increasing density requires larger configuration storage. 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 IP core in Intel MAX® devices (Intel MAX 10, MAX II, and MAX V devices) or all other FPGAs to program flash memory devices efficiently through the JTAG interface and to control configuration from the flash memory device to the Intel FPGA.

Related Links

- Parallel Flash Loader IP Core User Guide Archives on page 50
  Provides a list of user guides for previous versions of the Parallel Flash Loader IP core.
- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating Version-Independent IP and Qsys Simulation Scripts
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.
- JEDEC: Common Flash Interface (CFI)
  Provides more information about JEDEC CFI standard..

1.1 Features

Use the PFL IP core to:

- Program Common Flash Interface (CFI) flash, quad Serial Peripheral Interface (SPI) flash, or NAND flash memory devices with the device JTAG interface.
- Control Intel FPGA configuration from a CFI flash, quad SPI flash, or NAND flash memory device for Cyclone, Arria or Stratix series FPGA devices.

1.2 Device Support

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

Related Links

AN478: Using FPGA-Based Parallel Flash Loader with the Quartus Prime Software
Provides more information about using the FPGA-based PFL IP core to program a flash memory device.

1.2.1 Supported Flash Memory Devices

The Intel Quartus Prime software generates the PFL IP core logic for the flash programming bridge and FPGA configuration.

Table 1. CFI Flash Memory Devices Supported by PFL IP Core

If your CFI device is not in the following table, but is compatible with an Intel Quartus Prime or Cypress CFI flash device, Intel recommends selecting Define CFI Flash Device in the Intel Quartus Prime software.

Manufacturer	Product Family	Data Width	Density (Megabit)	Device Name[1]
Micron	C3[2]	16	8	28F800C3
			16	28F160C3
			32	28F320C3
			64	28F640C3
	J3[2]	8 or 16	32	28F320J3
			64	28F640J3
			128	28F128J3
		16	256	JS29F256J3
	P30[2]	16	64	28F640P30
			128	28F128P30
			256	28F256P30
			512	28F512P30
			1000	28F00AP30
			2000	28F00BP30
	P33[2]	16	64	28F640P33
			128	28F128P33
			256	28F256P33
			512	28F512P33
			1000	28F00AP33
			2000	28F00BP33
	M29EW[2]	8 or 16	256	28F256M29EW

continued...

(1) The PFL IP core supports top and bottom boot block of the flash memory devices. For Micron flash memory devices, the PFL IP core supports top, bottom, and symmetrical blocks of flash memory devices.

(2) Micron has discontinued this flash memory device family. Intel does not recommend you using this flash memory device.

Manufacturer	Product Family	Data Width	Density (Megabit)	Device Name[1]
			512	28F512M29EW
			1000	28F00AM29EW
	M29W[2]	8 or 16	16	M28W160CT
				M28W160CB
				M29W160F7
				M29W160FB
			32	M29W320E
				M29W320FT
				M29W320FB
			64	M29W640F
				M29W640G
			128	M29W128G
			256	M29W256G
	M29DW[2]	8 or 16	32	M29DW323DT
				M29DW323DB
	G18[2]	16	512	MT28GU512AAA1EGC-0SIT
			1024	MT28GU01GAAA1EGC-0SIT
	M58BW[2]	32	16	M58BW16FT
				M58BW16FB
			32	M58BW32FT
		16 or 32	32	M58BW32FB
	MT28EW[3]	8 or 16	128	MT28EW128ABA1
		8 or 16	256	MT28EW256ABA1
		8 or 16	512	MT28EW512ABA1
		8 or 16	1024	MT28EW01GABA1
Cypress[4]	GL-P[5]	8 or 16	128	S29GL128P
			256	S29GL256P
			512	S29GL512P
			1024	S29GL01GP

continued...

(1) The PFL IP core supports top and bottom boot block of the flash memory devices. For Micron flash memory devices, the PFL IP core supports top, bottom, and symmetrical blocks of flash memory devices.

(3) Device supports page mode.

(4) Cypress was formerly known as Spansion.

(5) Supports page mode.

Manufacturer	Product Family	Data Width	Density (Megabit)	Device Name[1]
	AL-D	8 or 16	16	S29AL016D
			32	S29AL032D
	AL-J	8 or 16	16	S29AL016J
	AL-M	8 or 16	16	S29AL016M
	JL-H	8 or 16	32	S29JL032H
			64	S29JL064H
	WS-N	16	128	S29WS128N
	GL-S	16	128	S29GL128S
			256	S29GL256S
			512	S29GL512S
			1024	S29GL01GS
Macronix	MX29LV	16	16	MX29LV160D
			32	MX29LV320D
			64	MX29LV640D
				MX29LV640E
	MX29GL	16	128	MX29GL128E
			256	MX29GL256E
ESMT[6]	EN29LV	16	16	EN29LV160B
	EN29GL	16	32	EN29LV320B
			128	EN29GL128

Table 2. Quad SPI Flash Memory Device Supported by PFL IP Core

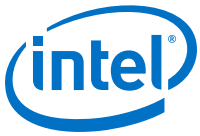
Manufacturer	Product Family	Density (Megabit)	Device Name
Micron	N25Q 1.8V[7]	128	N25Q128
	N25Q 3.3V[7]	128	N25Q128
		256	N25Q256
Cypress	FL	32	S25FL032P
		64	S25FL064P
		128	S25FL129P
Macronix	MX25L	8	MX25L8035E

continued...

(1) The PFL IP core supports top and bottom boot block of the flash memory devices. For Micron flash memory devices, the PFL IP core supports top, bottom, and symmetrical blocks of flash memory devices.

(6) ESMT was formerly known as Eon Silicon Solution

(7) Micron plans to discontinue this flash memory device family. Intel does not recommend you using this flash memory device.



Manufacturer	Product Family	Density (Megabit)	Device Name
			MX25L8036E
		16	MX25L1635D
			MX25L1635E
			MX25L1636D
			MX25L1636E
		32	MX25L3225D
			MX25L3235D
			MX25L3235D
			MX25L3236D
			MX25L3237D
		64	MX25L6436E
			MX25L6445E
			MX25L6465E
		128	MX25L12836E
			MX25L12845E
			MX25L12865E
		256	MX25L25635E
			MX25L25735E
	MX25U	8	MX25U8035
			MX25U8035E
		16	MX25U1635E
		32	MX25U3235E
		64	MX25U6435E

Table 3. NAND Flash Memory Device Supported by PFL IP Core

Manufacturer	Density (Megabit)	Device Name
Micron	512	NAND512 1.8V
		NAND512 3.0V
		NAND512 3.3V
		Micron(MT29)
Samsung	512	K9F1208R0C
Toshiba	1000	TC58DVG02A1
Hynix	512	HY27US0812(1/2)B

Related Links

- Cypress website
- ESMT website

1.2.2 Supported Schemes and Features

The PFL IP core allows you to configure the FPGA in passive serial (PS) or fast passive parallel (FPP) scheme. The PFL IP core supports configuration with FPGA on-chip data compression and data encryption.

When you use compressed or encrypted configuration data for FPP configuration, the PFL IP core holds one data byte for one, two, four, or eight DCLK cycles to ensure the DCLK frequency runs at the required data rate as specified by the DCLK-to-DATA[] Ratio. The PFL IP core 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 IP core to specify whether the configuration file stored in the flash memory device is a compressed or uncompressed image.

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

You can program the Intel 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 IP core does not support Raw Binary File (`.rbf`) format.

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

1.3 Functional Description

The PFL IP core allows you to program flash memory devices with Intel CPLDs through the JTAG interface and provides the logic to control configuration from the flash memory device to the Intel FPGA.

1.3.1 Programming Flash Memory

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

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

Related Links

- Supported Flash Memory Devices on page 4
- Third-party Programmer Support on page 37
  Provides more information about programming the flash memory using third-party tools.

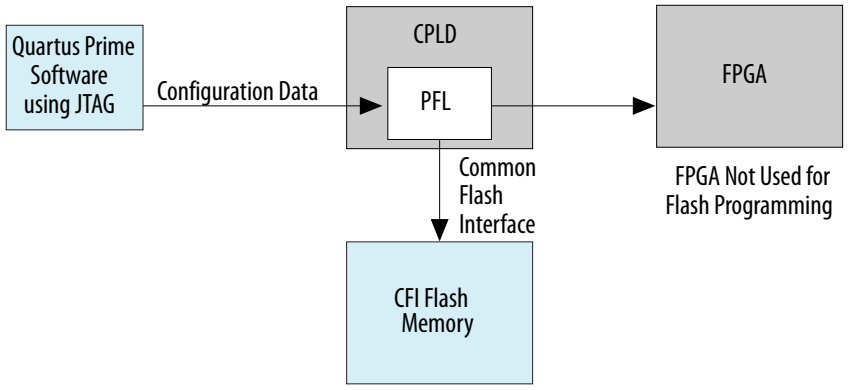
1.3.1.1 Programming CFI Flash

Intel 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 Intel CPLDs to indirectly program the flash memory device.

The Intel 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 Intel CPLD boundary-scan cells (BSCs). The PFL IP core provides JTAG interface logic to convert the JTAG stream provided by the Intel Quartus Prime software and to program the CFI flash memory devices connected to the CPLD I/O pins.

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

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

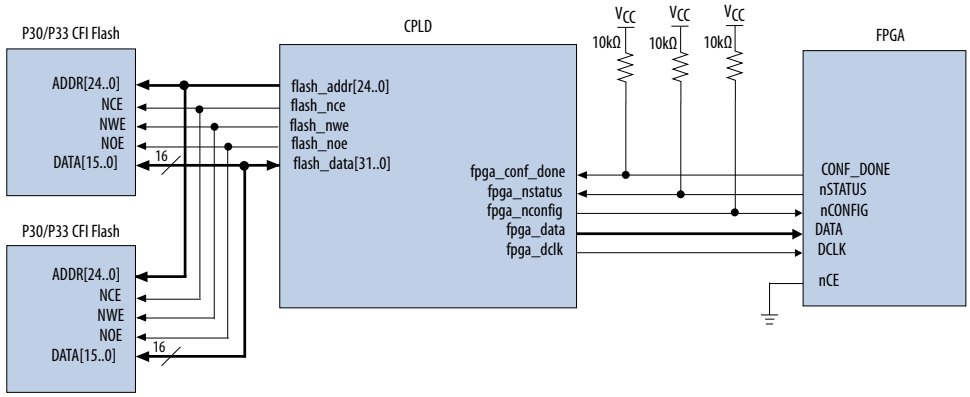


The PFL IP core supports dual MT28EW CFI flash memory devices in burst read mode to achieve faster configuration time. Two identical MT28EW CFI flash memory devices connect to the CPLD in parallel using the same data bus, clock, and control signals. During FPGA configuration, the FPGA DCLK frequency is four times faster than the flash_clk frequency.

Note: Dual mode is also supported in P30 and P33 devices.

Figure 2. PFL IP core with Dual MT28EW CFI Flash Memory Devices

The flash memory devices in the dual MT28EW CFI flash solution must have the same memory density from the same device family and manufacturer.



1.3.1.2 Programming Quad SPI Flash

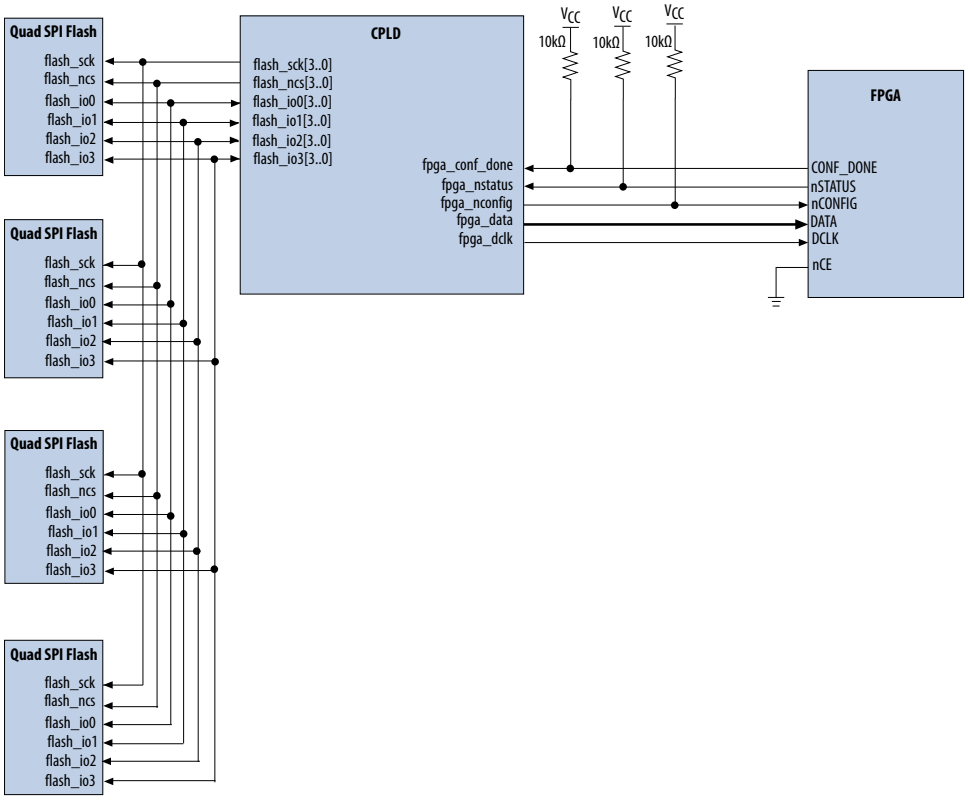
You can also use the JTAG interface in Intel CPLDs to program a quad SPI flash memory device with the PFL IP core.

The PFL IP core instantiated in the Intel CPLD functions as a bridge between the CPLD JTAG programming interface and the quad SPI flash memory device interface that connects to the Intel CPLD I/O pins. You can connect up to four identical quad SPI flashes in parallel to implement more configuration data storage.

Note: When connecting quad SPI flashes in parallel, use identical flash memory devices with the same memory density from the same device family and manufacturer.

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

Figure shows an Intel CPLD functioning as a bridge to program the quad SPI flash memory device through the JTAG interface. The PFL IP core supports multiple quad SPI flash programming of up to four devices.



Related Links

Supported Flash Memory Devices on page 4

1.3.1.3 Programming NAND Flash

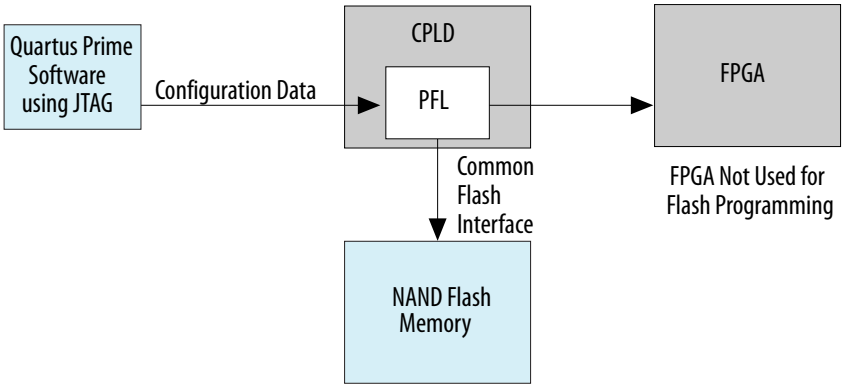
You can use the JTAG interface in Intel CPLDs to program the NAND flash memory device with the PFL IP core. The NAND flash memory device is a simpler device that has faster erase and write speed with higher memory density in comparison with the CFI flash.

You can use the JTAG interface in Intel 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 Intel

CPLD BSCs. The PFL IP core provides JTAG interface logic to convert the JTAG stream from the Intel Quartus Prime software and to program the NAND flash memory device that connects to the CPLD I/O pins.

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

Figure shows a CPLD functioning as a bridge to program the NAND flash memory device through the JTAG interface.

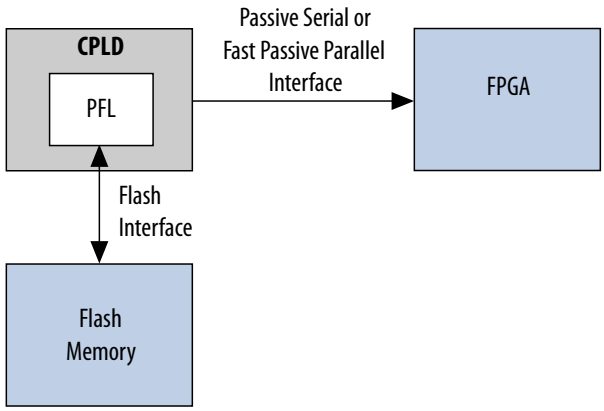


1.3.2 Controlling Intel FPGA Configuration from Flash Memory

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

Figure 5. FPGA Configuration With Flash Memory Data

Figure shows the Intel CPLD as the configuration controller for the FPGA. The flash memory includes CFI, quad SPI and NAND flash.



You can use the PFL IP core to either program the flash memory devices, configure your FPGA, or both; however, 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 Intel CPLD.
- You want to program the flash memory device with non-Intel FPGA data. For example, the flash memory device contains initialization storage for an ASSP. You can use the PFL IP core 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.

Creating Separate PFL Functions

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.

Note: 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.

Related Links

- Mapping PFL and Flash Address on page 12
- Implementing Page in the Flash .pof on page 14
- Using Enhanced Bitstream Compression and Decompression on page 17
- Using Remote System Upgrade on page 18

1.3.3 Mapping PFL and Flash Address

The address connections between the PFL IP core and the flash memory device vary depending on the flash memory device vendor and data bus width.

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

The address connection between the PFL IP core and the flash memory device are the same.

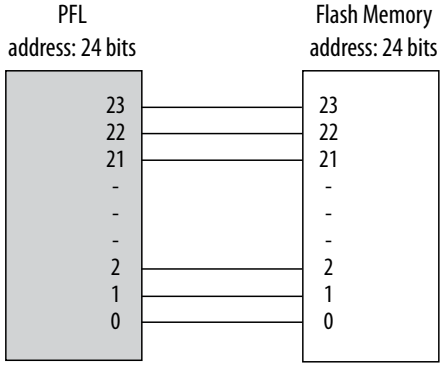


Figure 7. Micron J3, P30, and P33 Flash Memories in 16-Bit Mode

The flash memory addresses in Micron J3, P30, and P33 16-bit flash memory shift one bit down in comparison with the flash addresses in the PFL IP core. The flash address in the Micron J3, P30, and P33 flash memory starts from bit 1 instead of bit 0.

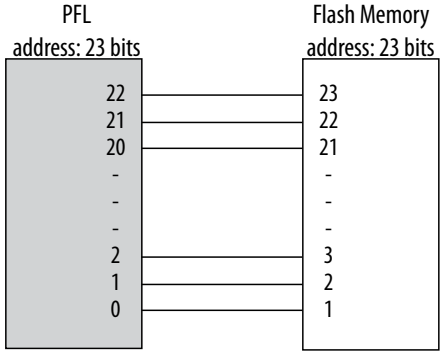


Figure 8. Cypress and Micron M28, M29, and MT28EW Flash Memory in 8-Bit Mode

The flash memory addresses in Cypress 8-bit flash shifts one bit up. Address bit 0 of the PFL IP core connects to data pin D15 of the flash memory.

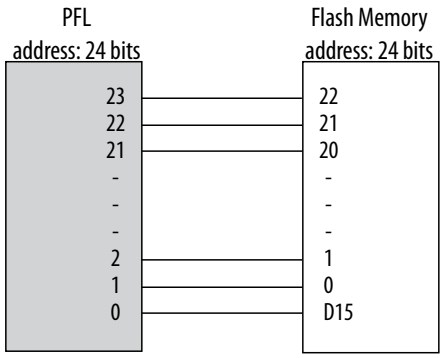
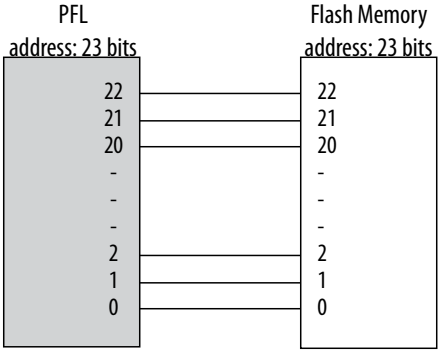


Figure 9. Cypress and Micron M28, M29, and MT28EW Flash Memory in 16-Bit Mode

The address bit numbers in the PFL IP core and the flash memory device are the same.



1.3.4 Implementing Page in the Flash .pof

The PFL IP core stores configuration data in a maximum of eight 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, the PFL IP core stores multiple SRAM Object Files (`.sof`) in the same page.

The total number of pages 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. You can locate the start address for each page 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 Intel Quartus Prime software to automatically determine the start address of the page. The Intel Quartus Prime 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.

Note: 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

1.3.4.1 Storing Option Bits

The PFL IP core 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. Option Bits Sector Format

Offset address 0x80 stores the .pof version required for programming flash memory. This .pof version applies to all eight pages of the configuration data. The PFL IP core requires the .pof version to perform a successful FPGA configuration process.

Sector Offset	Value
0x00–0x03	Page 0 start address
0x04–0x07	Page 1 start address
0x08–0x0B	Page 2 start address
0x0C–0x0F	Page 3 start address
0x10–0x13	Page 4 start address
0x14–0x17	Page 5 start address
0x18–0x1B	Page 6 start address
0x1C–0x1F	Page 7 start address
0x20–0x7F	Reserved
0x80[8]	.pof version
0x81–0xFF	Reserved

The Intel Quartus Prime 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 Intel Quartus Prime software 0x03. However, if you turn on the enhanced bitstream-compression feature, the value for the .pof version is 0x04.

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

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

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

- The end address depends on the density of the flash memory device. For the address range for devices with different densities, refer Byte Address Range table.
- You must specify the byte address for the option bits sector.

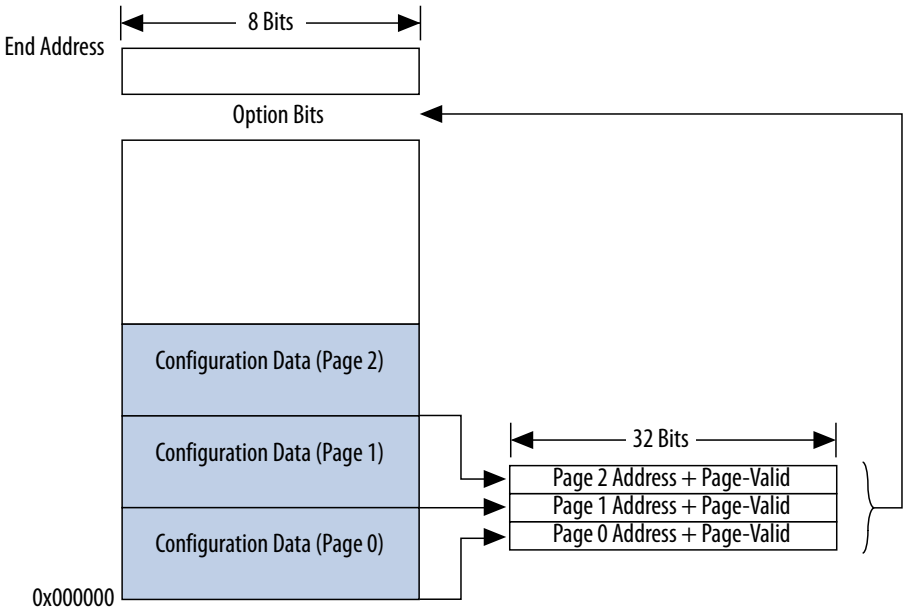


Figure 11. Page Start Address, End Address, and Page-Valid Bit Stored as Option Bits

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 IP core programs the Page-Valid bits after successfully programming the pages.

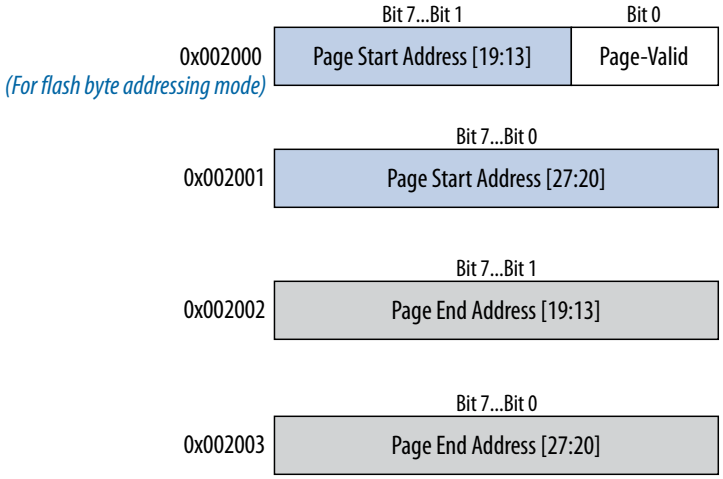


Table 5. Byte Address Range for CFI Flash Memory Devices with Different Densities

CFI Device (Megabit)	Address Range
8	0x0000000–0x00FFFFF
16	0x0000000–0x01FFFFF

continued...

CFI Device (Megabit)	Address Range
32	0x0000000–0x03FFFFF
64	0x0000000–0x07FFFFF
128	0x0000000–0x0FFFFFF
256	0x0000000–0x1FFFFFF
512	0x0000000–0x3FFFFFF
1024	0x0000000–0x7FFFFFF

1.3.5 Using Enhanced Bitstream Compression and Decompression

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

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(9)	Slow	Moderate(9)
FPP configuration time	Fast(10)	Very fast(11)	Not supported

Note: 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), a bit set notifies the PFL that the incoming files is a compressed file. The ×4 DCLK-to-data are handled automatically in the PFL.

Note: 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.

(9) The FPGA receives compressed bitstream which decreases the duration to transmit the bitstream to the FPGA.

(10) For FPP with on-chip bitstream decompression enabled, the DCLK frequency is ×2, ×4, or ×8 the data rate, depending on the device. 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.

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

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. FPGA Configuration Data Flow with Enhanced Bitstream Compression Feature in PS or FPP Configuration Scheme

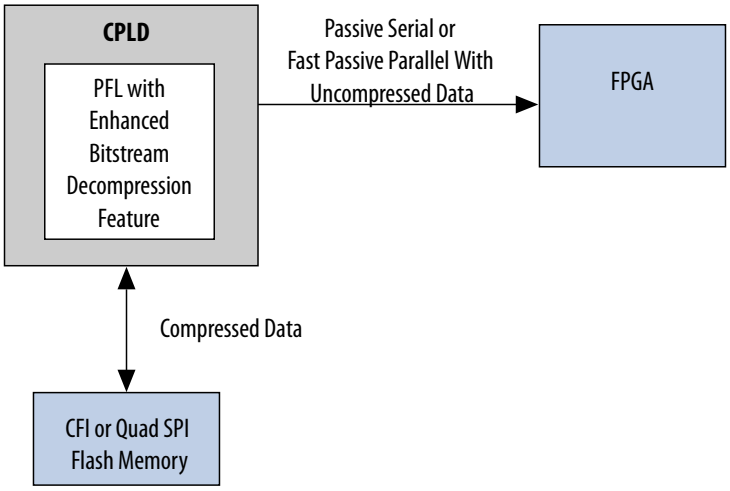
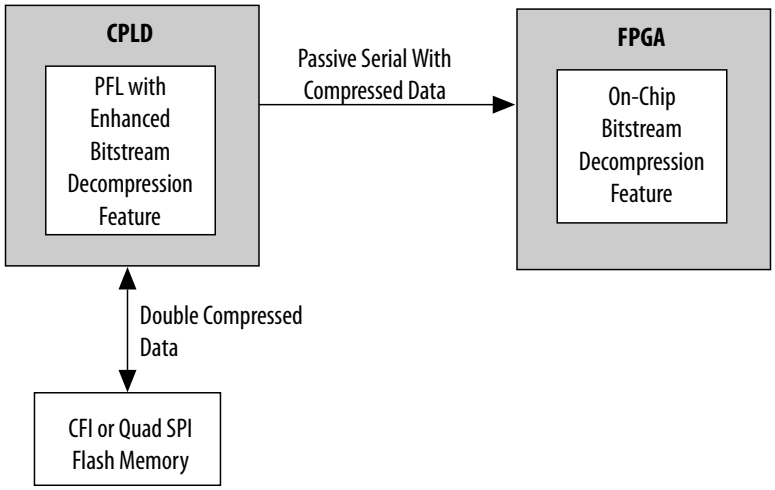


Figure 13. FPGA Configuration Data Flow with Double Compression Technique in PS Configuration Scheme



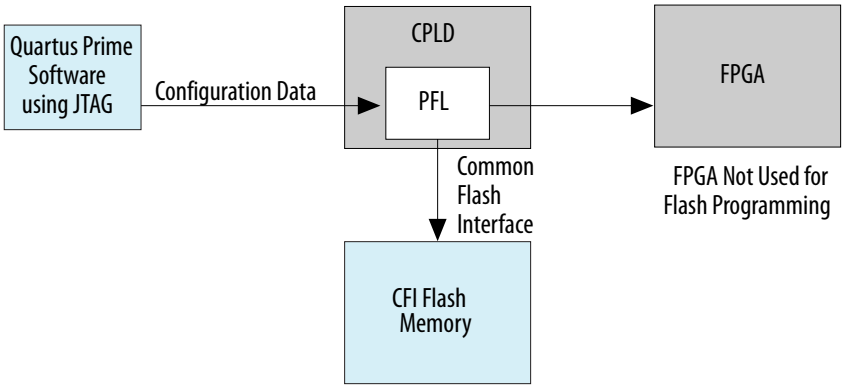
1.3.6 Using Remote System Upgrade

When you instantiate the PFL IP core in the Intel CPLD for FPP or PS configuration, you can use the features in the PFL IP core 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 IP core 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 IP core 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. Remote System Upgrade Implementation with the PFL IP Core in FPP and PS Configuration Scheme



1.3.6.1 Remote System Upgrade State Machine in the PFL IP Core

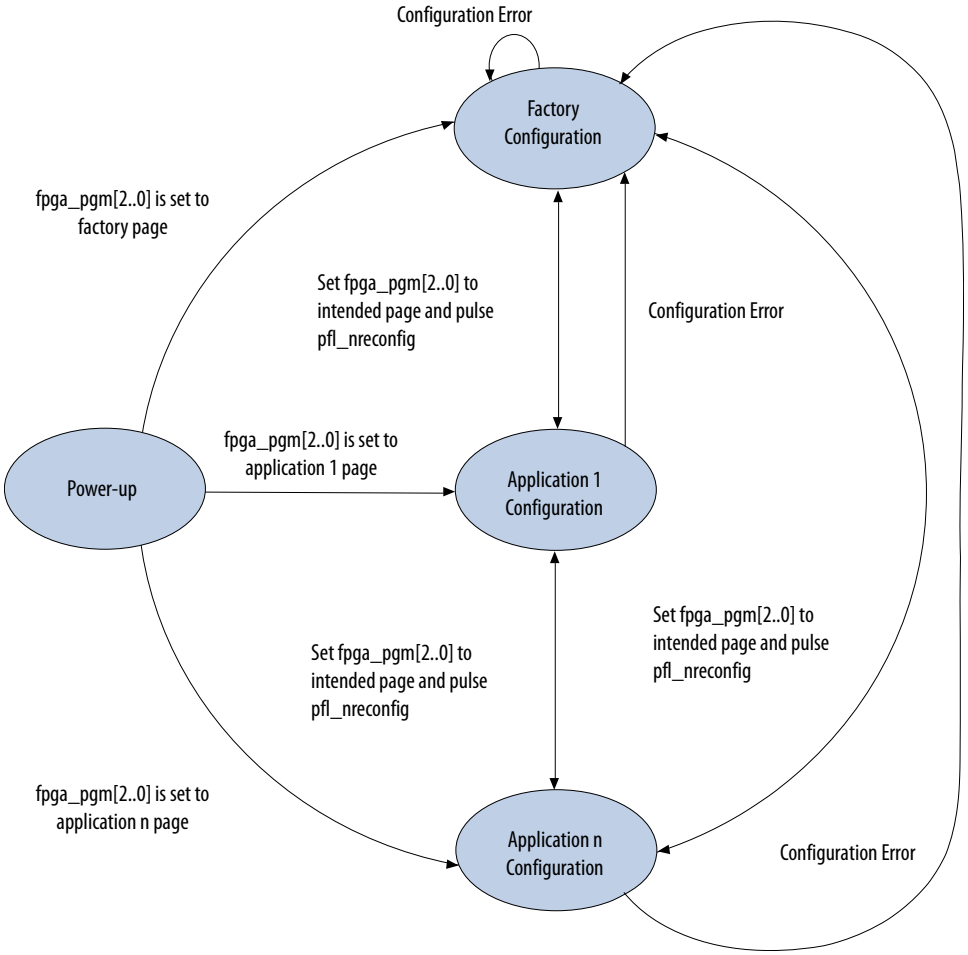
After the FPGA powers-up, you have the flexibility to determine whether a factory image or any application image is to be loaded by setting the `fpga_pgm[2..0]` input pin to the page in which the intended configuration image is stored.

If an error occurs while loading the configuration image, the PFL IP core 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 following these steps:

1. Set the `fpga_pgm[2.0]` input pin.
2. Release the `pfl_nreset` to high if the `pfl_nreset` is asserted to low.
3. After fifteen clock cycles, pulse the `pfl_nreconfigure` input pin to low.
4. Ensure that all transition is synchronized to `pfl_clk`.

Figure 15. Transitions Between Different Configurations in Remote System Upgrade

- The remote system upgrade feature in the PFL IP core does not restrict the factory image to page 0, but allows the factory image to be located on other pages in the flash.
- You can load the FPGA with either a factory image or any application image after power up, depending on the `fpga_pgm[2..0]` setting.



Note: The PFL IP core 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. Intel recommends that you write-protect the factory image blocks in the flash memory device.

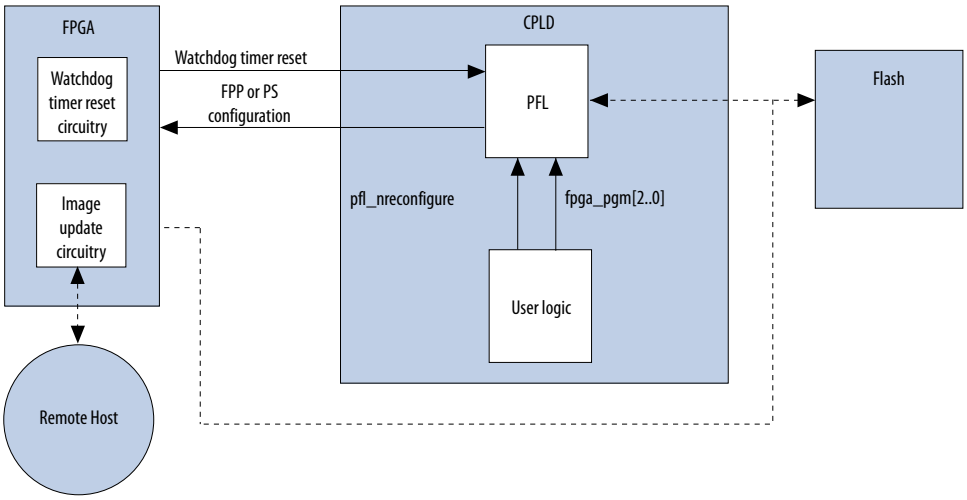
1.3.6.2 Implementing Remote System Upgrade with the PFL IP Core

You can achieve the remote system upgrade capabilities with the PFL IP core 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. The `pfl_nreconfigure` signal pulsed low for greater than one `pfl_clk` cycle.
- 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. Implementation of Remote System Upgrade with the PFL IP Core



1.3.6.3 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 the timer reaches the watchdog time out period. You must periodically reset this timer by asserting the `pfl_reset_watchdog` pin before the watchdog time-out period. If the timer does not reset before the watchdog time-out period, the PFL IP core detects watchdog time-out error and initiates a reconfiguration to load the factory image.

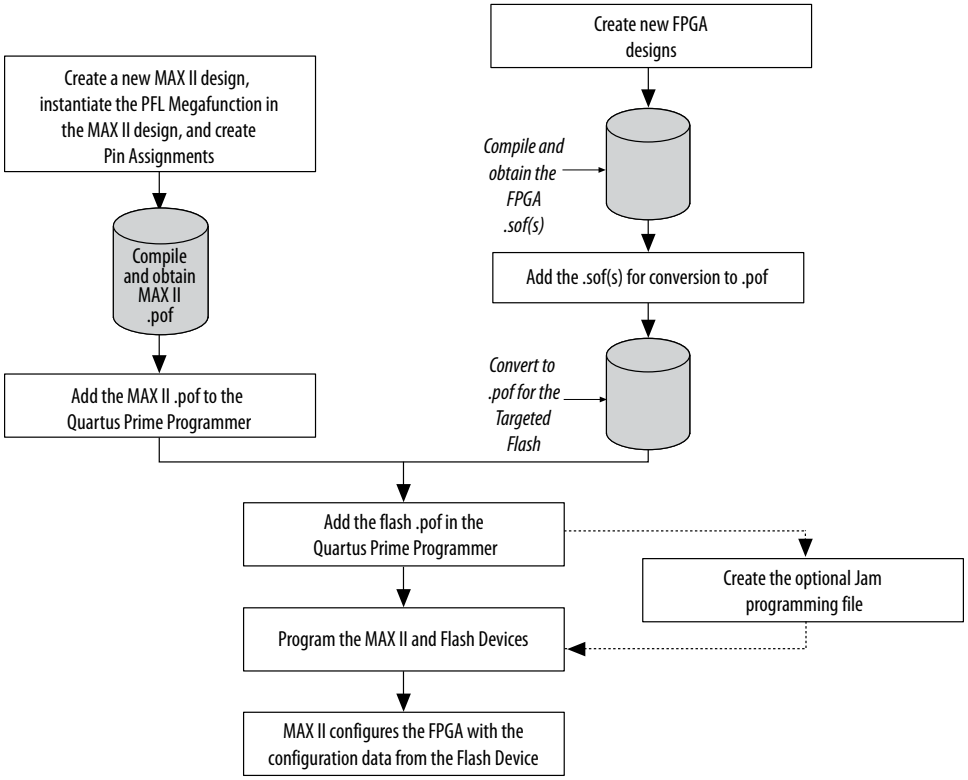
Instantiate the watchdog timer reset circuitry in the configuration image loaded into the FPGA. Connect one output signal from the reset circuitry to the `pfl_reset_watchdog` pin of the PFL in the CPLD to periodically send a reset signal to the user watchdog timer. To reset the watchdog timer correctly,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

1.4 Using the PFL IP Core

This section describes the procedures on how to use the PFL IP core.

Figure 17. Process for Using the PFL IP Core

Figure shows the process for using the PFL IP core, using MAX II as an example.



Related Links

AN478: Using FPGA-Based Parallel Flash Loader with the Quartus Prime Software
Provides more information about using the FPGA-based PFL IP core to program a flash memory device.

1.4.1 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.
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**. Under **Address mode for selected pages**, select **Auto** to let the Intel Quartus Prime software automatically set the start address for that page. Select **Block** to specify the start and end addresses, or select **Start** to specify the start address only.
7. Click **OK**.
8. 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, 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 Intel Quartus Prime software selects an address.

   Note: 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`.
9. 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 IP core. 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.
10. 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**.
11. 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
  a. Select `.sof` under **SOF Data**.
  b. Click **Properties**, and then turn on the **Compression** option.
  c. Click **OK**.
- Enhanced bitstream compression feature
  a. In the **Options** dialog box, turn on the **Enable enhanced bitstream-compression** when available option.
  b. Click **OK**.
- Double compression technique
  - Perform all the steps for the typical bitstream compression and enhanced bitstream compression features listed above.

Note: For more information about the compression feature in the PFL IP core, refer to "Using Enhanced Bitstream Compression and Decompression".

12. To generate programming files with encrypted data, select `.sof` under **SOF Data** and click **Properties**. Turn on the **Generate encrypted bitstream** check box.
13. Click **OK** to create the `.pof`.

Related Links

- Using Enhanced Bitstream Compression and Decompression on page 17
- Knowledge Center

1.4.2 Constraining PFL Timing

The PFL IP core supports the Intel Quartus Prime Timing Analyzer for accurate timing analysis on the Intel 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 Timing Analyzer.

Note: The Timing 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 Timing Analyzer, refer to the Intel Quartus Prime Timing Analyzer chapter in volume 3 of the Intel Quartus Prime Handbook.

Note: After you specify the timing constraint settings for the clock signal and for the asynchronous and synchronous input and output ports in the Timing 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.

Related Links

Quartus Prime TimeQuest Timing Analyzer of Quartus Prime Handbook
Provides more information about the TimeQuest analyzer.

1.4.2.1 Constraining Clock Signal

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

- 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. Constraining the clock signal is not mandatory.

You can constrain `pfl_clk` to the maximum frequency that the PFL IP core supports. 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 Timing Analyzer, follow these steps:

1. Run full compilation for the PFL design. Ensure that the timing analysis tool is set to `Timing Analyzer`.
2. After full compilation completes, on the **Tools** menu, select **Timing Analyzer** to launch the **Timing 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**.

1.4.2.2 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 IP core. The Intel Quartus Prime Fitter places and routes the input and output registers of the PFL IP core to meet the specified timing constraints.

Note: For more information about the synchronous input and output ports of the PFL IP core, refer to PFL Timing Constraints table.

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

Output delay value = Board delay from the PFL output port to the FPGA or flash input port + T_{SU} of FPGA or flash device.

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

To constrain the synchronous input and output signals in the Timing Analyzer, follow these steps:

1. Run full compilation for the PFL design. Ensure that the timing analysis tool is set to `Timing Analyzer`.
2. After full compilation completes, on the **Tools** menu, select **Timing Analyzer** to launch the **Timing 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 the **Unconstrained Paths** category, select **Setup Analysis**, and then click **Unconstrained Input Port Paths**.
5. 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.

Related Links

Summary of PFL Timing Constraints on page 26

1.4.2.3 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 IP core clock source. The internal structure of the PFL IP core handles the metastability of these asynchronous signals.

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

Note: 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.

1.4.2.4 Summary of PFL Timing Constraints

Table 7. PFL Timing Constraints

Type	Port	Constraint Type	Delay Value
Input clock	`pfl_clk`	`create_clock`	Can be constrained up to the maximum frequency supported by the PFL IP core.
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`	—
	`pfl_nreconfigure`	`set_false_path`	—

continued...

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 flash device data pins to CPLD data pins + T_{co} of flash device.
Output synchronous	fpga_data	set_output_delay	Board delay + T_{SU} of the FPGA
	fpga_dclk	set_output_delay	Board delay from fpga_dclk pin of the CPLD to DCLK pin of the FPGA

1.4.3 Simulating PFL Design

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

Note: 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. Intel certifies the PFL IP core based on actual hardware testing, and not through PFL simulation. The PFL simulation only provides primitive behavioral simulation.

Table 8. Files Required for PFL Simulation in the ModelSim-Intel FPGA Software

File/Library	Description
.vo or .vho	The Verilog HDL or VHDL output file of the PFL IP core.
.sdo	The Standard Delay Format Output file (.sdo) of the PFL IP core.
Simulation libraries: • altera • altera_mf • maxii • maxv	The precompiled library files for Intel FPGA IP core primitives and Intel CPLDs in the ModelSim-Intel FPGA software.
Test bench	Test bench file to establish the interface between the PFL IP core and the flash memory device.
Flash simulation model files	The simulation model files for the flash memory devices 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.

Related Links

- Getting Started with Quartus II Simulation Using the ModelSim-Altera Software User Guide
- Intel FPGA Knowledge Base

1.4.3.1 Creating a Test Bench File for PFL Simulation

You can use a test bench file to establish the interface between the PFL IP core and the flash memory device. You must map the input and output ports of the PFL IP core 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.

To establish the connection between the PFL IP core 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.

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>),
            );
```

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

1.4.3.2 Performing PFL Simulation in the ModelSim-Intel FPGA Software

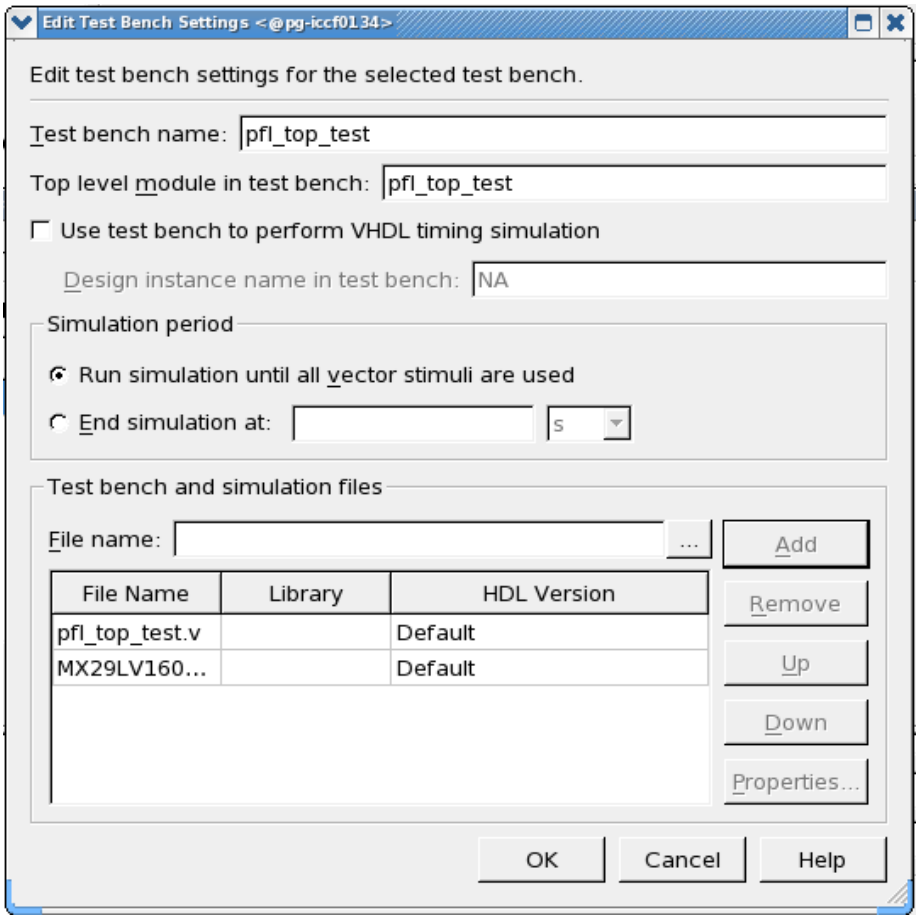
To perform PFL simulation in the ModelSim-Intel FPGA software, you must specify the `.sdo` or load the ModelSim precompiled libraries listed in Files Required for PFL Simulation in the ModelSim-Intel FPGA Software table. Alternatively, you can generate the `.vo`, `.sdo` and Modelsim precompiled libraries through NativeLink feature in Intel Quartus Prime.

To set up the simulation using NativeLink and perform ModelSim simulation, follow these steps:

1. On the **Assignments** menu, click **Settings** to open the **Settings dialog box** and then under **EDA Tool Settings**, click **Simulation**.
2. Verify that ModelSim-Intel FPGA is selected in the **Tool** name field and click **OK**.
3. To run simulation right after design compilation, turn on the**Run gate-level simulation automatically after compilation** option.
4. Specify **Format for output netlist**, **Time scale**, and **Output directory**.

5. Under **NativeLink** settings, select **Compile test bench** then click **Test Benches**.
6. In the **Test Bench** dialog box appears, click **New**. Fill in the settings, insert simulation model files for the flash memory devices and test bench.

Figure 18. Test Bench Settings



7. After settings are done, compile the design and the simulation starts automatically.

Related Links

Simulating PFL Design on page 27

1.4.3.3 Performing PFL Simulation for FPGA Configuration

Before beginning the FPGA configuration, the PFL IP core 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 IP core first reads from the final address, which is `0×800080`, to obtain the `.pof` version information. Because

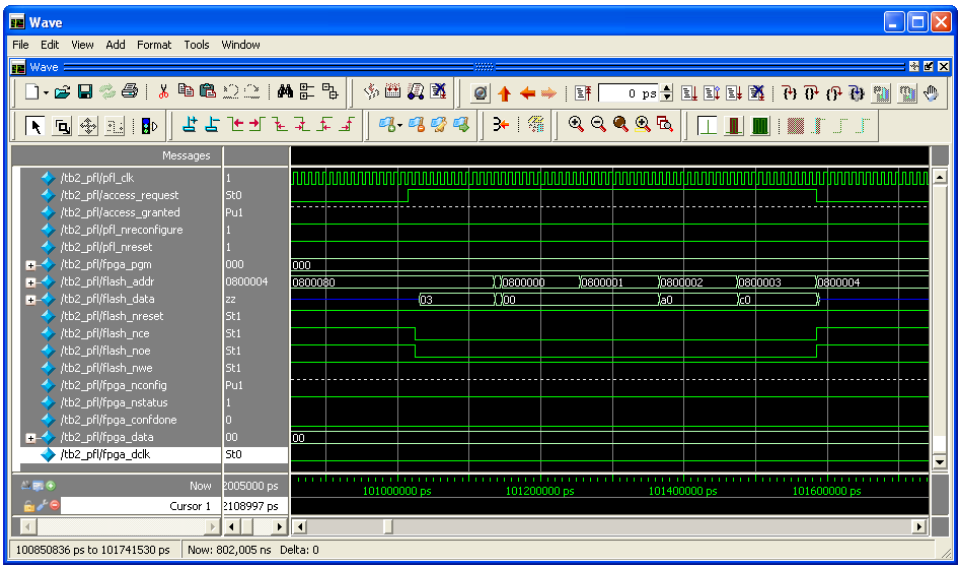
fpga_pgm[2..0] is set to 000, the PFL IP core reads from address 0×800000 to address 0×800003 to get the start and end address of page 0 and the Page-Valid bit. The LSB in address 0×800000 is the Page-Valid bit.

The Page-Valid bit must be 0 for the PFL IP core to proceed with FPGA configuration. While the PFL IP core 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.

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

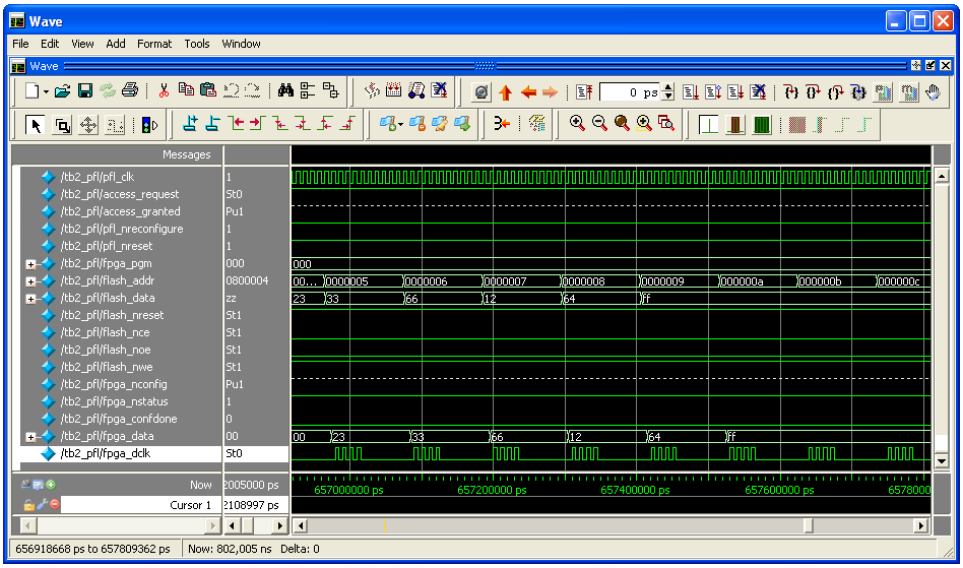
Figure 19. Simulation Before Configuration

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



After reading the option bits for page 0, the PFL IP core 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 IP core asserts the flash_nce and flash_noe signals low, and the pfl_flash_access_request signal high.

Figure 20. Simulation When FPGA Configuration Starts



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 IP core 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.

1.4.4 Programming Intel CPLDs and Flash Memory Devices

Using the Intel Quartus Prime Programmer, you can program Intel 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 Intel Quartus Prime Programmer window and click **Add File** to add the `.pof` for the CPLD.
2. Right-click the CPLD `.pof` and click **Attach Flash Device**.
3. In the **Flash Device** menu, select the density of the flash memory device to be programmed.
4. Right-click the necessary flash memory device density and click **Change File**.
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 Intel Quartus Prime 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 IP core.

Note: The Intel Quartus Prime programmer erases the flash memory device if you select the `.pof` of the flash memory device before programming. To prevent the Intel Quartus Prime Programmer from erasing other sectors in the flash memory device, select only the pages, `.hex` data, and option bits.

1.4.4.1 Programming Intel CPLDs and Flash Memory Devices Separately

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

1. Open the Intel Quartus Prime 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.

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

   Note: 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 IP core handles the 512-Mb `.pof` programming to the four quad SPI flash memory devices.
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.

1.4.5 Defining New CFI Flash Device

The PFL IP core supports Intel-compatible and AMD-compatible flash memory devices. In addition to the supported flash memory devices, 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. The following table lists the three functions available in the Define CFI Flash Device window.

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.
3. In the **New CFI Flash Device** dialog box, specify or update the parameters of the new flash memory device. You can obtain the values for these parameters from the datasheet of the flash memory device manufacturer.

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	Turn on the option 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

Note: You must specify either the word programming time parameters, buffer programming time parameters, or both. Do not leave both programming time parameters with 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**.

Related Links

Supported Flash Memory Devices on page 4

1.4.6 Programming Multiple Flash Memory Devices

The PFL IP core supports multiple-flash programming of as many as 16 flash memory devices. This feature allows the PFL IP core 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, use the content in the flash memory device that connects to the `nCE[0]` pin as configuration data.

To use the multiple flash programming feature, follow these steps:

1. Select the number of flash memory devices connected to the CPLD in the PFL IP core 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 Intel Quartus Prime 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.
4. Attach the flash memory device `.pof` to each flash memory device.
5. Check the boxes in the Intel Quartus Prime Programmer for the necessary operation and click **Start**.

1.4.7 Creating Jam Files for Intel CPLDs and Flash Memory Device Programming

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

1. Open the Intel Quartus Prime Programmer window and click **Add File** to add the `.pof` for the CPLD.
2. Right-click the CPLD `.pof` and click **Attach Flash Device**.
3. In the **Flash Device** menu, select the density of the flash memory device to be programmed.
4. Right-click the necessary flash memory device density and click **Change File**.
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. On the **File** menu, point to **Create/Update** and click **Create JAM, JBC, SVF, or ISF File**.
7. Enter a name and select the file format (`.jam`).
8. Click **OK**.

Note: Use the `.jam` files with the Intel Quartus Prime Programmer or `quartus_jli` executable file.

Related Links

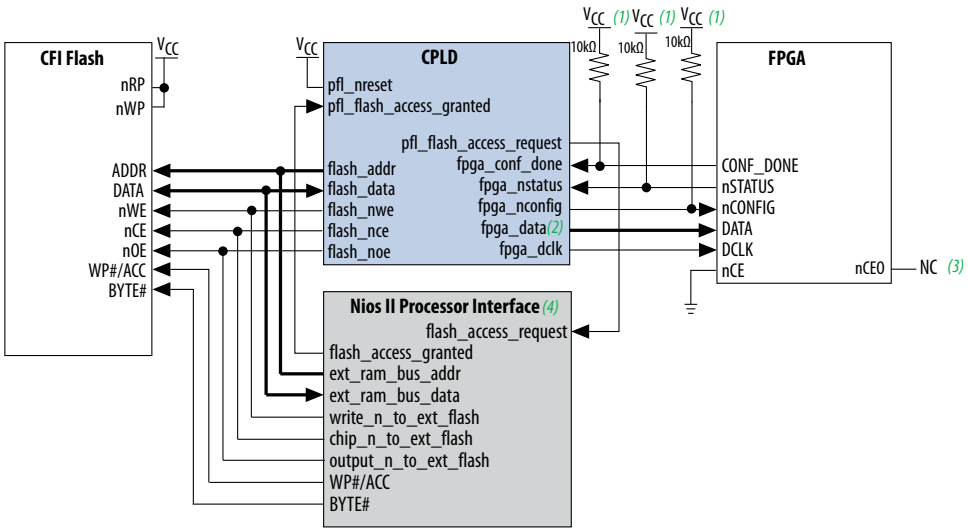
AN425: Using the Command-Line Jam STAPL Solution for Device Programming.
Provides more information about the quartus_jli executable.

1.5 PFL IP Core In Embedded Systems

The PFL IP core allows processors, such as the Nios® II processor, to access the flash memory device while programming flash and configuring an FPGA.

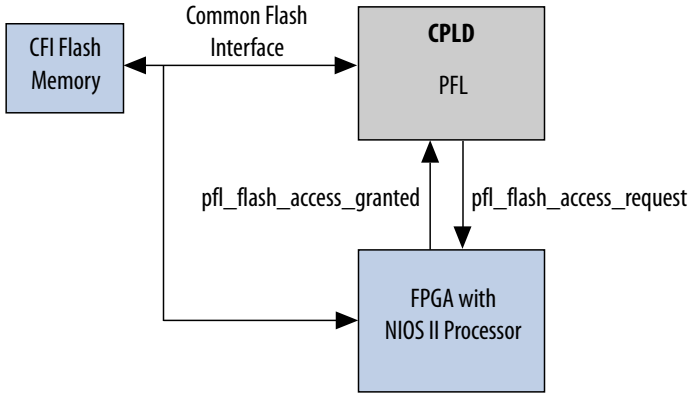
The following figure shows how you can use the PFL IP core to program the flash memory device and to configure the FPGA with a Nios II processor. The configured Nios II processor uses the non-configuration data stored in the same flash memory device.

Figure 21. Single-Device Configuration Using the PFL With the Controller



(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) You can use the Nios II processor in other FPGA, except when you are configuring the FPGA.

Figure 22. Relationship Between the Four Sections in the Design Example



You must configure the Intel 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 IP core to configure the image to the Intel FPGA. If you store the Nios II processor image in the same flash memory device you intend to program, do not 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, for example an enhanced configuration (EPC) device or an erasable programmable configurable serial (EPCS) memory.

In Relationship Between the Four Sections in the Design Example figure above, the Nios II processor and the PFL IP core share the same bus line to the flash memory device. However, to avoid data contention, the processor and the IP core cannot

access or program the flash memory device at the same time. To ensure that only one controller (the processor or the IP core), is accessing the flash memory device at any given time, you must tri-state all output pins from one controller to the flash memory device, 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 IP core.

Table 11. PFL Flash Access Pins and Functions

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

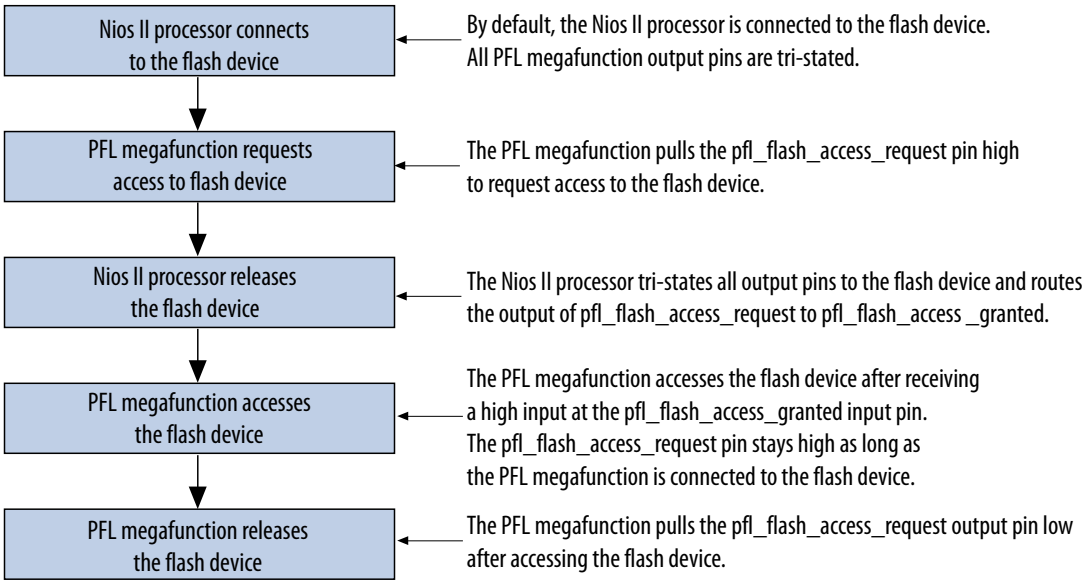
Table 12. `pfl_flash_access_request` and `pfl_flash_access_granted` Pins With the Nios II and PFL IP Core

Table 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.

Signal	Nios II Processor	PFL IP Core
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.

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

Figure 23. Nios II Processor and PFL Accessing the Flash Memory Device Sequence



Note: Intel recommends that you enable the safe state machine setting to prevent the PFL IP core 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 Intel 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.

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 IP core connect to your processor using the method in `pfl_flash_access_request` and `pfl_flash_access_granted` Pins With the Nios II and PFL IP Core table above.

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 IP core 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.

Related Links

Nios II Processor Reference Handbook
Provides more information about the Nios II processor.

1.6 Third-party Programmer Support

You can program the flash memory using a third-party programmer instead of using Parallel Flash Loader IP core. To program using third-party programmer, you need to convert the `.pof` to an `.rbf` by following the steps below:

1. Compile and generate a `.pof` file for the flash memory device.
2. Convert `.pof` file to `.hexout` file using the following command:

```
quartus_cpf -c <pof_file_base_name>.pof <hex_file_base_name>.hexout
```

3. Convert `.hexout` file created above to `.rbf` using the nios2-elf-objcopy command on Nios II Command Shell:

```
nios2-elf-objcopy –I ihex –O binary <input file>.hexout <output file>.rbf
```

Related Links

- Programming Flash Memory on page 8
- NIOS II Command-Line Tools
  Provides more information on the nios2-elf-objcopy command.

1.7 Parameters

Table 13. PFL General Parameters

Options	Value	Description
Operating mode	• Flash Programming and FPGA Configuration • Flash Programming • FPGA Configuration	Specifies the operating mode of flash programming and FPGA configuration control in one IP core or separate these functions into individual blocks and functionality.
Targeted flash device	• CFI Parallel Flash • Altera Active Serial ×4 • Quad SPI Flash • NAND Flash	Specifies the flash memory device connected to the PFL IP core.
Tri-state flash bus	On or Off	Allows the PFL IP core to tri-state all pins interfacing with the flash memory device when the PFL IP core does not require an access to the flash memory.

Table 14. PFL Flash Interface Setting Parameters

Options	Value	Description
Number of flash devices used	• CFI Parallel Flash: 1–16 • Altera Active Serial ×4: 1, 2, 4, 8 • Quad SPI Flash: 1, 2, 4, 8 • NAND Flash: 1	Specifies the number of flash memory devices connected to the PFL IP core.
Largest flash density	• CFI Parallel Flash: 8 Mbit–2 Gbit • NAND Flash: 512 Mbit • 1 Gbit - Micron (MT29) • Altera Active Serial x4: EPCQ 256 Mbit	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 IP core, specify the largest flash memory device density. For dual mode CFI and NAND flash devices, 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**.)
Flash interface data width	• CFI Parallel Flash: 8, 16, or 32 bits • NAND Flash: 8 bits or 16 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	Creates a `flash_nreset` pin in the PFL IP core 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. When using a GL flash device, connect this pin to the RESET# pin of the flash device. (Available only if you select **CFI Parallel Flash**.)
Quad SPI flash device manufacturer	• Macronix • Micron • Spansion	Specifies the device manufacturer of the quad SPI flash. (Available only if you select **Quad SPI Flash**.)

continued...

Options	Value	Description
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 IP core encounters. Intel 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	Enables the support for on-die ECC. Certain NAND flash memory devices has on-die ECC. Allows the PFL IP core to use the on-die ECC of the flash memory device. Turning off this option allows the PFL IP core to generate its own ECC engine. (Available only if you select **NAND Flash**.)

Table 15. PFL Flash Programming Parameters

Options	Value	Description
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 IP core uses more LEs. If you optimize the PFL IP core for area, the IP core uses less LEs, but the flash programming time is longer. (Available only if you select **CFI Parallel Flash**.)
FIFO size	16, 32	Specifies the FIFO size if you select Speed for flash programming IP optimization. The PFL IP core 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	Adds a block to accelerate verification. (Available only if you select **CFI Parallel Flash**.)

Table 16. PFL FPGA Configuration Parameters

Options	Value	Description
External clock frequency	—	Specifies the user-supplied clock frequency for the IP core 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 IP core 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 that a flash memory device requires from the flash datasheet. Intel 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 requires 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. See related for more information about option bits.

continued...

Options	Value	Description
FPGA configuration scheme	• PS • FPP • FPP ×16 (for Stratix V devices) • FPP ×32(for Stratix V devices)	Select the FPGA configuration scheme. The default FPP is FPP ×8. 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 IP core reconfigures the FPGA with data from the same page of the failure. • If you select **Retry from fixed address**, the PFL IP core reconfigures the FPGA with data from a fixed address in the next option field after failure.
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 IP core to read from the reconfiguration for a configuration failure.
Include input to force reconfiguration	On or Off	Includes an optional reconfiguration input pin (`pfl_nreconfigure`) to enable a reconfiguration of the FPGA.
Watchdog timer	On or Off	Enables a watchdog timer for remote system upgrade support. Turning on this option enables the `pfl_reset_watchdog` input pin and `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` generate one `fpga_dclk`. • Ratio 4 means every four external clocks to `pfl_clk` generate one `fpga_dclk`. • Ratio 2 means every two external clocks to `pfl_clk` generate one `fpga_dclk`. • Ratio 1 means every one external clock to `pfl_clk` generate one `fpga_dclk`.
Use advance read mode	• Normal Mode • Intel Burst Mode with 3 LC (P30 or P33) • Intel Burst Mode with 4 LC (P30 or P33) • Intel Burst Mode with 5 LC (P30 or P33) • Spansion Page Mode (GL) • Micron 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 with 3/4/5 LC (P30 or P33)—Applicable for Micron P30 and P33 flash memory only. Reduces sequential read access time. The flash sends valid data after 3/4/5 clock cycles. • Spansion page mode—Applicable for Cypress GL flash memory only • Micron burst mode—Applicable for Micron 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 • Speed	Select to enable or disable the enhanced bitstream decompression block.

Options	Value	Description
		• 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 IP core. • 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.

Related Links

Storing Option Bits on page 14

1.8 Signals

This section contains information about the PFL IP core input and output signals.

Table 17. PFL Signals

For maximum FPGA configuration DCLK frequencies, refer to the Configuration Handbook.

Pin	Input or Output	Weak Pull-Up	Function
pfl_nreset	Input	—	Asynchronous reset for the PFL IP core. Pull high to enable FPGA configuration. To prevent FPGA configuration, pull low when you do not use the PFL IP core. This pin does not affect the flash programming functionality of the PFL IP core.
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 IP core 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	Input	—	User input clock for the device. Frequency must match the frequency specified in the IP core and must not be higher than the maximum DCLK frequency specified for the specific FPGA during configuration. This pins are not available for the flash programming option in the PFL IP core.
fpga_pgm[]	Input	—	Determines the page for the configuration. This pins are not available for the flash programming option in the PFL IP core.
fpga_conf_done	Input	10-kW 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. This pins are not available for the flash programming option in the PFL IP core.
fpga_nstatus	Input	10-kW 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 IP core stops reading the data from the flash memory device. This pins are not available for the flash programming option in the PFL IP core.

continued...

Pin	Input or Output	Weak Pull-Up	Function
pfl_nreconfigure	Input	—	After pfl_nreset asserted high for at least fifteen clock cycles, the subsequent low signal at this pin initiates the FPGA reconfiguration. For more flexibility in controlling the FPGA reconfiguration, you can reconnect this pin to a switch to set this input pin high or low. When FPGA reconfiguration is initiated, the fpga_nconfig pin is pulled low to reset the FPGA device. The pfl_clk pin registers this signal. This pins are not available for the flash programming option in the PFL IP core.
pfl_flash_access_request	Output	—	Used for system-level synchronization. When necessary, this pin connects to a processor or an arbitrator. The PFL IP core drives this pin high when the JTAG interface accesses the flash or the PFL IP core configures the FPGA. This output pin works in conjunction with the flash_noe and flash_nwe pins.
flash_addr[]	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. The output of this pin depends on the setting of the unused pins if you did not select the PFL interface tri-state option when the PFL is not accessing the flash memory device.
flash_data[]	Bidirectional	—	Data bus to transmit or receive 8- or 16-bit data to or from the flash memory in parallel. The output of this pin depends on the setting of the unused pins if you did not select the PFL interface tri-state option when the PFL is not accessing the flash memory device. (12)
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	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. Use this pin 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.

continued...

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

Pin	Input or Output	Weak Pull-Up	Function
flash_nadv	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. Use this pin 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.
flash_nreset	Output	—	Connects to the reset pin of the flash memory device. A low signal resets the flash memory device.
fpga_data[]	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. This pins are not available for the flash programming option in the PFL IP core.
fpga_dclk	Output	—	Connects to the DCLK pin of the FPGA. Clock input data to the FPGA device during configuration. This pins are not available for the flash programming option in the PFL IP core.
fpga_nconfig	Open Drain Output	10-kW Pull-Up Resistor	Connects to the nCONFIG pin of the FPGA. A low pulse resets the FPGA and initiates configuration. This pins are not available for the flash programming option in the PFL IP core. (12)
flash_sck[]	Output	—	Clock source for flash data read operation. Connects to the CLK input pin of the quad SPI flash. If you use more than one quad SPI flash, 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 you use more than one quad SPI flash, 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[]	Bidirectional	—	The first bit of the data bus to or from the quad SPI flash. If you use more than one quad SPI flash, 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[]	Bidirectional	—	The second bit of the data bus to or from the quad SPI flash. If you use more than one quad SPI flash, 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[]	Bidirectional	—	The third bit of the data bus to or from the quad SPI flash. If you use more than one quad SPI flash, 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[]	Bidirectional	—	The fourth bit of the data bus to or from the quad SPI flash. If you use more than one quad SPI flash, connect this pin to the fourth bit of the data bus of

continued...

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

1.9 Specifications

This section provides the equations to estimate the time for reconfiguring the FPGA with the PFL IP core.

The equations in the following table 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). Only the larger number between C_{flash} and Ccfg is important because reading from the flash and clocking out the data for configuration are performed in parallel.
- F_{clk} is the input clock frequency to the PFL IP core.
- 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 Cypress flash memory devices and is only applicable for page mode access. T_{page_access} is set to 30 ns in the PFL IP core.
- N is the number of bytes to be clocked out. This value is obtained from the .rbf for the specific FPGA.

Table 18. FPP and PS Mode Equations for the PFL

Flash Access Mode	Configuration Data Option	Flash Data Width (bits)	DCLK Ratio = 1, 2, 4, or 8 (13)	
			FPP Mode	PS Mode
Normal Mode/ Page Mode	Normal	8	$C_{flash} = C_{access}$ C_{cfg} = DCLK Ratio $C_{overhead} = 5*C_{access}$	$C_{flash} = C_{access}$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 5*C_{access}$
		16	$C_{flash} = C_{access}/2$ C_{cfg} = DCLK Ratio $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 3*C_{access}$
	Compressed or encrypted or both	8	$C_{flash} = C_{access}$ C_{cfg} = 4*DCLK Ratio	$C_{flash} = C_{access}$ C_{cfg} = 8*DCLK Ratio

continued...

(13) Ratio between input clock and DCLK output clock. For more information, see related information

Flash Access Mode	Configuration Data Option	Flash Data Width (bits)	DCLK Ratio = 1, 2, 4, or 8 (13)	
			FPP Mode	PS Mode
			$C_{overhead} = 5*C_{access}$	$C_{overhead} = 5*C_{access}$
		16	$C_{flash} = C_{access}/2$ C_{cfg} = 4*DCLK Ratio $C_{overhead} = 3*C_{access}$	$C_{flash} = C_{access}/2$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 3*C_{access}$
Burst Mode	Normal	4	$C_{flash} = 4$ C_{cfg} = DCLK Ratio $C_{overhead} = 48$	$C_{flash} = 4$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 48$
		8	$C_{flash} = 2$ C_{cfg} = DCLK Ratio $C_{overhead} = 22*C_{access}+8$	$C_{flash} = 2$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 22*C_{access}+8$
		16	$C_{flash} = 1$ C_{cfg} = DCLK Ratio $C_{overhead} = 20*C_{access}+8$	$C_{flash} = 1$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 20*C_{access}+8$
	Compressed or encrypted or both	4	$C_{flash} = 4$ C_{cfg} = 4*DCLK Ratio $C_{overhead} = 48$	$C_{flash} = 4$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 48$
		8	$C_{flash} = 2$ C_{cfg} = 4*DCLK Ratio $C_{overhead} = 22*C_{access}+8$	$C_{flash} = 2$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 22*C_{access}+8$
		16	$C_{flash} = 1$ C_{cfg} = 4*DCLK Ratio $C_{overhead} = 20*C_{access}+8$	$C_{flash} = 1$ C_{cfg} = 8*DCLK Ratio $C_{overhead} = 20*C_{access}+8$

- 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
  $C_{access} = [(T_{access}*F_{clk}+1) + ((T_{page\_access}*F_{clk}+1)*15)]/16$
  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
- For FPP (×8)
  Total Clock Cycles (from nRESET asserted high to N bytes of data clocked out)
  C_{flash} remains the same
- For FPP (×16)
  Total Clock Cycles (from nRESET asserted high to N word of data clocked out)
  $C_{flash} = C_{flash} \times 2$ (×2 of C_{flash} from FPP ×8)
- For FPP (×32)
  Total Clock Cycles (from nRESET asserted high to N double word of data clocked out)
  $C_{flash} = C_{flash} \times 4$ (×4 of C_{flash} from FPP ×8)

(13) Ratio between input clock and DCLK output clock. For more information, see related information

1.9.1 Configuration Time Calculation Examples

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

Note: Any reference to the core clock speed of 100 MHz is only an example of the configuration time calculation and not a recommendation of the actual clock.

Example 2. Normal Mode

- 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

- Use the following formulas in this calculation:

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

  C_{flash} for Normal Mode = C_{access} / 2

  $C_{cfg} = 2$

  $C_{overhead} = 3 * C_{access}$

  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

  $C_{overhead}$ = 3*11 = 33

  Total Clock Cycles = 33 + 5.5 * 590848 = 3249697

  Total Configuration Time at 100 MHz = 3249697/ 100 × 106 = 32.5ms

Example 3. Page Mode

- 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
- Use the following formulas in this calculation:
  T_{page_access} = 30 ns
  $C_{access} = [(T_{access}*F_{clk}+1) + ((T_{page\_access}*F_{clk}+1)*15)]/16$
  C_{flash} for Page Mode = C_{access} / 2
  C_{cfg} = 2
  $C_{overhead}$ = 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
  $C_{overhead}$ = 3*3.5 = 10.5
  Total Clock Cycles = 10.5 + 2.5*590848 = 1477130.5
  Total Configuration Time at 100 MHz = 1477130.5 / 100 × 106 = 14.77 ms

Example 4. Burst Mode

- 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
- Use the following formulas 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 × 106 = 11.82 ms

Example 5. Single Quad SPI Flash

- Single quad SPI flash configuration time calcualtion

  .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
- Use the following formulas 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 × 106 = 23.63 ms

Example 6. Four Cascaded Quad SPI Flashes

- Four cascaded quad SPI flashes configuration time calcualtion:

  .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.

1.10 Parallel Flash Loader IP Core User Guide Archives

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
16.0	Parallel Flash Loader IP Core User Guide
15.0	Parallel Flash Loader IP Core User Guide
14.1	Parallel Flash Loader IP Core User Guide

1.11 Document Revision History for Intel FPGA Parallel Flash Loader IP Core User Guide

Date	Version	Changes
November 2017	2017.11.06	• Updated the *Use advance read mode* parameter to include latency count options for Intel Burst mode in *PFL FPGA Configuration Parameters* table. • Changed instances of Spansion to Cypress and added note stating Cypress was formerly known as Spansion. • Updated values for FIFO size parameter in *PFL Flash Programming Parameters*. • Added link to JEDEC CFI standard in related links. • Updated clock cycles to fifteen before pulsing input pin to low in *Remote System Upgrade State Machine in the PFL IP Core*. • Updated description for `pfl_nreconfigure` signal in *PFL Signals* table. • Rebranded to Intel. • Added Micron MT28EW CFI flash product family support. • Added note to all discontinued supported Micron CFI flash memory. • Updated `flash_io0[]`, `flash_io1[]`, `flash_io2[]` and `flash_io3[]` from output to bidirectional in *PFL Signals* table. • Changed instances of EON Silicon Solution to ESMT and added note stating ESMT was formerly known as EON Silicon Solution. • Updated Largest flash density parameter description.
October 2016	2016.10.31	• Updated Micron flash in *CFI Flash Memory Devices Supported by PFL IP Core*. • Removed FS Spansion flash in *Quad SPI Flash Memory Device Supported by PFL IP Core*. • Updated parameter value in *PFL Flash Interface Setting Parameters*.
June 2016	2016.06.01	• Corrected DCLK ratio and C_{cfg} in Normal Mode and Page Mode examples. • Added N25Q256 in *Quad SPI Flash Memory Device Supported by PFL IP Core*. • Removed S25FS512S in *Quad SPI Flash Memory Device Supported by PFL IP Core*. • Edited Parameters table by breaking grouped tables. • Updated *Number of flash devices used* and *Largest flash density* options in *PFL Flash Interface Setting Parameters* table
May 2016	2016.05.02	• Changed instances of Quartus II to Quartus Prime. • Added steps to set up PFL simulation using NativeLink and perform ModelSim simulation. • Corrected constraint type for `fpga_data` and `fpga_dclk`. • Corrected DCLK ratio in Normal Mode and Page Mode examples. • Corrected C_{access} formula for Page Mode configuration time calculation. • Corrected delay value for `flash_data`.

continued...

Date	Version	Changes
June 2015	2015.06.15	Added support for Spansion S25FS256S and S25FS512S.
January 2015	2015.01.23	• Corrected DATA width in PFL IP core With Dual P30 or P33 CFI Flash Memory Devices figure. • Corrected Spansion part number for S29JL032H and S29JL032H from 229JL032H and 229JL032H respectively. • Added Micron MT28GU512AAA1EGC-0SIT and MT28GU01GAAA1EGC-0SIT • Added example of programming PFL using command line. • Rearranged the supported flash memory device by grouping device families. • Added third-party programmer support.
June 2014	2014.06.30	Replaced MegaWizard Plug-In Manager information with IP Catalog.
May 2014	3.2	Updated Table 16 on page 41 to remove Stratix V limitation for the Enhanced bitstream decompression IP core option.
May 2013	3.1	Updated Table 2 on page 5 to add 28F00BP30 and 28F00BP33.
September 2012	3.0	• Updated manufacturer name from Numonyx to Micron. • Updated "Implementing Remote System Upgrade with the PFL IP Core" on page 22, and "Specifications" on page 49 • Updated Table 4 on page 7, Table 10 on page 30, Table 16 on page 41, Table 17 on page 45, and Table 18 on page 50. • Removed figures.
August 2012	2.1	Updated Table 1.
December 2011	2.0	• Updated "Using Enhanced Bitstream Compression and Decompression" to include reference. • Updated Table 1 to include Eon Silicon CFI device EN29GL128 and to remove S29GL-N devices. • Updated Table 2 to include Micron 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.