

This document describes the latest enhanced configuration (EPC) device flash memory standard with a feature-rich configuration controller. A single-chip configuration solution provides you with several new and advanced features that significantly reduce configuration times. This document discusses the hardware and software implementation of the EPC device features such as concurrent and dynamic configuration, data compression, clock division, and an external flash memory interface. EPC devices include EPC4, EPC8, and EPC16 devices.

Concurrent Configuration

Configuration data is transmitted from the EPC device to the SRAM-based device on the DATA lines. The DATA lines are outputs on the EPC devices and inputs to the SRAM-based devices.

These DATA lines correspond to the Bit*n* lines in the **Convert Programming Files** window in the Altera® Quartus® II software. For example, if you specify a SRAM Object File (.sof) to use Bit0 in the Quartus II software, that .sof is transmitted on the DATA[0] line from the EPC device to the SRAM-based device.


Supported Schemes and Guidelines


There are several different ways to configure Altera SRAM-based programmable logic devices (PLDs) with EPC devices:

- 1-bit passive serial (PS)
- 2-bit PS
- 4-bit PS
- 8-bit PS
- Fast passive parallel (FPP)

Additionally, you can use these configuration schemes in conjunction with the dynamic configuration option (previously called page mode operation) for sophisticated configuration setups.

FPP configuration mode uses the eight DATA[7..0] lines from the EPC device, which is used to configure APEX™ II and Stratix® series devices. To decrease configuration time, FPP configuration provides eight bits of configuration data per clock cycle to the target device.

 For more information about configuration schemes, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet* and *Configuring Stratix & Stratix GX Devices* chapter in the *Stratix Handbook*.

 For more information about additional EPC devices, refer to the [PCN0506: Addition of Intel Flash Memory As Source For EPC4, EPC8, and EPC16 Enhanced Configuration Devices](#) and the [Using the Intel Flash Memory Based EPC4, EPC8, and EPC16 Devices](#) white paper.

Concurrent Configuration Using n-Bit PS Modes

The n -bit ($n = 1, 2, 4, \text{ or } 8$) PS configuration mode allows EPC devices to concurrently configure SRAM-based devices or device chains. In addition, these devices do not have to be the same device family or density and they can be any combination of Altera SRAM-based devices. An individual EPC device DATA line is available for each targeted device. Each DATA line can also feed a daisy chain of devices.

The Quartus II software only allows the selection of n -bit PS configuration modes. However, you can use these modes to configure any number of devices from 1 to 8. When configuring SRAM-based devices using n -bit PS modes, refer to [Table 1](#) to select the appropriate configuration mode for the fastest configuration time.


 Mode selection has an impact on the amount of memory used. For more information, refer to [“Calculating the Size of Configuration Space”](#) on page 14.

Table 1. Recommended Configuration Using n -Bit PS Modes

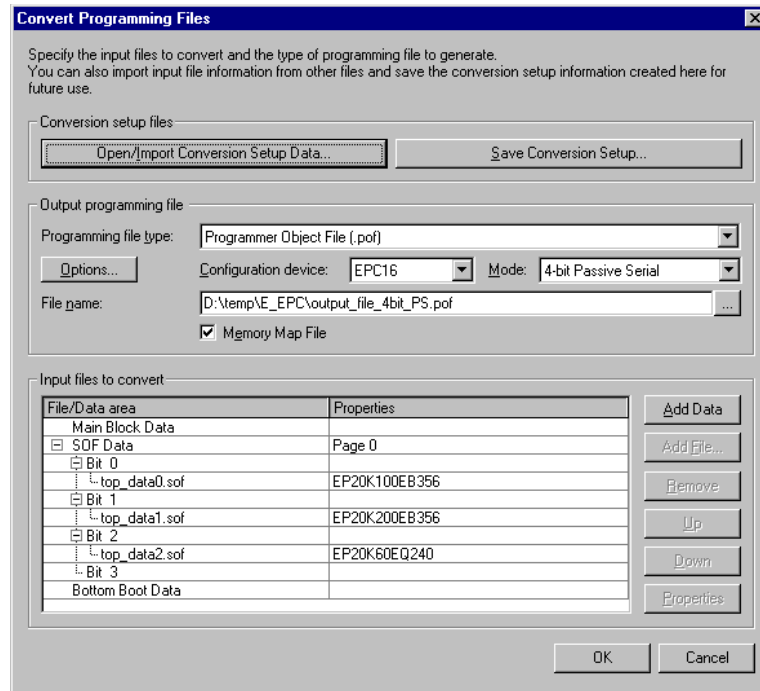
Number of Devices ⁽¹⁾	Recommended Configuration Mode
1	1-bit PS
2	2-bit PS
3	4-bit PS
4	4-bit PS
5	8-bit PS
6	8-bit PS
7	8-bit PS
8	8-bit PS

Note to Table 1:

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

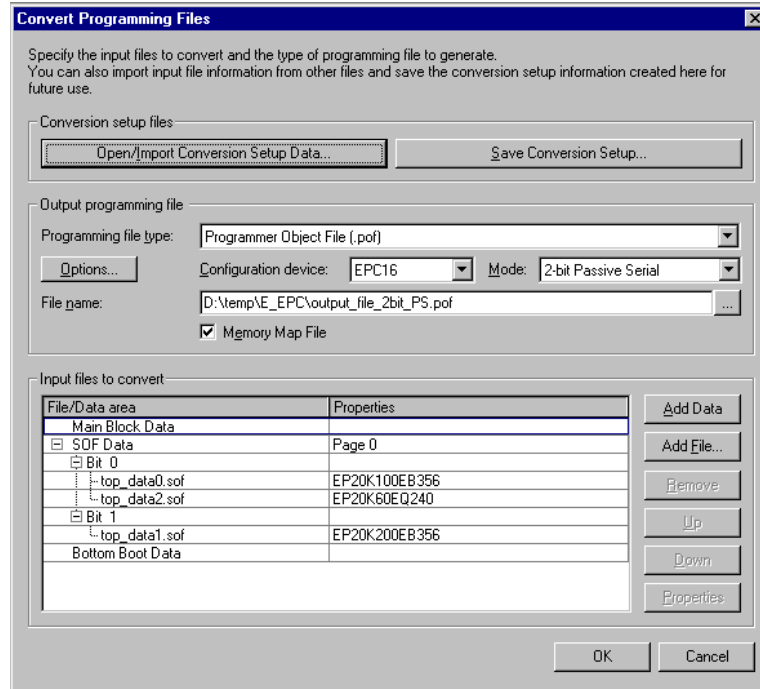
For example, if you configure three SRAM-based devices, you would use the 4-bit PS mode. For the DATA0, DATA1, and DATA2 lines, the corresponding .sof data will be transmitted from the configuration device to the SRAM-based PLD. For the DATA3 line, you can leave the corresponding Bit3 line blank in the Quartus II software. On the PCB, leave the DATA3 line from the EPC device unconnected. Figure 1 shows the Quartus II Convert Programming Files window (Tools menu) setup for this scheme.

Figure 1. Software Settings for Configuring Devices Using *n*-Bit PS Modes



Alternatively, you can daisy chain two SRAM-based devices to one DATA line while other DATA lines drive one device each. For example, you could use the 2-bit PS mode to drive two SRAM-based devices with DATA Bit0 (EP20K100E and EP20K60E devices) and the third device (the EP20K200E device) with DATA Bit1. This 2-bit PS configuration scheme requires less space in the configuration flash memory, but may increase the total system configuration time as shown in Figure 2.

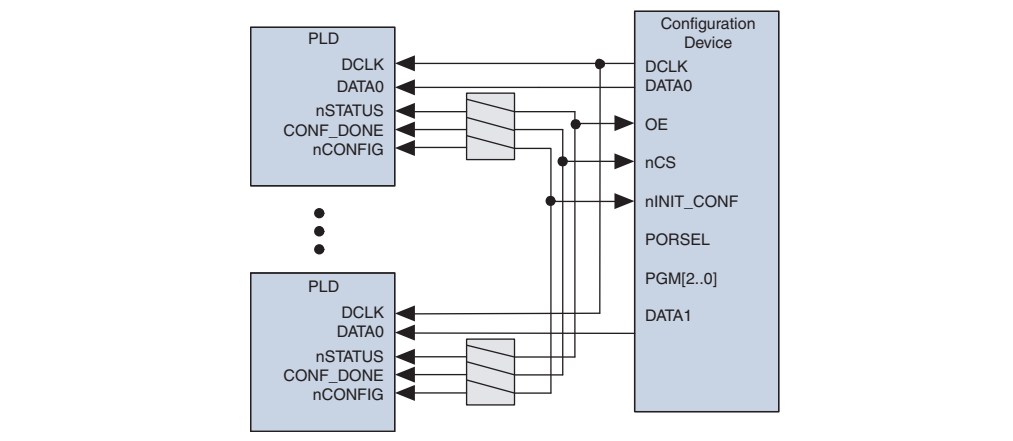
Figure 2. Daisy Chaining Two SRAM-Based Devices to One DATA Line



Design Guidelines

For debugging, Altera recommends keeping the control lines such as $nSTATUS$, $nCONFIG$, and $CONF_DONE$ between each PLD and the configuration device separate. You can keep control lines separate with a switch to manage which control signals are fed back into the EPC device. Figure 3 shows an example of the connections between the EPC device and the targeted PLDs.

Figure 3. Example of Using Debugging Switches for Control Lines



Dynamic Configuration (Page Mode) Implementation Overview

Pages in EPC devices allow you to organize and store various configurations for the entire system that use one or more Altera PLDs. This dynamic configuration (or page mode) feature allows systems to dynamically reconfigure their PLDs with different configuration files.

You can use different pages to store configuration files that support different standards (for example, I/O standards or memory). Alternatively, the different pages place the system in different modes. For example, page 0 could contain a configuration `.sof` for the PLD that only processes data packets and page 1 could contain a configuration file for the same PLD that processes data and voice packets.

With the ability to dynamically switch pages, you can also configure Altera devices with various revisions for debugging without having to reprogram the configuration device. For example, you can configure a device that is on “stand-by” to perform another function and then reconfigure it back with the original configuration file.

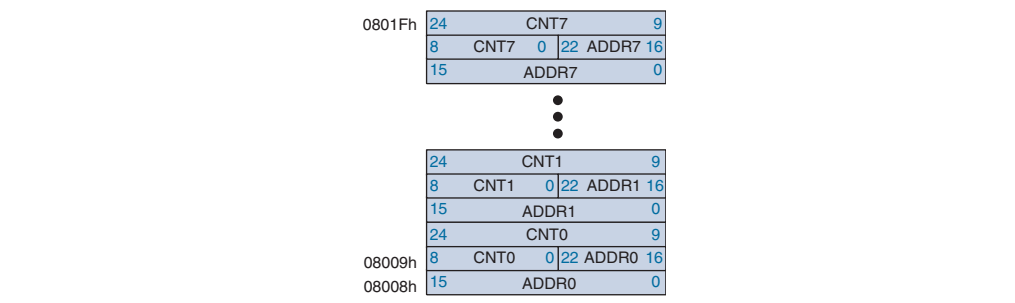
A page is a section of the flash memory space that contains configuration data for all PLDs in the system. One page stores one system configuration regardless of the number of PLDs in the system. The size of each page is dynamic and changes each time the EPC device is reprogrammed. EPC devices support a maximum of eight pages of configuration data. The number of pages is also limited to the density of the configuration device.



The number of pages required in a system is not dependent on the number of PLDs in the system, but depends on the number of unique system configurations.

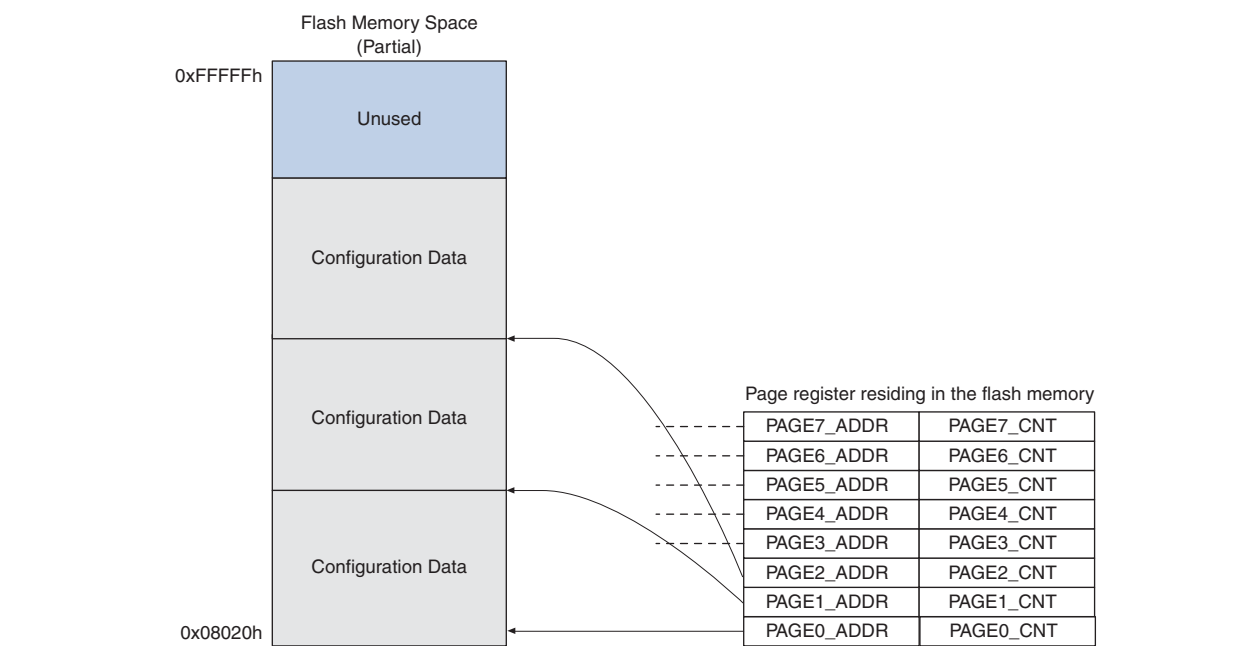
External page mode input pins PGM[2..0] determine which page to use during PLD configuration and page pointers determine the data location. Each page pointer consists of a starting address register and a length count register. The word-addressable starting address register (23 bits) is used to determine where the page begins in the flash memory. The count register (25 bits) determines the length of the page counted in nibbles (group of 4 bits equaling half of a byte). Figure 4 shows a block diagram of the option-bit space and its address locations.

Figure 4. Option-Bit Memory Map




For example, a page for the EPC16 device must start between word addresses 0x08020h and 0xFFFFFh and cannot overlap with other pages. Figure 5 shows an example of an EPC16 page mode using three pages.

Figure 5. EPC16 Page Mode Implementation Example



During configuration, different pages are selected by the PGM[2..0] pins. These pins are used to select one out of eight pages. PGM[2..0] pins are sampled at least one time before the configuration data is sent to the targeted PLDs.


Setting the PGM[2..0] pins to select an incorrect page (for example, a page that is non-existent or a blank page) causes the EPC device to enter an erroneous state. The only way to recover from this state is to set the PGM[2..0] pins to select a valid page and power cycle the board.

 To ensure proper configuration, only set the PGM[2..0] pins to select valid pages.

Within each page, you can store as many configuration files as your system needs. There is no limitation to the length of a page except for the physical limitation determined by the size of the flash memory (for example, 0xFFFFFFFFh for EPC16 devices). However, all pages must be contiguous.

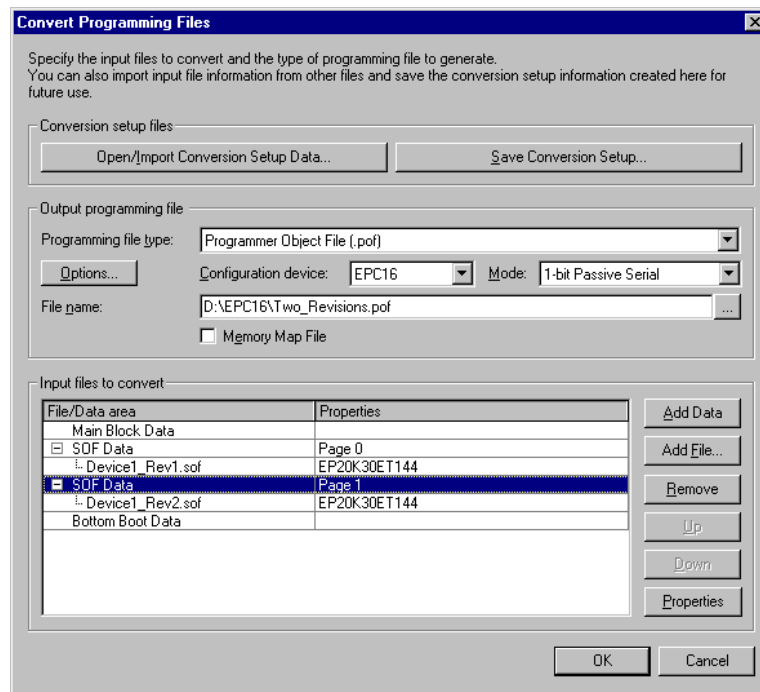
Software Implementation (Convert Programming Files)

The **Convert Programming Files** window (Tools menu) in the Quartus II software allows you to create the EPC device Programmer Object File (.pof) and enable the dynamic configuration feature.

 EPC devices do not support passive parallel asynchronous (PPA) and passive parallel synchronous (PPS) configuration modes. If you choose one of these modes, the Quartus II software reports an error message when the .pof is generated.

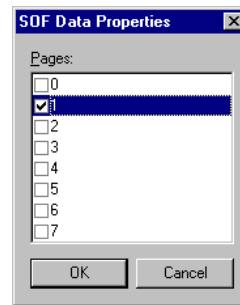
In the **Convert Programming Files** window, there are **SOF Data** entries (.sof), located in the **Input files to convert** dialog box. Each **SOF Data** entry refers to a unique system configuration. [Figure 6](#) shows the setup for a system that has one APEX device and uses page 0 and page 1. Each page has a different version of the configuration file for the same APEX device.

Figure 6. Using Page Mode Example



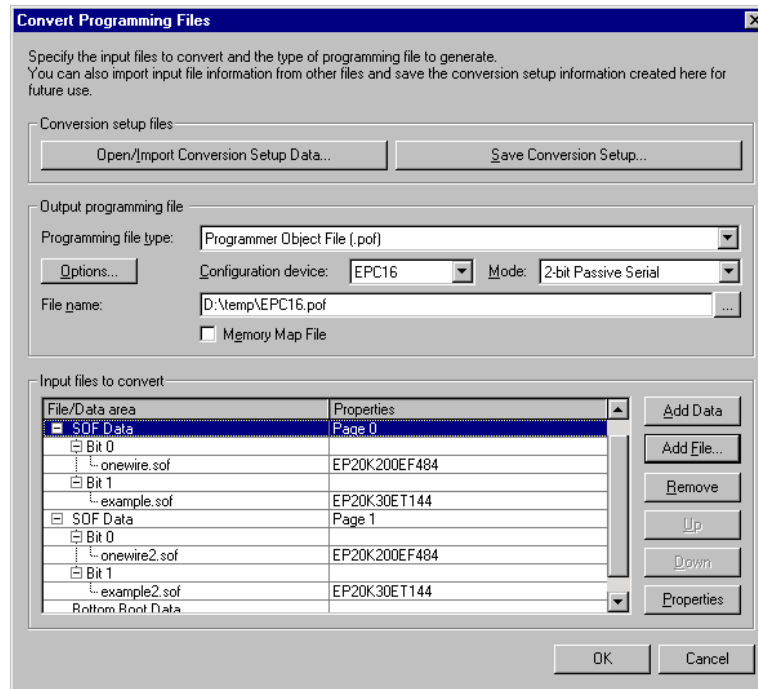
To set the page pointer to point to a particular page or **SOF Data** entry, select **SOF Data** and click **Properties**. Clicking **Properties** launches the **SOF Data Properties** window where you can select page pointers to point to the chosen **SOF Data**. If you do not use the **SOF Data Properties** window to make changes, the default page is 0. Each **SOF Data** entry for your configuration device must have a unique page number. [Figure 7](#) shows page pointer 1 assigned to the **SOF Data** section containing **Device1_Rev2.sof** (refer to [Figure 6](#)).

Figure 7. Software Setting for Selecting Pages



[Figure 8](#) shows a more complex setup that uses the 2-bit PS configuration mode to concurrently configure two different APEX devices with multiple pages storing two revisions of each design. Two configurations for the entire system requires four configuration files (the number of devices multiplied by the number of unique system configurations).

Figure 8. Concurrent Configuration of Two Devices with Two System Configurations



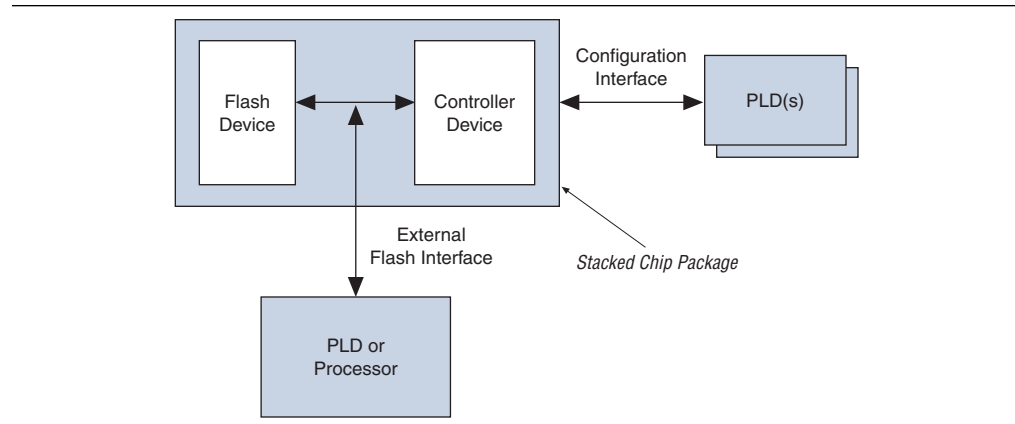
By selecting the **Memory Map File** option, the Quartus II Memory Map File (**.map**) is generated, describing the flash memory address locations. This information is useful when you are using the external flash interface feature.

External Flash Memory Interface

EPC devices support an external flash interface that allows devices external to the controller access to the EPC device's flash memory. You can use the flash memory to store boot or application code for processors or as general-purpose memory for processors and PLDs.

Figure 9 shows the interfaces available on the EPC device.

Figure 9. EPC Device Interfaces



Applications that require remote update capabilities for on-board programmable logic (Stratix series devices) and applications that use soft-embedded processor cores (Nios® embedded processor) use the external flash memory interface feature.

For soft-embedded processor core applications, the controller configures the programmable logic with the configuration data stored in the flash memory. On successful configuration, the embedded processor uses the external flash interface to boot up and run code from the same flash memory, eliminating the need for a stand-alone flash memory device.

For applications requiring remote system configuration capabilities, a processor or PLD can use the external flash interface to store an updated configuration image into a new page in flash memory—the external flash interface coupled with dynamic configuration. You can obtain new configuration data from a local intelligent host or through the Internet. Reconfiguring the system with the new page updates the system configuration.

- For more information about implementing remote and local system updates with EPC devices, refer to the *Using Remote System Configuration with Stratix & Stratix GX Devices* chapter in the *Stratix Handbook* or the *Remote System Upgrades with Stratix II & Stratix II GX Devices* chapter in the *Stratix II Handbook*.
- Currently, only EPC4 and EPC16 configuration devices support the external flash interface. For support of this feature in other EPC devices, refer to the [Altera Applications 24/7 Technical Support](#) page.

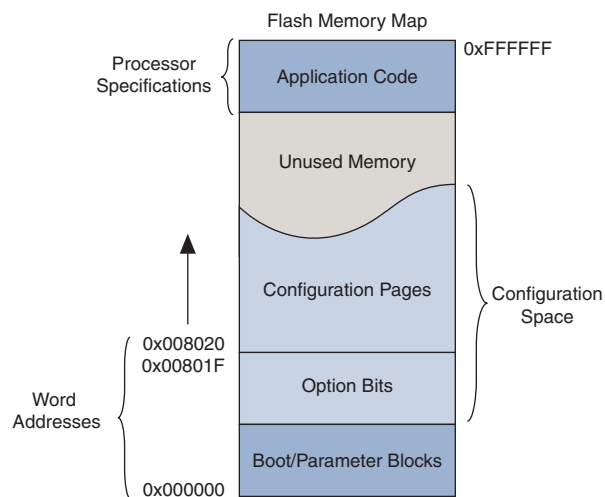
Flash Memory Map

You can divide an EPC device's flash memory into two categories—logical with configuration and processor space and physical with flash data block boundaries. Configuration space consists of portions of memory used to store configuration option bits and configuration data. Processor space consists of portions of memory used to store boot and application code.

Logical Divisions

In all EPC devices, configuration option bits are stored ranging from word address 0x008000 to 0x00801F (byte address 0x010000 to 0x01003F). These bits are used to enable various controller features such as configuration mode selection, compression mode selection, and clock divider selection. In all EPC devices, configuration data is stored starting from word address location 0x008020 or byte address 0x010040. The ending address of configuration space is not fixed and depends on the number and density of PLDs configured using the EPC device as well as the number of pages. All remaining address locations above the configuration space are available for processor application code. The boot space spans addresses 0x000000 to 0x007FFF. Both boot and application code spaces are intended for use by an external processor or PLD. [Figure 10](#) shows the flash memory map inside an EPC16 device.

Figure 10. EPC16 Flash Memory Map



Physical Divisions


Physical divisions are flash data blocks that can be individually written to and erased. For example, the Sharp flash-based EPC16 device contains 16-Mb Sharp flash memory that is divided into 2 boot blocks, 6 parameter blocks, and 31 main data blocks. These physical divisions vary from one flash memory or vendor to another and must be considered if the external flash interface is used to erase or write flash memory. These divisions are not significant if the interface is used as a read-only interface after initial programming.

Interface Availability and Connections

Flash memory ports are shared between the internal controller and the external device. A processor or PLD uses the external flash interface to access flash memory only when the controller is not using the interface. Therefore, the internal controller is the primary master of the bus while the external device is the secondary master.

Flash memory ports such as address, data, and control are internally connected to the controller device. Additionally, these ports are connected to pins on the package providing the external interface. During in-system programming (ISP) of the EPC device as well as configuration of the PLDs, the controller uses the internal interface to flash memory, rendering the external interface unavailable. External devices should tri-state all connections for the entire duration of the ISP and configuration to prevent contention.

On completion of the ISP and configuration, the internal controller tri-states its interface to the flash memory and enables weak internal pull-up resistors on address and control lines as well as bus-hold circuits on the data lines. The internal flash interface is now disabled and the external flash interface is available.

 If you do not use the external flash interface feature, most flash-related pins must be left unconnected on the board to avoid contention. There are a few exceptions to this guideline which are outlined in the datasheet and pin-out tables.

 For more information about detailed schematics, refer to the *Enhanced Configuration Devices (EPC4, EPC8, and EPC16) Data Sheet*.


Quartus II Software Support

You can use the **Convert Programming Files** window to generate flash memory programming files. You can program flash memory in-system using JTAG or the external flash interface. Select the **.pof** when programming the flash memory in-system. You can also convert this **.pof** to a Jam™ Standard Test and Programming Language (STAPL) Format File (**.jam**) or JAM Byte Code File (**.jbc**) for ISP. When programming the flash memory through the external flash interface, you can create a Hexadecimal (Intel-Format) Output File (**.hexout**) from this window.

 The **.hexout** file used for programming EPC devices is different from the **.hexout** configuration file generated for SRAM PLDs.

Along with PLD configuration files, you can program processor boot and application code into flash memory through the **Convert Programming Files** window. You can add a Hexadecimal (Intel-Format) File (**.hex**) containing boot code to the **Bottom Boot Data** section of the window. Similarly, you can add a **.hex** file containing application code to the **Main Block Data** section. You can store these files in the flash memory using relative or absolute addressing. To select the type of addressing, highlight the **Bottom Boot Data** or **Main Block Data** section and click **Properties (Convert Programming Files** window).

Relative addressing mode allows the Quartus II software to pick the location of the file in memory. For example, the Quartus II software always stores boot code starting at address location 0x000000. This data increases to higher addresses.

 The maximum boot file size for the EPC16 configuration device is 32K words or 64 Kbytes. The boot code is limited to the boot and flash memory parameter blocks.

When you select relative addressing mode for **Main Block Data**, the Quartus II software aligns the last byte of information with the highest address (for example, 0x1FFFFFF). Therefore, the starting address is dependent on the size of the **.hex** file. You can obtain the starting address of the application code with the **.map** file.

The absolute addressing mode forces the Quartus II software to store the boot or application **.hex** file data in address locations specified inside the **.hex** file itself. When this mode is selected, create **.hex** files with the correct offsets and ensure there is no overlap with addresses used for storing configuration data.

Figure 11 shows a screenshot of the **Convert Programming Files** window setup to create a **.pof** and **.map** file for an EPC device.


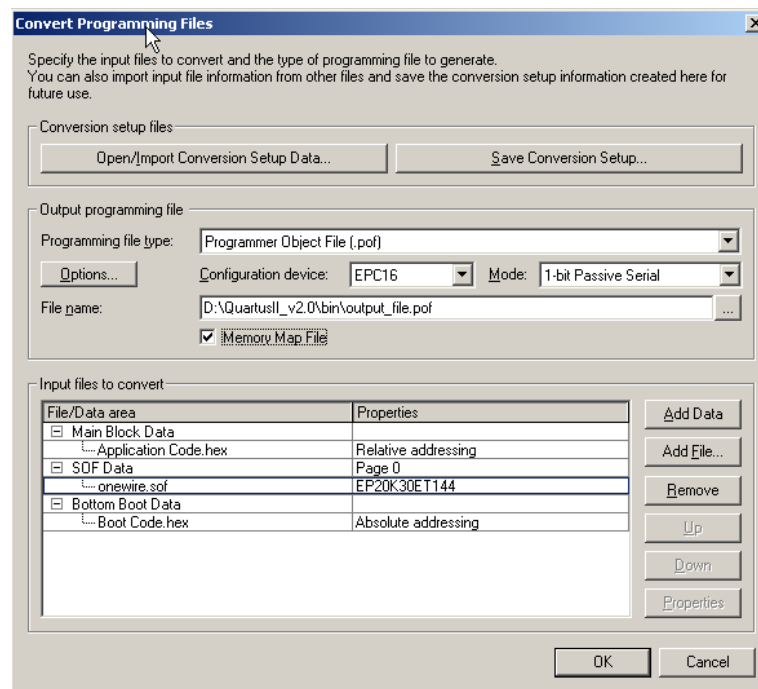
 Only one **.hex** file can be added to the **Bottom Boot Data** and **Main Block Data** sections of this window.

Figure 11. Storing Boot and Application Code in Flash Memory



You can use the **Quartus II Convert Programming Files** window to create two files specific to the external flash interface feature—the **.hexout** and the **.map** files. The **.hexout** file contains an image of the flash memory and the **.map** file contains memory map information. The **.hexout** file is used by an external processor or PLD to program the flash memory using the external flash interface. The **.map** file contains starting and ending addresses for boot code, configuration page data, and application code.

You can use the `.hexout` file to program blank EPC devices, update portions of the flash memory (for example, a new configuration page), or both. This file contains 16 Mbits or 2 Mbytes of data. [Table 2](#) lists the format of the `.map` file.

Table 2. Memory Map File ⁽¹⁾

Block	Start Address	End Address
BOTTOM BOOT	0x00000000	0x0000001F
OPTION BITS	0x00010000	0x0001003F
PAGE 0	0x00010040	0x0001AD7F
MAIN	0x001FFFE0	0x001FFFFFFF

Note to Table 2:

(1) All the addresses in this file are byte addresses.

To perform partial flash memory updates, select the relevant portions of the `.hexout` file using memory map information provided in the `.map` file.



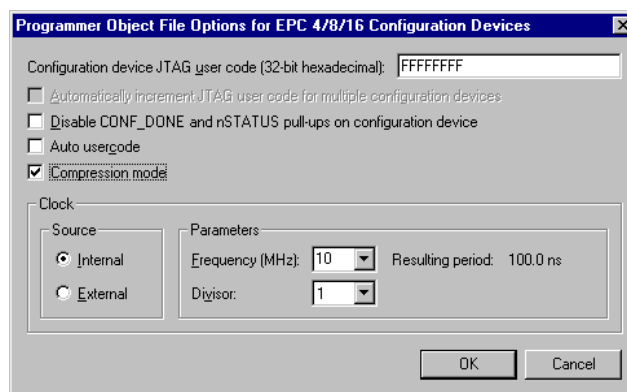
Configuration data and processor space data could exist within the same physical data block. In such cases, erasing the physical data block would affect both configuration and processor data, requiring you to update both. You can avoid this situation by storing application data starting from the next available whole data block.

Data Compression

EPC devices support an efficient compression algorithm that compresses configuration data by 1.9x for typical designs, effectively doubling the size of the device. To select the right density for EPC devices, you should pre-calculate the total size of uncompressed configuration space.

By clicking **Options** in the Convert Programming Files window, you can turn on the **Compression mode** option in the Programming Object File Options window with the `.pof` file selected as the programming file type. [Figure 12](#) shows how to select the compression mode.

Figure 12. Selecting Compression Mode



Calculating the Size of Configuration Space

When using 1-bit PS configuration mode to serially configure multiple devices, all configuration data is transmitted through the same DATA line and the devices are daisy-chained together. Therefore, the total size of the uncompressed configuration data is equal to the sum of the SRAM-based device's configuration file size multiplied by the number of pages used.

When using n -bit PS configuration mode to concurrently configure multiple devices, each SRAM-based device has its own DATA line from the EPC devices. The total size of the uncompressed configuration space is equal to the size of the largest device's configuration file size multiplied by n (where $n = 1, 2, 4, \text{ or } 8$), which is then multiplied by the number of pages used. For example, if three devices are concurrently configured using 4-bit PS configuration mode, the total size of the uncompressed configuration space is equal to the size of the largest device's configuration file multiplied by four.

When using FPP configuration mode, the total size of the uncompressed configuration space is equal to the sum of the SRAM-based device's configuration file size multiplied by the number of pages used.

Clock Divider

The clock divider value specifies the clock frequency divisor, which is used to determine the DCLK frequency, or how fast the data is clocked into the SRAM-based device. You must consider the maximum DCLK input frequency of the targeted SRAM device family while selecting the clock input and divider settings. For DCLK timing specifications of SRAM-based devices, refer to the *Configuration Handbook*.

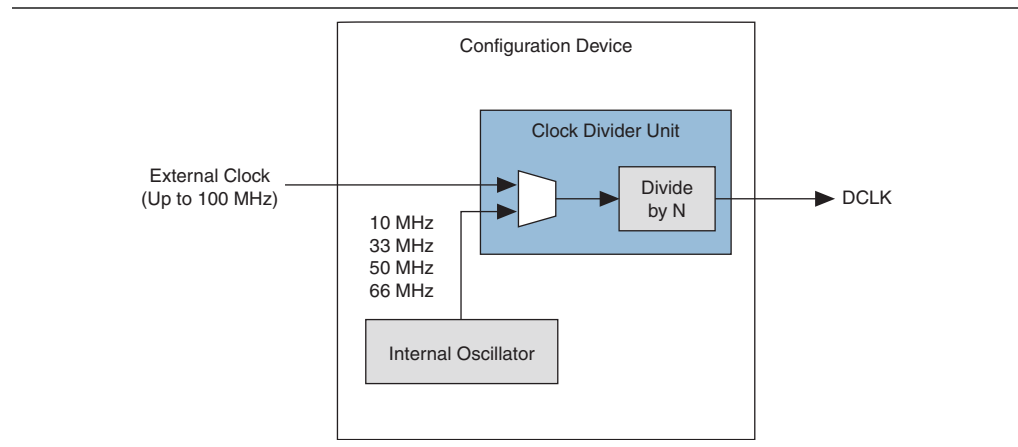
Settings and Guidelines

EPC devices use either an internal oscillator or an external clock source to clock data into SRAM-based devices. The EPC device's internal oscillator runs at nominal speeds of 10, 33, 50, or 66 MHz. Additionally, the EPC device accepts an external clock source running at speeds of up to 100 MHz.

For more information about the minimum and maximum speeds of the internal oscillator, refer to the *Enhanced Configuration Device Data Sheet*.

Figure 13 shows the clock divider unit in EPC devices.

Figure 13. Clock Divider Unit in EPC Devices

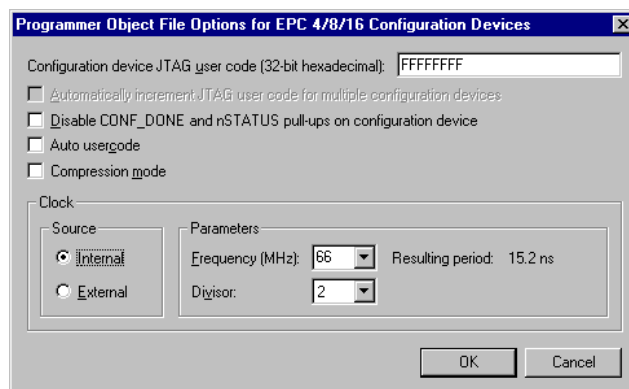


Software Implementations

You can select the clock source and the clock speed in the **Programming Object File Options** window with the **.pof** file selected as the programming file type in the Convert Programming Files window. You can type the appropriate external clock frequency in the **Frequency (MHz)** drop-down menu and select any value from the divisor list regardless of the clock source setting.

Figure 14 shows how to set the clock source and clock divisor.

Figure 14. Software for Setting Clock Source and Clock Divisor



Document Revision History

Table 3 lists the revision history for this document.

Table 3. Document Revision History

Date	Version	Changes
January 2012	3.0	Minor text edits.
December 2009	2.5	Removed “Referenced Documents” section.
October 2008	2.4	<ul style="list-style-type: none"> ■ Added “Referenced Documents” section. ■ Updated new document format.
April 2007	2.3	Added document revision history.
October 2005	2.2	Technical content added.
July 2004	2.0	<ul style="list-style-type: none"> ■ Added text regarding pointing to an incorrect page after Figure 2–8. ■ Renamed .hexpof to .hexout throughout chapter.
September 2003	1.0	Initial Release.