


## Core Overview


The EPCS device controller core with Avalon® interface allows Nios® II systems to access an Altera® EPCS serial configuration device. Altera provides drivers that integrate into the Nios II hardware abstraction layer (HAL) system library, allowing you to read and write the EPCS device using the familiar HAL application program interface (API) for flash devices.

Using the EPCS device controller core, Nios II systems can:

- Store program code in the EPCS device. The EPCS device controller core provides a boot-loader feature that allows Nios II systems to store the main program code in an EPCS device.
- Store non-volatile program data, such as a serial number, a NIC number, and other persistent data.
- Manage the device configuration data. For example, a network-enabled embedded system can receive new FPGA configuration data over a network, and use the core to program the new data into an EPCS serial configuration device.

The EPCS device controller core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system. The flash programmer utility in the Nios II IDE allows you to manage and program data contents into the EPCS device.

 For information about the EPCS serial configuration device family, refer to the *Serial Configuration Devices (EPCS1, EPCS4, EPCS16, EPCS64 and EPCS128) Data Sheet*. For details about using the Nios II HAL API to read and write flash memory, refer to the *Nios II Software Developer's Handbook*. For details about managing and programming the EPCS memory contents, refer to the *Nios II Flash Programmer User Guide*.

 For Nios II processor users, the EPCS device controller core supersedes the Active Serial Memory Interface (ASMI) device. New designs should use the EPCS device controller core instead of the ASMI core.

This chapter contains the following sections:

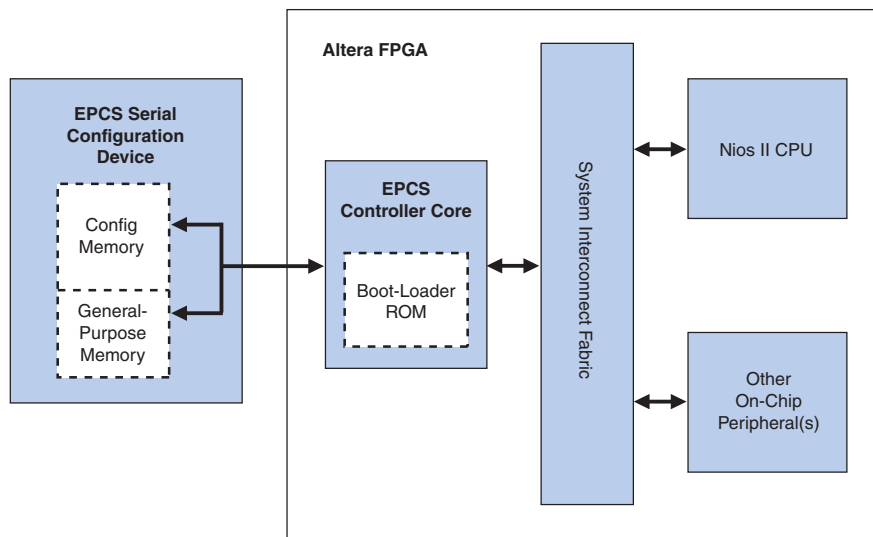
- “Functional Description” on page 4-2
- “Device and Tools Support” on page 4-4
- “Instantiating the Core in SOPC Builder” on page 4-4
- “Software Programming Model” on page 4-4

## Functional Description

Figure 4-1 shows a block diagram of the EPCS device controller core in a typical system configuration. As shown in Figure 4-1, the EPCS device's memory can be thought of as two separate regions:


- **FPGA configuration memory**—FPGA configuration data is stored in this region.
- **General-purpose memory**—If the FPGA configuration data does not fill up the entire EPCS device, any left-over space can be used for general-purpose data and system startup code.

**Figure 4-1.** Nios II System Integrating an EPCS Device Controller Core




By virtue of the HAL generic device model for flash devices, accessing the EPCS device using the HAL API is the same as accessing any flash memory. The EPCS device has a special-purpose hardware interface, so Nios II programs must read and write the EPCS memory using the provided HAL flash drivers.

The EPCS device controller core contains an on-chip memory for storing a boot-loader program. When used in conjunction with Cyclone®, Cyclone II, and Cyclone III devices, the core requires 512 bytes of boot-loader ROM. For Stratix® II and Stratix III devices, the core requires 1 KByte of boot-loader ROM. The Nios II processor can be configured to boot from the EPCS device controller core. To do so, set the Nios II reset address to the base address of the EPCS device controller core. In this case, after reset the CPU first executes code from the boot-loader ROM, which copies data from the EPCS general-purpose memory region into a RAM. Then, program control transfers to the RAM. The Nios II IDE provides facilities to compile a program for storage in the EPCS device, and create a programming file to program into the EPCS device.


 For more information, refer to the *Nios II Flash Programmer User Guide*.

The Altera EPCS configuration device connects to the FPGA through dedicated pins on the FPGA, not through general-purpose I/O pins. In all Altera device families except Cyclone III, the EPCS device controller core does not create any I/O ports on the top-level SOPC Builder system module. If the EPCS device and the FPGA are wired together on a board for configuration using the EPCS device (in other words, active serial configuration mode), no further connection is necessary between the EPCS device controller core and the EPCS device. When you compile the SOPC Builder system in the Quartus II software, the EPCS device controller core signals are routed automatically to the device pins for the EPCS device.

 If you program the EPCS device using the Quartus® II Programmer, all previous content is erased. To program the EPCS device with a combination of FPGA configuration data and Nios II program data, use the Nios II IDE flash programmer utility.

You have the flexibility to connect the output pins of Cyclone III devices, which are exported to the top-level design, to any EPCS devices. Perform the following tasks in the Quartus® II software to make the necessary pin assignments:

- On the **Dual-purpose pins** page (**Assignments > Devices > Device and Pin Options**), ensure that the following pins are assigned to the respective values:
  - Data [0] = **Use as regular I/O**
  - Data [1] = **Use as regular I/O**
  - DCLK = **Use as regular I/O**
  - FLASH\_nCE/nCS0 = **Use as regular I/O**
- Using the Pin Planner (**Assignments > Pins**), ensure that the following pins are assigned to the respective configuration functions on the device:
  - data0\_to\_the\_epcs\_controller = DATA0
  - sdo\_from\_the\_epcs\_controller = DATA1,ASDO
  - dclk\_from\_epcs\_controller = DCLK
  - sce\_from\_the\_epcs\_controller = FLASH\_nCE

 For more information about the configuration pins in Cyclone III devices, refer to the [Pin-Out Files for Altera Device](#) page.

## Avalon-MM Slave Interface and Registers

The EPCS device controller core has a single Avalon-MM slave interface that provides access to both boot-loader code and registers that control the core. As shown in [Table 4-1](#), the first segment is dedicated to the boot-loader code, and the next seven words are control and data registers. A Nios II CPU can read the instruction words, starting from the core's base address as flat memory space, which enables the CPU to reset the core's address space.

The EPCS device controller core includes an interrupt signal that can be used to interrupt the CPU when a transfer has completed.

**Table 4-1.** EPCS Device Controller Core Register Map

Offset—Cyclone and Cyclone II (32-bit Word Address)	Offset—Other Device Families (32-bit Word Address)	Register Name	R/W	Bit Description
				31:0
0x00 .. 0x7F	0x00 .. 0xFF	Boot ROM Memory	R	Boot Loader Code
0x080	0x100	Read Data	R	(1)
0x081	0x101	Write Data	W	
0x082	0x102	Status	R/W	
0x083	0x103	Control	R/W	
0x084	0x104	Reserved	—	
0x085	0x105	Slave Enable	R/W	
0x086	0x106	End of Packet	R/W	

**Note to Table 4-1:**

(1) Altera does not publish the usage of the control and data registers. To access the EPCS device, you must use the HAL drivers provided by Altera.

## Device and Tools Support

The EPCS device controller core supports all Altera device families except the Hardcopy® series. The core must be connected to a Nios II processor. The core provides drivers for HAL-based Nios II systems, and the precompiled boot loader code compatible with the Nios II processor.

## Instantiating the Core in SOPC Builder

You can add the EPCS device controller core from the **System Contents** tab in SOPC Builder. There are no user-configurable settings for this component.


 Only one EPCS device controller core can be instantiated in each FPGA design.

## Software Programming Model

This section describes the software programming model for the EPCS device controller core. Altera provides HAL system library drivers that enable you to erase and write the EPCS memory using the HAL API functions. Altera does not publish the usage of the cores registers. Therefore, you must use the HAL drivers provided by Altera to access the EPCS device.

## HAL System Library Support

The Altera-provided driver implements a HAL flash device driver that integrates into the HAL system library for Nios II systems. Programs call the familiar HAL API functions to program the EPCS memory. You do not need to know the details of the underlying drivers to use them.

 The driver for the EPCS device is excluded when the reduced device drivers option is enabled in a BSP or system library. To force inclusion of the EPCS drivers in a BSP with the reduced device drivers option enabled, you can define the preprocessor symbol, `ALT_USE_EPCS_FLASH`, before including the header, as follows:

```
#define ALT_USE_EPCS_FLASH
#include <altera_avalon_epcs_flash_controller.h>
```



The HAL API for programming flash, including C-code examples, is described in detail in the *Nios II Software Developer's Handbook*. For details about managing and programming the EPCS device contents, refer to the *Nios II Flash Programmer User Guide*.

## Software Files

The EPCS device controller core provides the following software files. These files provide low-level access to the hardware and drivers that integrate into the Nios II HAL system library. Application developers should not modify these files.

- **altera\_avalon\_epcs\_flash\_controller.h, altera\_avalon\_epcs\_flash\_controller.c**—Header and source files that define the drivers required for integration into the HAL system library.
- **epcs\_commands.h, epcs\_commands.c**—Header and source files that directly control the EPCS device hardware to read and write the device. These files also rely on the Altera SPI core drivers.

## Referenced Documents

This chapter references the following documents:


- *Nios II Flash Programmer User Guide*
- *Nios II Software Developer's Handbook*
- *Serial Configuration Devices (EPCS1, EPCS4, EPCS16, EPCS64 and EPCS128) Data Sheet*

## Document Revision History

Table 4–2 shows the revision history for this chapter.

**Table 4–2.** Document Revision History

Date and Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	<ul style="list-style-type: none"> <li>■ Revised descriptions of register fields and bits.</li> <li>■ Updated the section on HAL System Library Support.</li> </ul>	—
March 2009 v9.0.0	Updated the boot ROM memory offset for other device families in Table 4–1.	—
November 2008 v8.1.0	Changed to 8-1/2 x 11 page size. No change to content.	—
May 2008 v8.0.0	<ul style="list-style-type: none"> <li>■ Updated the boot rom size.</li> <li>■ Added additional steps to perform to connect output pins in Cyclone III devices.</li> </ul>	Updates made to comply with the Quartus II software version 8.0 release.

 For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).