Nios® II

Nios II Flash Programmer User Guide

ALTERA®

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com

Copyright © 2004 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Printed on recycled paper

UG-NIOSIIFLSHPROG-1.1

NSAI
I.S. EN ISO 9001

Contents

About this Document

This document provides comprehensive information about the Altera® Nios® II flash programmer.

Table 1–1 shows this document's revision history.

Refer to the Nios II embedded processor **readme** file for late-breaking information that is not available in this document.

Table 1–1. Tutorial Revision History

Date	Description
December 2004	Updates for the Nios II version 1.1 release.
May 2004	First release of the flash programmer user guide for the Nios II development boards.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the Find dialog box
- Bookmarks serve as an additional table of contents
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages
- Numerous links, shown in green text, allow you to jump to related information

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	www.altera.com	www.altera.com
Altera literature services	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
FTP site	ftp.altera.com	ftp.altera.com

Note to table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, check box options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”

Visual Cue	Meaning
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

1. Using the Nios II Flash Programmer & Instantiating Flash Memory

Introduction

Many designs that utilize the Nios® II processor also incorporate flash memory on the board as a means to store an FPGA configuration and / or Nios II program data. The Nios II development kit includes a convenient method of programming this flash. Any common flash interface (CFI)-compliant flash device connected to the FPGA can be programmed using the Nios II integrated development environment (IDE) flash programmer.

CFI is a flash interface specification that provides a common interface to flash devices from different vendors. As long as a flash adheres to the specification, it can be queried for it's specific parameters with a special command, and then accessed appropriately. Through this process, the Nios II IDE flash programmer can program any single CFI-compliant flash device, or multiple devices at any offset, and with any type of content.

In addition to CFI flash, the Nios II IDE flash programmer can program any Altera® EPCS Serial Configuration Device connected to the FPGA.

This chapter contains three main sections:

- *Nios II flash programmer overview*: This section describes the flash programmer's design, components, and general options.
- *Using the Nios II flash programmer to program flash*: This section guides you through the process of using Nios II IDE to program flash devices.
- *Instantiating flash in SOPC Builder systems*: This section discusses using SOPC Builder to instantiate flash into a new hardware system so that it may be programmed using the Nios II IDE flash programmer.

If your needs are only to program the flash on an Altera Nios development board using the Nios II flash programmer, refer to "Programming Content into Flash" on page 1–7. The information in this chapter is provided as background to the flash programmer's operation as well as a primer for Chapter 2, Porting the Nios II Flash Programmer to Custom Boards.

Nios II Flash Programmer Overview

This section provides an overview of the Nios II flash programmer, including:

- How the Nios II flash programmer works
- Flash programmer design
- Target boards
- Types of flash content
- Booting options

How the Nios II Flash Programmer Works

The flash programmer employs a two-step process to write data into flash. The first step consists of configuring the FPGA with a specific flash programmer design (see "Flash Programmer Design"). Once this design has been configured in the FPGA, the flash programmer utility (running on the host) collects the appropriate flash content files and sends them to the flash programmer design running on the FPGA. The flash programmer design then programs the content into flash. See Figures 1–1 and 1–2.

Figure 1–1. Step 1: Configuring FPGA with Flash Programmer Design

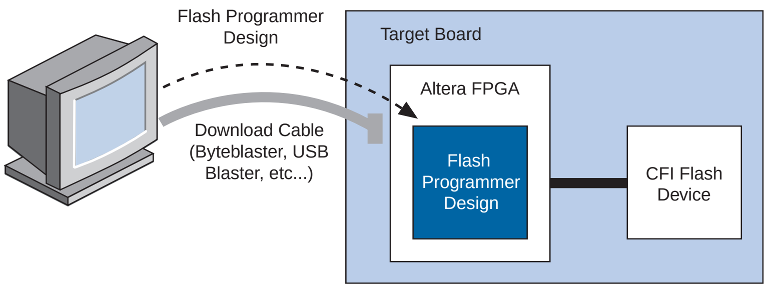


Figure 1–1 shows step 1, the flash programmer utility downloads the flash programmer design to the FPGA.

Figure 1–2. Step 2: Transmitting Flash Content To be Programmed into Flash Device

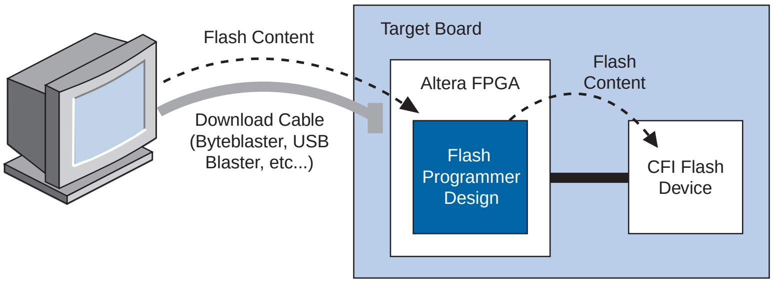


Figure 1–2 shows step 2, the flash programmer utility sends flash content to the flash programmer design running on the FPGA. The flash programmer design then programs the content into the flash device(s). When flash programming is complete, you may freely reconfigure the FPGA with any design.

Flash Programmer Design

A key piece of the Nios II IDE flash programmer is the flash programmer design, which is a small FPGA design containing an SOPC Builder system having the minimal hardware and firmware necessary to:

- Communicate with the host computer over JTAG interface
- Program data provided by the host into flash

Different boards often use different flash devices, different pin-outs, and different Altera FPGA families. For this reason, each flash programmer design is board-specific and therefore cannot be used to program flash on a different board.

If using a custom board, the board designer must create that board's flash programmer design and provide it to anyone who will be programming flash on the board. For more information, refer to Chapter 2, Porting the Nios II Flash Programmer to Custom Boards.

Flash programmer designs for all of the Altera Nios development kit boards are included with the Nios II development kit and can be found in the board's component directory within the Nios II development kit install directory. For example, the flash programmer design for the Stratix® EP1S10 Nios development board is located in the following directory:

<*nios2kitinstall*>/components/altera_nios_dev_board_stratix_1s10/system

Each flash programmer design contains the following components:

- Nios II CPU
- JTAG UART
- Active serial memory interface, which is only required if the FPGA uses an EPCS serial configuration device
- Tri-state bridge
- CFI-compatible flash interface
- System ID peripheral
- On-chip memory for firmware and buffers

For the flash programmer to be used, the actual system being developed does not need to include a Nios II CPU. As long as the flash is connected to the FPGA properly, it can be programmed with the Nios II IDE flash programmer, whether or not a Nios II CPU is used in the design.

Target Boards

A "target board" is a group of files SOPC Builder uses to determine certain characteristics about the board you are using. Some of these characteristics have to do with how flash is connected to the FPGA, making target boards useful to the flash programmer. The following information is contained in a target board:

- Reference designators for each flash chip that is connected to the board's FPGA
- Flash device base addresses in the flash programmer design
- The location of the board's flash programmer design file, or SRAM object file (**.sof**), on the host computer

Reference designators are a way to keep track of individual flash devices between the design(s) being developed and the flash programmer design. Board flash devices may be given different design names and design base addresses, but their reference designators will always be the same. For example, "U5" is the reference designator of the flash device on Altera Nios development boards.

To choose a target board for your SOPC Builder System, open SOPC Builder from Quartus® II (**Tools > SOPC Builder**), then select the board you are using from the **Target** drop down list at the top of the SOPC Builder window.

☞ When using a custom board, your board will not appear in the **Target** drop down list until a target board for the custom board is created. You must create a target board and provide it to anyone who will create SOPC Builder systems for that board. For more information, refer to Chapter 2, Porting the Nios II Flash Programmer to Custom Boards.

Types of Flash Content

There are three types of content you may want to program into flash:

- *Software content*: Software content will normally exist in the form of an **.elf** file generated by Nios II IDE. Programming software into flash allows the Nios II processor to boot from flash upon reset. Because running software from flash can be quite slow, the flash programmer also allows you to place a boot copier in front of your software in flash. The boot copier, when run on the target device, copies your software from flash to a RAM in the system; then branches to the RAM location, where it can run much faster.

- *FPGA configuration data*: FPGA configuration data usually will be in the form of a **.sof** file. The Nios II flash programmer allows you to program the FPGA configuration into flash. If using a configuration controller, as Altera Nios development boards do, the FPGA can be configured from flash at power-on reset.

- *Arbitrary content*: Arbitrary content can be any type of binary data that you may wish to program into flash, such as graphics, audio, etc.

Flash Files

Flash files are data files that have been formatted so they can be read by the Nios II IDE flash programmer. Flash files can be identified by their **.flash** file extension, although they are actually implemented as industry standard SREC files. The <*x*>**2flash** utilities (refer to "Using Command-Line Utilities" on page 1–10) can be used to create flash files from other types of data files. However, when using Nios II IDE to program flash, the utilities are run invisibly in the background.

Booting Options

This section discusses the following Nios II flash programmer booting options:

- Booting from CFI flash
- Booting from EPCS serial configuration devices
- Boot copier program

Booting From CFI Flash

The Altera Nios development boards utilize an Altera EPM7128AE CPLD as a configuration controller. At power-on reset, the configuration controller automatically looks for FPGA configuration data in flash, and if the data exists, automatically configures the FPGA from flash.

Once the FPGA is configured, the configuration may contain a Nios II processor whose reset address is within the flash address space. In this case, the Nios II processor will boot from flash as well.

The Nios II flash programmer is capable of programming both FPGA configuration data and software content into flash at any address so the entire system, both hardware and software, can boot from a CFI flash device.

For more information about the configuration controller on Altera Nios development boards, refer to the *Nios Development Board Reference Manuals*, **www.altera.com**.

Booting From EPCS Serial Configuration Devices

If using an Altera EPCS serial configuration device, the system is also capable of booting both hardware and software from the device. The Nios II IDE flash programmer can program both FPGA configuration data and software content into an EPCS serial configuration device. The flash programmer first checks the size of the FPGA configuration data, then appends the software content to the end of it in the EPCS device.

Booting from an EPCS serial configuration device requires that the system being developed include an EPCS serial flash controller component in SOPC Builder. This component contains a small amount of on-chip ROM that is used for booting. If the reset address of the Nios II processor is set to the base address of the EPCS serial flash controller, the small amount of on-chip ROM attached to the controller is initialized during FPGA configuration with a boot copier. At CPU reset, the processor runs the boot-copier from the on-chip ROM, and relocates the software from the EPCS serial configuration device to the address at which the **.elf** file is linked—presumably occupied by a volatile memory such as a RAM, SDRAM, on-chip RAM, etc. The boot copier then branches to the relocated software and begins executing it.

Boot-Copier Program

When programming software content (**.elf** files) into flash, you can automatically insert a small boot-copier program immediately before your software contents in flash to allow the system to boot from flash. The boot copier will relocate your software to the address where it has been linked in the **.elf** file, then branch to it. Normally, the address to where the software is relocated will be occupied by a volatile memory such as a RAM, SDRAM, on-chip RAM, etc.

☞ Nios II IDE can be used to configure where the different sections of your software are linked in the **.elf** file. For more information, refer to *Editing Project Properties* in the *Nios II IDE Software Tutorial*.

The decision of whether or not to insert a boot copier is made by the `elf2flash` utility, based on the switches `--base`, `--end`, `--reset`, and the location to which your software content is linked in the **.elf** file. The `elf2flash` utility will insert a boot copier in front of your software content when—and **only** when—the reset address is within flash and the **.elf** file is linked to a location outside of the flash.

For example, if your flash memory is mapped to (`0x0` – `0x7FFFFF`) in SOPC Builder, and you have linked your software so that it is to be located at `0x800000` where an external RAM is mapped in SOPC Builder, `elf2flash` will insert a boot copier at address `0x0`, followed by your software contents. Upon reset, the boot copier relocates the software contents from flash to the external RAM located at `0x800000` before executing it.

Programming Content into Flash

This section discusses two methods for programming the three types of content into flash, using:

- Nios II IDE
- Command line utilities

Using Nios II IDE

The preferred method for programming flash is from within Nios II IDE. Nios II IDE uses the same command line utilities as discussed in "Using Command-Line Utilities" on page 1–10; however with Nios II IDE, the process is much more automated, and options are presented in an easy-to-use graphical interface.

The following steps outline the procedure for programming flash using Nios II IDE.

1. Start Nios II IDE, either from the **System Generation** tab in SOPC Builder or from the **Start** menu.

2. Build a software project.

For more information on building a software project, refer to the *Nios II IDE Software Tutorial*.

3. Choose **Tools** > **Flash Programmer**.

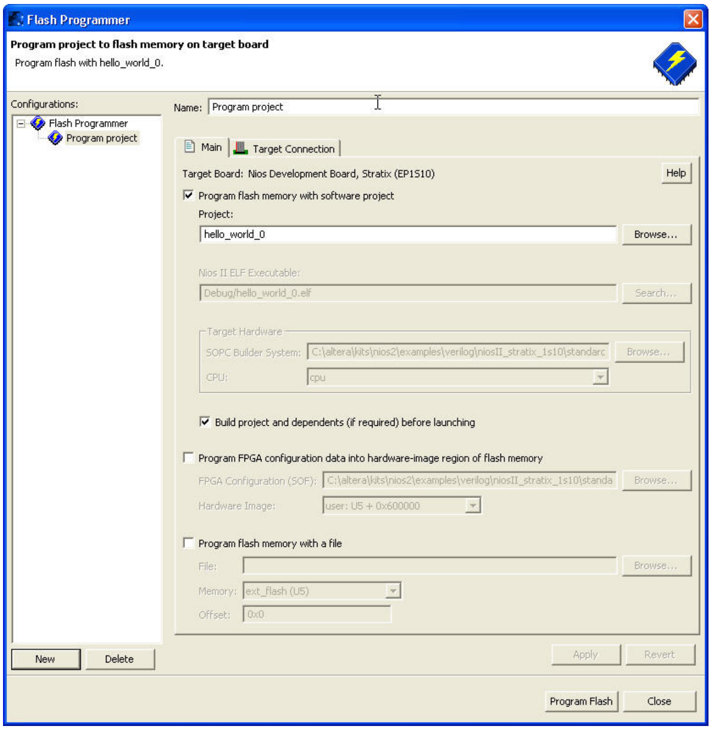


Figure 1–3. Flash Programmer

Flash Programmer

Program project to flash memory on target board

Program flash with hello_world_0.

Configurations:

- Flash Programmer
  - Program project

Name: Program project

Main | Target Connection

Target Board: Nios Development Board, Stratix (EP1S10)

[✓] Program flash memory with software project

Project: hello_world_0

Nios II ELF Executable: Debug/hello_world_0.elf

Target Hardware

SOPC Builder System: C:\altera\kits\nios2\examples\verilog\niosII_stratix_1s10\standarc

CPU: cpu

[✓] Build project and dependents (if required) before launching

[ ] Program FPGA configuration data into hardware-image region of flash memory

FPGA Configuration (SOF): C:\altera\kits\nios2\examples\verilog\niosII_stratix_1s10\standa

Hardware Image: user: U5 + 0x600000

[ ] Program flash memory with a file

File:

Memory: ext_flash (U5)

Offset: 0x0

4. Click **New** (bottom left hand side).

5. Turn on **Program flash memory with software project** to program the project’s **.elf** file into flash .

    a. Verify that the correct software project is listed.

    b. Turn on **Build project and dependents (if required) before launching**.

6. Turn on **Program FPGA configuration data into hardware-image region of flash memory** to program an FPGA configuration file into flash.

   a. Select the FPGA configuration **.sof** file for your hardware.

   b. Select the hardware image offset location for programming the configuration data.

7. Turn on **Program flash memory with a file** to program some other binary file into flash.

   a. Click **Browse...** to select the file you wish to program into flash.

   b. Select the flash memory device you want to program.

   c. Enter the offset location within the flash memory device for programming the file.

8. Click **Apply**. The **Apply** button will be grayed out if none of the settings are changed. This is normal.

9. Click **Program Flash**.

☞ If the Nios II IDE is unable to detect your JTAG cable setting, the **Program Flash** button is grayed out. Click the **Target Connection** tab to setup your JTAG cable.

Nios II IDE will configure the FPGA with the flash programmer design appropriate for the target board, and then program the contents you have specified into the appropriate flash devices on the board.

Flash Programming Script

As well as automatically programming flash, the Nios II IDE also creates a script to program flash from the command line. When you click the **Apply** button in Step 7 of "Using Nios II IDE" on page 1–7, a `program_<project_name>.sh` script file is created in the software project's configuration directory. For instance, if your project is named `hello_world_0` and you built the project using the default **Debug** configuration, the flash programming script will be `hello_world_0\Debug\program_hello_world_0.sh`.

You can run this script from the command line by starting a Nios II software development kit (SDK) shell, changing to the directory containing the script, and then typing `./program_hello_world_0.sh`. Including `./` is often required so the shell interprets the script as executable.

Using Command-Line Utilities

In some cases, you may wish to use the command line instead of the Nios II IDE to program flash. To facilitate this option, the four executables used by the Nios II IDE flash programmer are available from the Nios II SDK shell as command line utilities. The four utilities are located in the <*nios2-kit-install*>**/bin** directory, and this section lists the utilities and their functions.

☞ The utilities must be run from the Nios II SDK shell.

elf2flash Utility

The `elf2flash` utility takes a software file in *elf* format, and translates it to a flash file that can be programmed into flash. This utility can also optionally insert a boot copier into the flash file to copy the software from flash to RAM before running it.

☞ As part of the software build process, this command may have already run. Check the Release directory of your software project to see if a flash file for your software has already been created.

The typical options used with `elf2flash` are listed below. Type `elf2flash --help` from the Nios II SDK shell prompt for a full list of command line options.

`elf2flash` options:

- `--input=<file>`
  Name of input **.elf** file to process
- `--output=<file>`
  Name of output flash file
- `--base=<addr>`
  Base address of the flash in your system (not the flash programmer design)
- `--end=<addr>`
  End address of flash in your system (not the flash programmer design)
- `--reset=<addr>`
  CPU reset address (set in SOPC Builder)
- `--boot=<file>`
  Name of boot copier SREC file

- `--epcs`
  Set this option if the **.elf** content will be programmed in an EPCS serial configuration device.

- `--flash=<designator>`
  Reference designator of the flash device that is to be programmed with the **.elf** content.

sof2flash Utility

The `sof2flash` utility takes an FPGA configuration file in **.sof** format and translates it to a flash file that can be programmed into flash.

The typical options used with `sof2flash` are listed below. Type `sof2flash --help` from the Nios II SDK shell prompt for a full list of command line options.

`sof2flash` options:

- `--input=<file>`
  Name of input .sof file to process

- `--output=<file>`
  Name of output flash file

- `--offset=<addr>`
  Offset within the flash at which **.sof** content is to be programmed

- `--epcs`
  Set this option if the **.sof** content is to be programmed in an EPCS serial configuration device.

- `--flash=<designator>`
  Reference designator of the flash device that is to be programmed with the **.sof** content.

bin2flash Utility

The `bin2flash` utility converts any binary data file to a flash file that can be used by the flash programmer.

The typical options used with `bin2flash` are listed below. Type `bin2flash --help` from the Nios II SDK shell prompt for a full list of command line options.

`bin2flash` options:

- `--input=<file>`
  Name of input file to process

- `--output=<file>`
  Name of output flash file

- `--base=<addr>`
  Base address of the flash in your system

- `--location=<addr>`
  Offset within the flash at which content is to be programmed

- `--epcs`
  Set this option if the content is going to be programmed in an EPCS serial configuration device.

- `--flash=<designator>`
  Reference designator of the flash device that is going to be programmed.

nios2-flash-programmer Utility

The `nios2-flash-programmer` utility takes a flash file, created by one of the conversion utilities, and programs it into the specified flash. The `nios2-flash-programmer` utility is capable of programming any CFI-compatible flash or EPCS serial configuration device in the system.

The typical options used with `nios2-flash-programmer` are listed below. Type `nios2-flash-programmer --help` from the Nios II SDK shell prompt for a full list of command line options.

`nios2-flash-programmer` options:

- `--base=<addr>`
  Base address of the CFI-compatible flash or EPCS serial flash controller component in the board's flash programmer design

- `--epcs`
  Set this option if you are trying to program an EPCS serial configuration device

- `--input=<file>`
  Name of the flash file you wish to program into flash

- `--sof=<file>`
  Name of the flash programmer design's **.sof** file

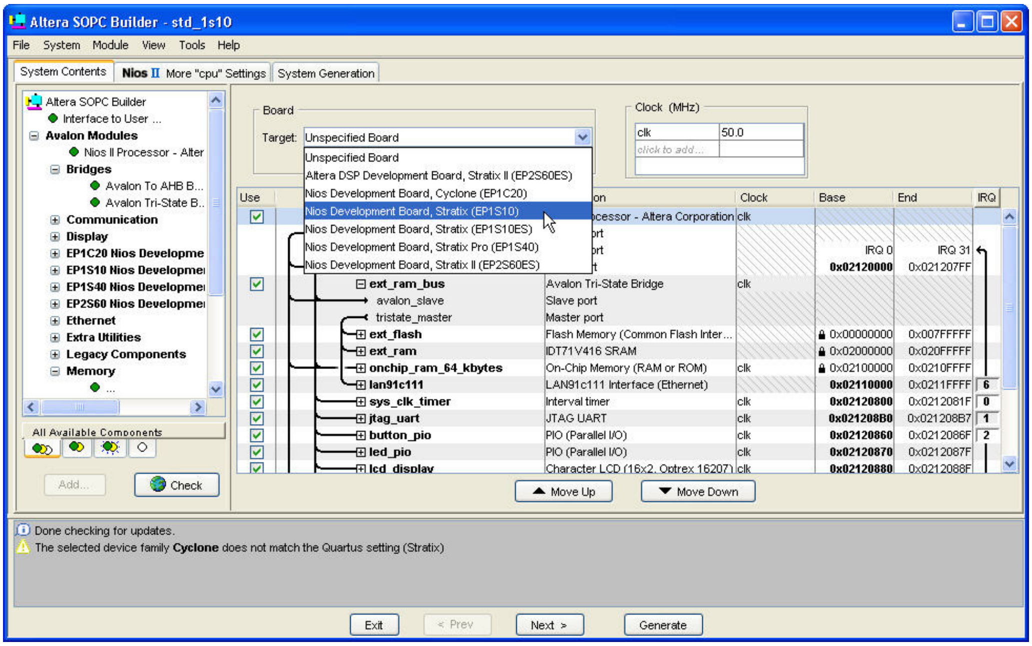
Instantiating Flash in SOPC Builder Systems

To program flash, you must first ensure that your hardware system is generated properly, i.e., so that the flash programmer has all the information it needs.

The first step occurs during the creation of the system in SOPC Builder. An appropriate target board must be selected at the top of the **System Contents** tab in SOPC Builder, see Figure 1–4. The target board contains the following information:

- Number of flash devices connected to the FPGA
- Flash devices' individual board reference designators
- The base addresses for each flash device
- Location of the board's unique FPGA flash programmer design

Figure 1–4. Choosing a Target Board



Board

Target: Unspecified Board

Unspecified Board
Altera DSP Development Board, Stratix II (EP2S60ES)
Nios Development Board, Cyclone (EP1C20)
Nios Development Board, Stratix (EP1S10)
Nios Development Board, Stratix (EP1S10ES)
Nios Development Board, Stratix Pro (EP1S40)
Nios Development Board, Stratix II (EP2S60ES)

Clock (MHz)

clk	50.0
click to add...	

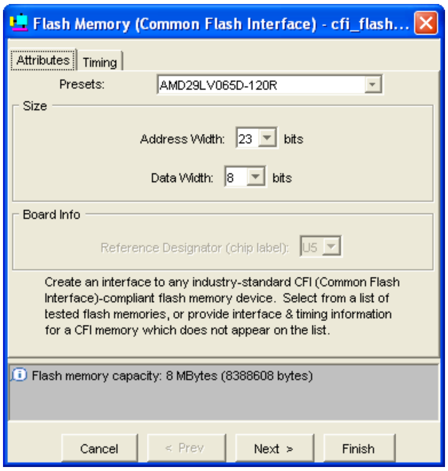
Use		on	Clock	Base	End	IRQ
✓		ocessor - Altera Corporation	clk			
		rt				
		rt		IRQ 0	IRQ 31	
		t		0x02120000	0x021207FF	
✓	ext_ram_bus	Avalon Tri-State Bridge	clk			
	avalon_slave	Slave port				
	tristate_master	Master port				
✓	ext_flash	Flash Memory (Common Flash Inter...		0x00000000	0x007FFFFF	
✓	ext_ram	IDT71V416 SRAM		0x02000000	0x020FFFFF	
✓	onchip_ram_64_kbytes	On-Chip Memory (RAM or ROM)	clk	0x02100000	0x0210FFFF	
✓	lan91c111	LAN91c111 Interface (Ethernet)		0x02110000	0x0211FFFF	6
✓	sys_clk_timer	Interval timer	clk	0x02120800	0x0212081F	0
✓	jtag_uart	JTAG UART	clk	0x021208B0	0x021208B7	1
✓	button_pio	PIO (Parallel I/O)	clk	0x02120860	0x0212086F	2
✓	led_pio	PIO (Parallel I/O)	clk	0x02120870	0x0212087F	
✓	lcd_display	Character LCD (16x2, Optrex 16207)	clk	0x02120880	0x0212088F	

Done checking for updates.
The selected device family **Cyclone** does not match the Quartus setting (Stratix)

If you add CFI flash memory or an EPCS serial flash controller to your SOPC Builder system, you will be presented with component wizards for each. In the component wizards, Figures 1–5 and 1–6, you must select a board reference designator for the flash component. This provides a relationship between the system component and the actual flash device on the board. If the target board you have selected has only one flash device, the flash device's reference designator will be chosen by default in the CFI-compatible **Flash Memory** component wizard, and you will not be able to change it.

☞ Your final design does not need a flash component for the Nios II flash programmer to be able to program the flash. However, the flash programmer design for the board must contain a flash component for each flash device you wish to program using the flash programmer. Using the flash programmer design, the flash programmer can program an attached flash device, even if the SOPC Builder system for the final design does not contain a flash component.

Figure 1–5. CFI-Compatible Flash Memory Wizard


Flash Memory (Common Flash Interface) - cfi_flash...

Attributes | Timing

Presets: AMD29LV065D-120R

Size

Address Width: 23 bits

Data Width: 8 bits

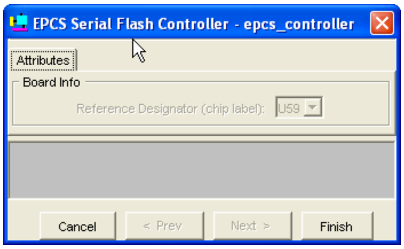
Board Info

Reference Designator (chip label): U5

Create an interface to any industry-standard CFI (Common Flash Interface)-compliant flash memory device. Select from a list of tested flash memories, or provide interface & timing information for a CFI memory which does not appear on the list.

Flash memory capacity: 8 MBytes (8388608 bytes)

Figure 1–6. EPCS Serial Flash Controller Wizard


EPCS Serial Flash Controller - epcs_controller

Attributes

Board Info

Reference Designator (chip label): U59

2. Porting the Nios II Flash Programmer to Custom Boards

Porting the Nios II Flash Programmer to New Hardware

Before the flash programmer can work with a new custom board, you must first port the flash programmer to the new hardware. This chapter guides you through the porting procedure. To avoid making mistakes, it is critical to exactly follow the instructions.

This section includes the following:

- Before you begin
- Creating a target board
- Selecting your new target board

Before You Begin

It is recommended that you first create a preliminary version of the final design intended for the board before porting the flash programmer to the board. This will help ease the porting work as you will have already made the appropriate interface pin assignments for your flash devices. Thus, in most cases, you won't have to re-type the pin assignments for the flash programmer design. Also, the interface to your flash device(s) will have been worked out, reducing the chance of having to re-port the flash programmer again later.

Creating a Target Board

In this document, the term "target board" refers to the set of files used by SOPC Builder to determine board characteristics on which the system is being implemented. SOPC Builder uses target boards much like peripheral components, except that target boards describe the system's connections to the board, whereas peripheral components describe a peripheral's connections to the system. The Nios II Development Kit comes with a pre-built target board for each of the Nios development boards, which you can see in the **Target** drop-down list from the **System Contents** tab (SOPC Builder).

Target boards are important to flash programming because they provide the needed information about the board's flash so that the flash programmer can program the flash. Therefore, the first step in porting the flash programmer to your custom board is to create a target board for it in SOPC Builder. The target board will contain the following elements:

- Reference designators for each flash chip on the board that is connected to the FPGA.

- Base addresses of those flash devices in the flash programmer design.
- A flash programmer design tailored specifically for your board.
- The location of that board's flash programmer design (**.sof** file) on the host computer.

A target board consists of the following files:

- **class.ptf** – Describes the flash that is connected to the FPGA. Also points to the location of the board's flash programmer design.
- <*board_name*>**.ptf** – SOPC Builder system file for the flash programmer design.
- <*board_name*>**.qpf** – Quartus® II project file for the flash programmer design.
- <*board_name*>**.sof** – FPGA configuration file for the flash programmer design. As the first step in programming flash, the flash programmer configures the FPGA with this file.

The `mk_target_board` utility is used to create a generic target board template. Then using SOPC Builder, the target board template will be used as a starting point to build up a fully functional target board that represents your custom board. To create a generic target board template using `mk_target_board`, perform the following steps:

1. Open the Nios® II SDK shell from the **Start** menu. Choose **Programs >Altera > Nios II Development Kit 1.0 > Nios SDK shell**.

2. Using the `cd` command, change to the directory in which your actual Quartus® II design project resides.

3. Run the `mk_target_board` utility to create a generic target board template. The required parameters for `mk_target_board` are as follows:

`mk_target_board` Parameters:

- `--name=<name>`
  The name you would like to call your target board.

- `--family=<family>`
  The name of the FPGA family you are using on your board. Valid entries are "stratix" and "cyclone."

- `--clock=<freq>`
  Clock frequency in MHz that is driving the system clock pin.

☞ It is recommended that you run the flash programmer design at 50 MHz or faster. If it is run at a slower speed, flash programmer performance may suffer. If the only available clock is slower than 50 MHz, you may add a PLL later in the design to the top-level flash programmer design to create a faster clock.

- `--index=<index>`
  The index in the JTAG chain at which your FPGA resides (`1` if it is the only device in the chain)

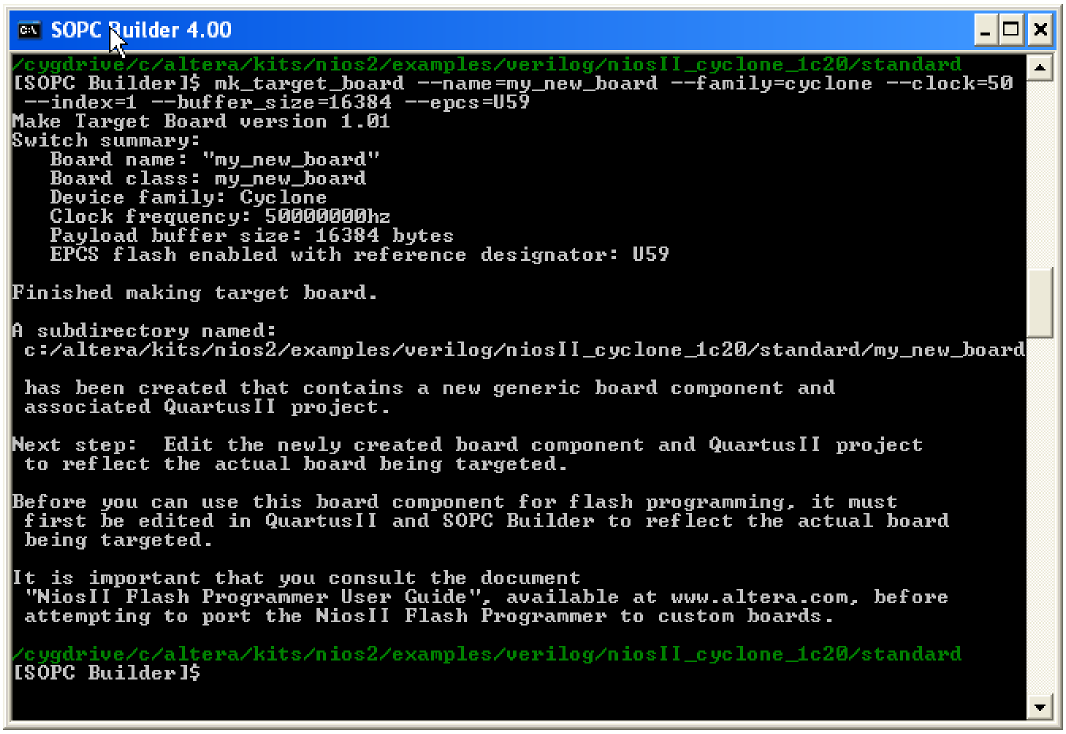
- `--epcs=<refdes>`
  Enter this option only if you are using an EPCS serial configuration device on your board and wish to program it with the flash programmer. Specify the EPCS device's reference designator as the `refdes` parameter.

- `--buffer_size=<size>`
  The buffer size, in bytes, you would like the flash programmer design to use to communicate with the host. The buffer size is largely dependent on the amount of available RAM in the FPGA device being used. Table 2–1 lists the appropriate values.

Table 2–1. Buffer Sizes for Stratix & Cyclone Devices

Stratix Device	Buffer Size (Bytes)	Cyclone Devices	Buffer Size (Bytes)
1S10	16384	1C3	512
1S20	32768	1C4	2048
1S25	32768	1C6	4096
1S30	65536	1C12	16384
1S40	65536	1C20	16384
1S60	65536		
1S80	65536		

Figure 2–1. MK_TARGET_BOARD Output



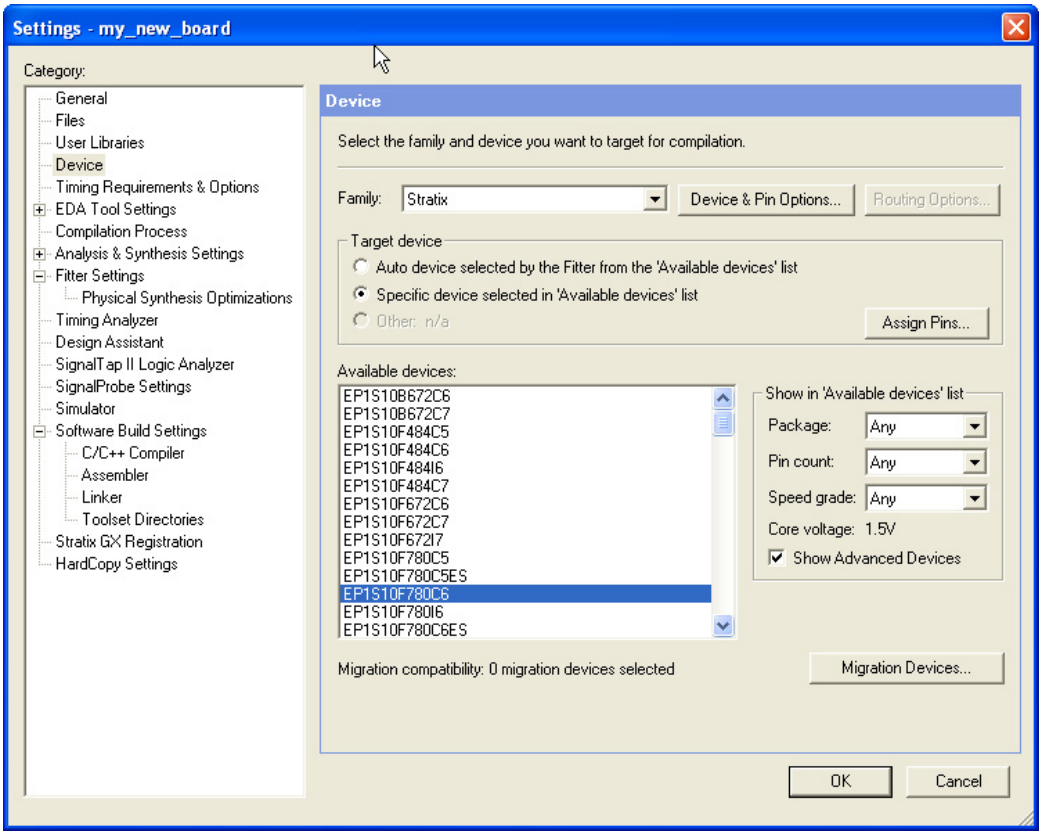
```
/cygdrive/c/altera/kits/nios2/examples/verilog/niosII_cyclone_1c20/standard
[SOPC Builder]$ mk_target_board --name=my_new_board --family=cyclone --clock=50
 --index=1 --buffer_size=16384 --epcs=U59
Make Target Board version 1.01
Switch summary:
   Board name: "my_new_board"
   Board class: my_new_board
   Device family: Cyclone
   Clock frequency: 50000000hz
   Payload buffer size: 16384 bytes
   EPCS flash enabled with reference designator: U59

Finished making target board.

A subdirectory named:
 c:/altera/kits/nios2/examples/verilog/niosII_cyclone_1c20/standard/my_new_board

 has been created that contains a new generic board component and
 associated QuartusII project.

Next step:  Edit the newly created board component and QuartusII project
 to reflect the actual board being targeted.

Before you can use this board component for flash programming, it must
 first be edited in QuartusII and SOPC Builder to reflect the actual board
 being targeted.

It is important that you consult the document
 "NiosII Flash Programmer User Guide", available at www.altera.com, before
 attempting to port the NiosII Flash Programmer to custom boards.

/cygdrive/c/altera/kits/nios2/examples/verilog/niosII_cyclone_1c20/standard
[SOPC Builder]$
```

This will create a subdirectory within your actual project directory with the name you specified in the `mk_target_board` command line. The subdirectory will contain your new generic target board template. See Figure 2–1.

4. Start Quartus II software and open the new project in the subdirectory that was just created by `mk_target_board`.

5. From Quartus II software, choose **Assignments** > **Device** and select the device you are using on your board. See Figure 2–2.

Figure 2–2. Device Assignment Dialog Box



Select the family and device you want to target for compilation.

Family: Stratix

Target device
- Auto device selected by the Fitter from the 'Available devices' list
- [✓ Specific device selected in 'Available devices' list]
- Other: n/a

Available devices:

```
EP1S10B672C6
EP1S10B672C7
EP1S10F484C5
EP1S10F484C6
EP1S10F484I6
EP1S10F484C7
EP1S10F672C6
EP1S10F672C7
EP1S10F672I7
EP1S10F780C5
EP1S10F780C5ES
EP1S10F780C6
EP1S10F780I6
EP1S10F780C6ES
```

Show in 'Available devices' list
Package: Any
Pin count: Any
Speed grade: Any
Core voltage: 1.5V
✓ Show Advanced Devices

Migration compatibility: 0 migration devices selected

6. Click **OK**.

7. Choose **Tools -> SOPC Builder** to start the SOPC Builder tool.

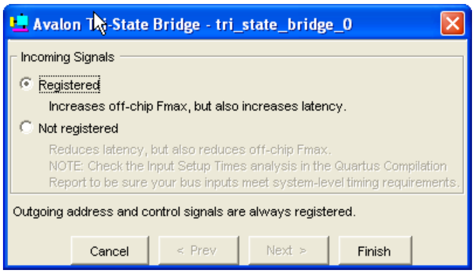
8. Ensure that the SOPC Builder’s **System Contents** tab is selected.

Because we are creating a target board and not a real system, it is desirable to leave the **Target** field as Unspecified Board.

9. Ensure the System Clock Frequency is set to the frequency at which you intend to run the flash programmer design.

10. Add the Avalon Tri-State bridge component to the system, choose **Bridges > Avalon Tri-State Bridge** from the components list on the left. Accept the default parameters. See Figure 2–3.

Figure 2–3. Avalon Tri-State Bridge Wizard



Incoming Signals

(●) Registered

Increases off-chip Fmax, but also increases latency.

( ) Not registered

Reduces latency, but also reduces off-chip Fmax.
NOTE: Check the Input Setup Times analysis in the Quartus Compilation Report to be sure your bus inputs meet system-level timing requirements.

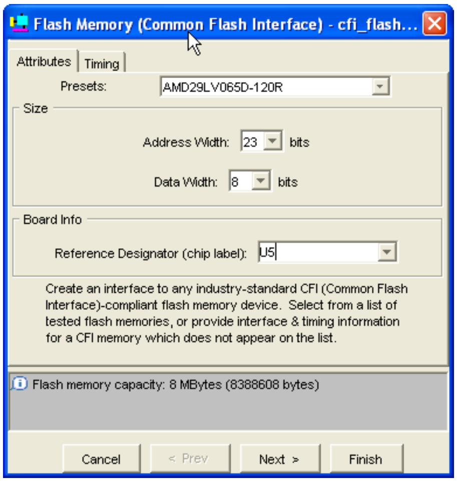
Outgoing address and control signals are always registered.

11. For each flash device on your board that is connected to the FPGA, add a flash memory (CFI) component. Choose **Memory** > **Flash Memory (Common Flash Interface)**. See Figure 2–4.

12. In the **Flash Memory Component** wizard, be sure to enter the appropriate parameters for each flash device, or select your flash device from the list of presets at the top of the **Flash Memory Component** wizard, if it exists there.

13. In the **Flash Memory Component** wizard, be sure to specify a reference designator for each flash device. If the device corresponds to a CFI flash component in your actual design, ensure that the reference designator chosen matches the reference designator you gave the CFI flash component in your actual design. A device's reference designator is often silk-screened next to the actual device mounted on the board, e.g., "U5" is the reference designator of the flash device on the Altera Nios development boards. See Figure 2–4.

Figure 2–4. Flash Memory Wizard



Presets: AMD29LV065D-120R

Address Width: 23 bits

Data Width: 8 bits

Reference Designator (chip label): U5

Create an interface to any industry-standard CFI (Common Flash Interface)-compliant flash memory device. Select from a list of tested flash memories, or provide interface & timing information for a CFI memory which does not appear on the list.

Flash memory capacity: 8 MBytes (8388608 bytes)

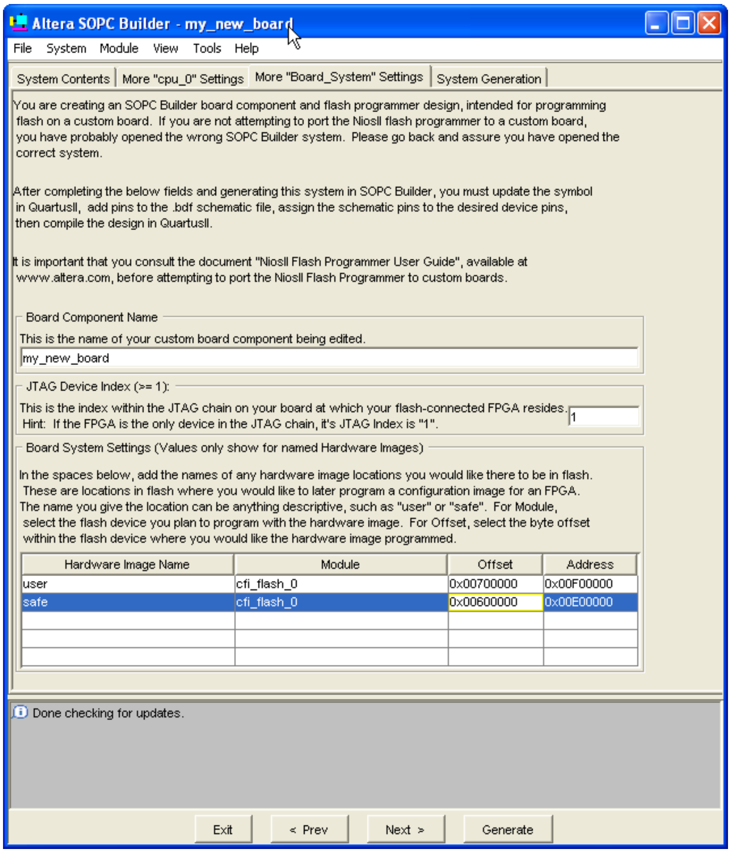
14. Select the **More "Board_System" Settings** tab in SOPC Builder. See Figure 2–5.

15. The **Board Component Name** and **JTAG Device Index** value should be the same as was entered in the `mk_target_board` utility.

16. For each hardware configuration image location you would like to exist in your flash, type a name in the **Hardware Image Name** column. Common hardware image names are "user," "safe," etc.

Refer to *AN 346: Using the Nios Development Board Configuration Controller Reference Designs* for more information on "user" and "safe" images.

Figure 2–5. More Board System Settings


Altera SOPC Builder - my_new_board
File System Module View Tools Help
System Contents
More "cpu_0" Settings
More "Board_System" Settings
System Generation
Exit
< Prev
Next >
Generate

You are creating an SOPC Builder board component and flash programmer design, intended for programming flash on a custom board. If you are not attempting to port the NiosII flash programmer to a custom board, you have probably opened the wrong SOPC Builder system. Please go back and assure you have opened the correct system.

After completing the below fields and generating this system in SOPC Builder, you must update the symbol in QuartusII, add pins to the .bdf schematic file, assign the schematic pins to the desired device pins, then compile the design in QuartusII.

It is important that you consult the document "NiosII Flash Programmer User Guide", available at www.altera.com, before attempting to port the NiosII Flash Programmer to custom boards.

Board Component Name

This is the name of your custom board component being edited.

my_new_board

JTAG Device Index (>= 1):

This is the index within the JTAG chain on your board at which your flash-connected FPGA resides. 1

Hint: If the FPGA is the only device in the JTAG chain, it's JTAG Index is "1".

Board System Settings (Values only show for named Hardware Images)

In the spaces below, add the names of any hardware image locations you would like there to be in flash. These are locations in flash where you would like to later program a configuration image for an FPGA. The name you give the location can be anything descriptive, such as "user" or "safe". For Module, select the flash device you plan to program with the hardware image. For Offset, select the byte offset within the flash device where you would like the hardware image programmed.

Hardware Image Name	Module	Offset	Address
user	cfi_flash_0	0x00700000	0x00F00000
safe	cfi_flash_0	0x00600000	0x00E00000

Done checking for updates.

17. For each **Hardware Image Name** you enter, select the flash module (**Module** column) in which you would like the hardware image name located. Options here will be any CFI flash or EPCS serial configuration device components you instantiate in the target board system.

18. For each **Hardware Image Name**, select the offset within the flash module in which you would like the hardware image to exist. For example, the “user” and “safe” hardware image locations on the Nios Development Boards are `0x700000` and `0x600000`, respectively.

If you are setting a **Hardware Image Name** to be located in an EPCS serial configuration device, no offset is allowed.

19. Click the **System Generation** tab in SOPC Builder.

20. Turn off the **Simulation** check box, as you are creating a target board, not a complete system. Simulation is not applicable.

21. Click **Generate.**

22. When generation is complete, click **Exit**.

23. If asked to update the symbol you just changed in SOPC Builder, click **Yes**. If not asked to update the symbol, do so anyway by right-clicking the system symbol and selecting **Update Symbol** or **Block**.

24. The flash interface signals should now appear on the system symbol.

25. Add the appropriate number of output and `bidir` pins for the new flash interface to the design by double-clicking on an empty space in the schematic and entering `output` or `bidir`, then click **OK**.

26. Connect the pins to their corresponding flash interface signals on the system symbol.

27. Name the pins by double-clicking them.

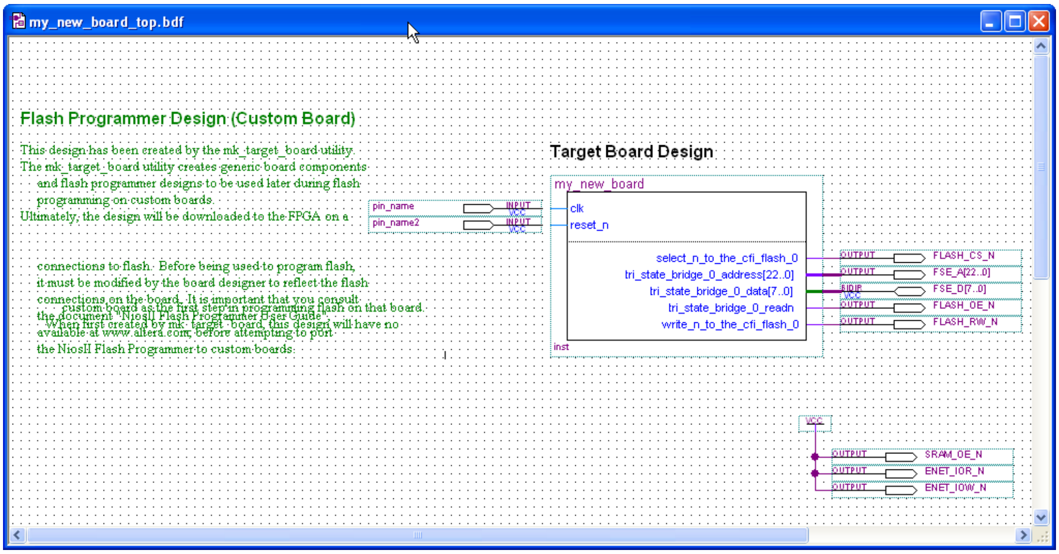
It is highly recommended you name the pins exactly the same as they are named in your actual design. This will help ease the upcoming step of making pin assignments.

28. **THIS IS A VERY IMPORTANT STEP**:

If there are other non-flash devices on the same external tri-state bus, you **<u>must</u>** add output pins for their output enable (`OE`), read enable, or other such pins. The pins must be connected to either `GND` or `VCC`, depending on the polarity of the external device pin to disable the output from the other devices. Otherwise, there will likely be tri-state bus contention, and damage to the devices or the FPGA's can occur. See Figure 2–6.

As with the flash interface pins, it is recommended that you name the other device `OE` pins the same as in your actual design to ease the next step of making pin assignments.

Figure 2–6. BDF with Flash Pins & Other Device Pins



Flash Programmer Design (Custom Board)

This design has been created by the mk_target_board utility.
The mk_target_board utility creates generic board components
and flash programmer designs to be used later during flash
programming on custom boards.
Ultimately, the design will be downloaded to the FPGA on a
custom board as the first step in programming flash on that board.
When first created by mk_target_board, this design will have no
connections to flash. Before being used to program flash,
it must be modified by the board designer to reflect the flash
connections on the board. It is important that you consult
the document "NiosII Flash Programmer User Guide",
available at www.altera.com, before attempting to port
the NiosII Flash Programmer to custom boards.

29. Make pin assignments to the new pins.

    a. If you have named the flash and other devices' `OE` pins the same as in your actual design, follow these steps to make your pin assignments:

        - Open the **.qsf** file for your actual project in Quartus II
        - Open the **.qsf** file for your new target board project in Quartus II.
        - Copy the **Pin & Location Assignments** section from your actual project's **.qsf** file to your target board project's **.qsf** file.
        - Save your target board project's **.qsf** file.
        - Close both **.qsf** files.

    b. If you have **not** named the flash and other devices' IO pins the same as in your actual design, you will have to make your pin assignments by following the Quartus II Help instructions for making pin assignments.

30. Choose **Save** (File menu).

31. Choose **Processing > Start Compilation**.

Selecting Your New Target Board

Now that you have created a target board, you can select it as the target for your original design, which will then enable flash programming.

When the target board is created by the `mk_target_board` utility, it is automatically added to the SOPC Builder component search path so that other projects may use it from SOPC Builder. However, if you move the target board, it may no longer be in that search path. To inspect and edit the SOPC Builder component search path, open SOPC Builder (**Tools > SOPC Builder)** and choose **SOPC Builder Setup** (File menu). The Component/Kit Library Search Path field contains the additional locations SOPC Builder will search for components and target boards when the software launches.

Perform the following steps to instantiate the new target board in your design.

1. Open your actual original project in Quartus II.

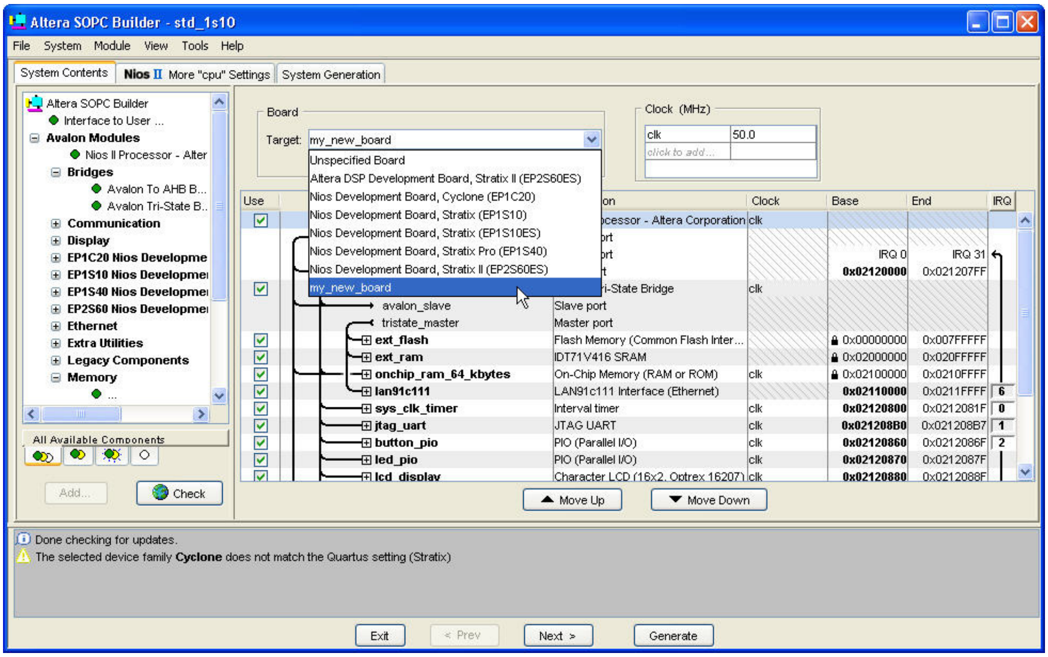
2. Open SOPC Builder by choosing **Tools > SOPC Builder**.

3. Ensure the **System Contents** tab is selected.

4. From the **Target** drop-down list, select your new target board.

    If your target board is not shown in the **Target** drop-down list, something in the previous steps was not successful. Ensure that the target board project directory is in either your actual project directory, or some other directory within the SOPC Builder component search path. See Figure 2–7.

Figure 2–7. Target Drop Down List in SOPC Builder



Board

Target: my_new_board

- Unspecified Board
- Altera DSP Development Board, Stratix II (EP2S60ES)
- Nios Development Board, Cyclone (EP1C20)
- Nios Development Board, Stratix (EP1S10)
- Nios Development Board, Stratix (EP1S10ES)
- Nios Development Board, Stratix Pro (EP1S40)
- Nios Development Board, Stratix II (EP2S60ES)
- my_new_board

Clock (MHz)

clk	50.0
click to add...	

Use		on	Clock	Base	End	IRQ
✓		ocessor - Altera Corporation	clk			
		ort				
		ort		IRQ 0	IRQ 31	
		t		0x02120000	0x021207FF	
✓		i-State Bridge	clk			
	avalon_slave	Slave port				
	tristate_master	Master port				
✓	ext_flash	Flash Memory (Common Flash Inter...		0x00000000	0x007FFFFF	
✓	ext_ram	IDT71V416 SRAM		0x02000000	0x020FFFFF	
✓	onchip_ram_64_kbytes	On-Chip Memory (RAM or ROM)	clk	0x02100000	0x0210FFFF	
✓	lan91c111	LAN91c111 Interface (Ethernet)		0x02110000	0x0211FFFF	6
✓	sys_clk_timer	Interval timer	clk	0x02120800	0x0212081F	0
✓	jtag_uart	JTAG UART	clk	0x021208B0	0x021208B7	1
✓	button_pio	PIO (Parallel I/O)	clk	0x02120860	0x0212086F	2
✓	led_pio	PIO (Parallel I/O)	clk	0x02120870	0x0212087F	
✓	lcd_display	Character LCD (16x2, Optrex 16207)	clk	0x02120880	0x0212088F	

Done checking for updates.

The selected device family **Cyclone** does not match the Quartus setting (Stratix)

5. Select the **System Generation** tab.
6. Click **Generate**.
7. When generation finishes, click **Exit**.
8. If asked to update the symbol you just changed in SOPC Builder, click **Yes**.
9. Choose **File > Save**.
10. Choose **Processing > Start Compilation**.

You are now ready to program flash. Assuming everything was done correctly, you should now be able to program flash by following the instructions in Chapter 1, Using the Nios II Flash Programmer & Instantiating Flash Memory.