Nios® II

Nios II Flash Programmer User Guide

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

UG-NIOSIIFLSHPROG-1.2

Contents

About this User Guide

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

The table shows this document's revision history.

Refer to the Nios II release notes and errata at www.altera.com for late-breaking information that is not available in this document.

Nios II Flash Programmer User Guide Revision History	
Date	**Description**
October 2005	Updates for the Nios II version 5.1 release. Includes major chages to the flash programmer target design.
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 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/	www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	(800) 767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox 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."
`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.
CAUTION	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. Overview of the Nios II Flash Programmer

Introduction

The purpose of the Nios® II flash programmer is to program data into a flash memory device connected to an Altera® FPGA. The flash programmer sends file contents over an Altera download cable, such as the USB Blaster®, to a Nios II system running on the FPGA. Many hardware designs that include the Nios® II processor also incorporate flash memory on the board to store FPGA configuration data or Nios II program data. The Nios II flash programmer is part of the Nios II development tools, and is a convenient method of programming this memory.

The Nios II flash programmer can program three types of content into flash memory:

- Nios II software executable files - Many systems use flash memory to store nonvolatile program code, or firmware. Nios II systems can boot out of flash memory.
- FPGA configuration data - At system power-up, the FPGA configuration controller on the board can read FPGA configuration data from the flash memory. Depending on the design of the configuration controller, it might be able to choose between multiple FPGA configuration files stored in flash memory.
- Other arbitrary data files - The Nios II flash programmer can program a binary file to an arbitrary offset in a flash memory for any purpose. For example, a Nios II program might use this data as a coefficient table or a sine lookup table.

You can use the flash programmer to program the following types of memory:

- CFI-compliant flash memory - Common flash interface, or CFI, is an industry standard that provides a common, vendor-independent interface to flash memory devices.
- Altera EPCS serial configuration device - Altera EPCS serial configuration devices can store FPGA configuration data and Nios II executable software.

For further information on the CFI specification, see **www.intel.com/design/flash/swb/cfi.htm**. For further information on EPCS devices, see the *Serial Configuration Devices (EPCS1, EPCS4, EPCS16 & EPCS64) Data Sheet* and the *EPCS Device Controller Core with Avalon Interface* chapter of the *Quartus II Handbook, Volume 5: Embedded Peripherals*.

In this document, the term "flash memory" refers to both CFI and EPCS memory devices, unless otherwise noted.

Prerequisites

This user guide assumes that you are familiar with the Nios II hardware and software development flow. You should be familiar with the contents of the following tutorials:

- Nios II Hardware Development Tutorial
- Nios II Software Development Tutorial, which is available in the Nios II integrated development environment (IDE) help system

If you use the Nios II flash programmer to program FPGA configuration data to flash memory, you also must understand the configuration method used on the board.

Refer to *AN346: Using the Nios Development Board Configuration Controller Reference Design* or the reference manual for a specific Nios development board.

Changes from Earlier Versions

The Nios II development tools version 5.1 include a completely redesigned flash programmer back end. If you used the flash programmer in prior versions of the Nios II development tools, read this user guide carefully to familiarize yourself with the changes. The main points you need to be aware of are the following:

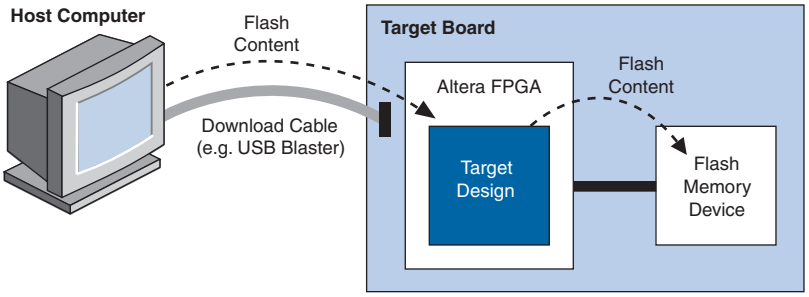
- You must rebuild Nios II software projects created prior to version 5.1, because the formats of some output files have changed.
- The Nios II flash programmer no longer requires a special hardware design to program flash memory. You can run any suitable hardware design on the FPGA.
- The syntax of the command-line utilities has changed. If you have scripts that call the flash programmer command-line utilities, you probably need to update your scripts for the new syntax.

For more information on command-line syntax, see Chapter 3, Using the Flash Programmer in Command-Line Mode for details.

How the Flash Programmer Works

The flash programmer has two parts, the host and the target, as shown in Figure 1–1. The host portion runs on your computer. It sends flash programming files over a download cable to the target. The target portion is your hardware design, running in the FPGA. It accepts the programming data sent by the host, and writes data to the flash memory device. In order to work with the Nios II flash programmer, your FPGA design must meet certain requirements. See "Flash Programmer Target Design" on page 1–3.

Figure 1–1. How the Nios II Flash Programmer Works



IDE and Command-Line Modes

You can run the Nios II flash programmer in either of two modes:

- IDE Mode – The Nios II IDE provides an easy-to-use interface to the flash programmer features. The IDE mode is suitable for most flash programming needs.
- Command-Line Mode - For advanced users, command-line mode provides complete control over the flash programmer features. You can run the command-line flash programmer utilities from a command shell such as the Nios II SDK shell. You might have to calculate some parameters manually.

In this document, the terms "Nios II flash programmer" and "flash programmer" refer to both IDE mode and command-line mode, unless explicitly noted.

Flash Programmer Target Design

To use the Nios II flash programmer, you must have a valid flash programmer target design. The target design contains an SOPC Builder system with the following characteristics:

- Contains at least the SOPC Builder components shown in Table 1–1.
- Specifies a target board.

The target design must be running on the FPGA before you can run the Nios II flash programmer on the host.

Minimum Component Set

The minimum component set provides facilities for the target design to communicate with the host, and to write to flash memory. The minimum component set depends on the type of flash memory you intend to program. Table 1–1 lists the minimum component set.

Table 1–1. Minimum Component Set for the Flash Programmer Target Design

Component	Flash memory to be programmed		
	CFI Only	EPCS Only	CFI and EPCS
Nios II Processor, with JTAG debug module level 1 or greater	Required	Required	Required
System ID Peripheral	Recommended *(1)*	Recommended *(1)*	Recommended *(1)*
Avalon Tristate Bridge	Required		Required
Flash Memory (Common Flash Interface)	Required *(2)*		Required *(2)*
EPCS Serial Flash Controller		Required	Required

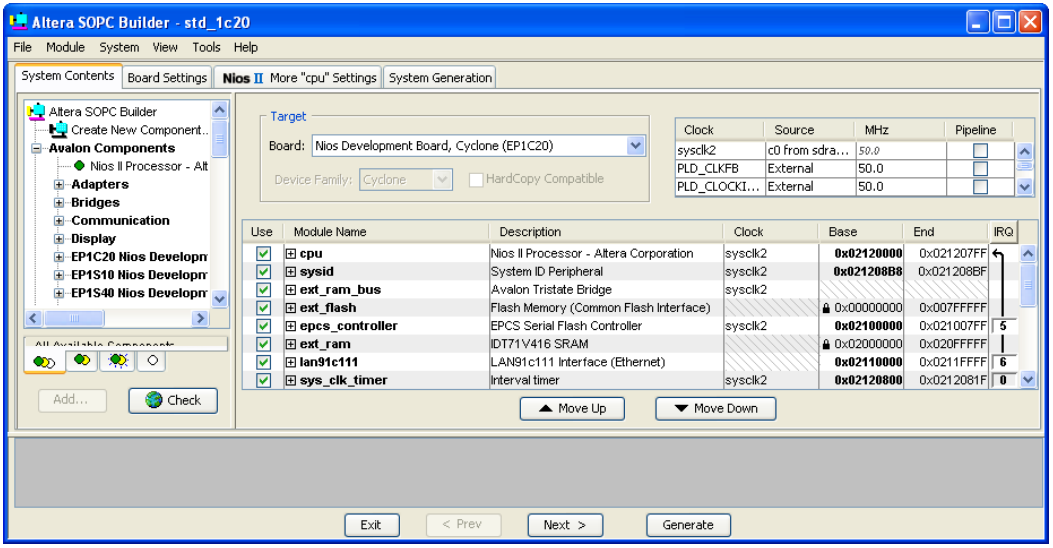
Notes to Table 1–1:

(1) If present, a **System ID Peripheral** component allows the flash programmer to validate the target design before programming the flash memory.

(2) The system can contain more than one CFI flash memory. The system must contain one **Flash Memory (Common Flash Interface)** component for each flash memory on the board.

Figure 1–2 shows an example of an SOPC Builder system containing the minimum component set for a system with one CFI flash memory and an EPCS serial configuration device. The system also includes other components which relate to the purpose of the system, not the flash programmer.

Figure 1–2. Example Target Design Containing the Minimum Component Set



The **full_featured** or **standard** hardware example designs included with Nios II development tools are ready-made target designs that work with Altera development boards. If you are developing for a custom board, consider using one of these example designs as a starting point in creating your first target design.

Board Description File

The SOPC Builder system in the target design must specify a target board in SOPC Builder. To specify a target board, you must have an SOPC Builder board description for the board. The board description provides necessary information to the flash programmer, such as the addresses and names of flash memories in the system.

Altera provides board descriptions for Altera Nios development boards. If you are using a custom board, you need to create a board description with the SOPC Builder board description editor.

For more information, see the *Board Description Editor* chapter of the *Quartus II Handbook, Volume 4: SOPC Builder*.

2. Using the Flash Programmer in IDE Mode

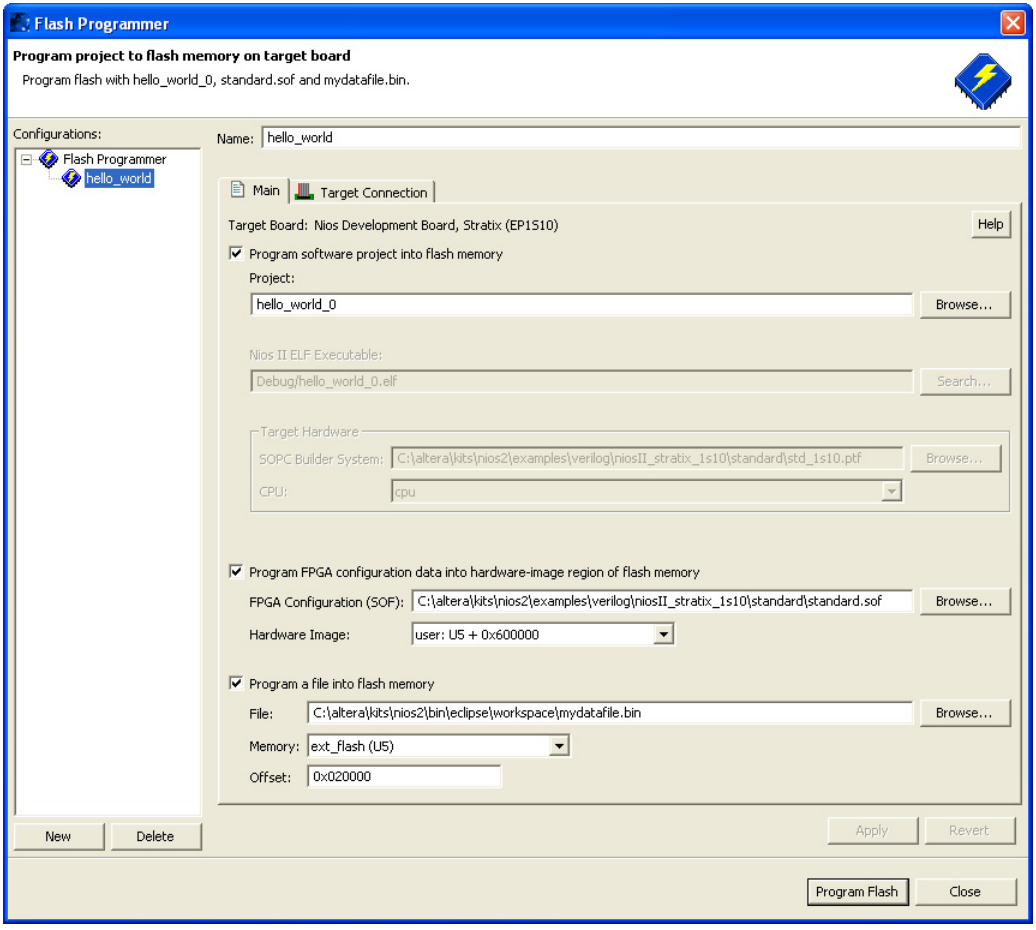
The Nios II integrated development environment (IDE) automates the process of programming flash memory, and allows you to control the programming parameters with an easy-to-use graphical interface. The IDE lets you program any combination of software, hardware, and binary data into flash memory in one operation. The IDE mode is the recommended method to use the Nios II flash programmer.

For details on using the flash programmer in command-line mode, see Chapter 3, Using the Flash Programmer in Command-Line Mode.

The Flash Programmer Dialog

To open the Nios II flash programmer in the Nios II IDE, on the **Tools** menu, click **Flash Programmer**. Figure 2–1 shows an example of the **Flash Programmer** dialog.

Figure 2–1. Flash Programmer Dialog in the Nios II IDE



Using the **Flash Programmer** dialog in the IDE is a two-step process:

1. Specify the data contents to program, and select which flash memory and where in the memory to program.
2. Click **Program Flash**. The IDE performs the sequence of operations required to program all the specified files into flash memory.

Refer to the Nios II IDE help system for an explanation of controls in the **Flash Programmer** dialog. Click **Help** in the upper-right-hand corner of the **Flash Programmer** dialog to open the help system.

If your target design has a System ID component, the IDE verifies that the system is running on the FPGA before attempting to program flash memory. If it is not, the flash programmer does not continue to program the flash memory. The IDE skips this check if there is no System ID component in the target system.

Regardless of the system ID, you cannot program flash memory if the hardware design configured in the FPGA is not a valid flash programmer target design. Therefore, the Nios II hardware system for your C/C++ application project must be a valid flash programmer target system, containing at least the minimum component set specified in Table 1–1 on page 1–4.

The IDE also provides the Read-Only Zip File System software component, which is another easy-to-use tool for storing data to flash memory. Depending on your application, you might find it more convenient to use the Read-Only Zip Filing System.

For details, see the Zip Read-Only File System topic in the Nios II IDE help system.

3. Using the Flash Programmer in Command-Line Mode

The Nios II development tools provide four command-line utilities which give you complete control of the Nios II flash programmer features. You can create a custom script file to automate a flash programming task. Using the flash programmer in command-line mode gives you more control than IDE mode, but it is also more complex. When possible, Altera recommends using IDE mode to program flash.

The Nios II development tools version 5.1 include a completely redesigned flash programmer. If you have used the command-line utilities in prior versions of the Nios II development tools, read the argument descriptions carefully to familiarize yourself with the changes. In particular, note that:

- The Nios II flash programmer no longer requires a special hardware design to program flash memory. The flash programmer now uses your application hardware design as the target hardware. For details, see "Flash Programmer Target Design" on page 1–3.
- The syntax of the command-line utilities has changed.

Table 3–1 lists the command-line utilities.

Table 3–1. Flash Programmer Command-Line Utilities

`nios2-flash-programmer`	Programs an S-record file into flash memory. Can also read back data, verify data, provide debug information about the flash chip, and more.
`sof2flash`	Converts an FPGA configuration (**.sof**) file to S-record format.
`elf2flash`	Converts a Nios II software executable (**.elf**) file to S-record format.
`bin2flash`	Converts an arbitrary data file to S-record format.

The Nios II IDE programs flash by creating a script based on the command-line utilities. The script is well-formed, customized to your project, and human-readable. You can use it as a reference for flash programmer command-line syntax. After you successfully program flash memory using the IDE, you can find the script in the **C/C++ Projects** view in your project's **Debug** or **Release** folder. The flash programmer script is a file with extension **.sh** named **<***Project Name***>_programmer.sh**.

The main utility for programming flash memory from the command line is **nios2-flash-programmer**. It requires industry-standard S-record input files. The file conversion utilities **sof2flash**, **elf2flash** and **bin2flash** create the S-record files for **nios2-flash-programmer**. These utilities ensure that the input is compatible with the flash programmer. Input file names for all utilities must include an explicit extension, such as **.elf** or **.flash**.

On Windows computers, when you launch the Nios II SDK shell, the flash programmer utilities are available in your default search path. For more detail about the Nios II SDK Shell, see the *Altera-Provided Development Tools* chapter of the *Nios II Software Developer's Handbook*.

The following sections list the utilities and their functions.

nios2-flash-programmer

The **nios2-flash-programmer** utility programs a preformatted file into a specified flash memory. The input is an industry-standard S-record file, normally created by one of the conversion utilities, **sof2flash**, **elf2flash**, or **bin2flash**. **nios2-flash-programmer** can use any S-record file as an input, provided that the addresses specified in the S-record file represent offsets from the beginning of flash memory. The Nios II IDE creates flash programmer files with a **.flash** extension.

The **nios2-flash-programmer** utility is capable of programming, erasing, or reading from any CFI-compatible flash memory or EPCS serial configuration device in the hardware target design.

The **nios2-flash-programmer** command-line syntax is:

```
nios2-flash-programmer [--help] [--cable=<cable name>]
[--device=<device index>] [--instance=<instance>]
--base=<address> [--epcs] { <file> } [--go]
```

Table 3–2 lists the parameters commonly used with **nios2-flash-programmer**.

Table 3–2. nios2-flash-programmer Parameters (Part 1 of 2)

Name	Required?	Description
General Parameters		
`--cable=<cable name>`	Required if there are multiple download cables connected to the host computer.	Specifies which download cable to use.
`--device=<device index>`	Required if there are multiple devices in the JTAG chain.	Specifies the FPGA's device number in the JTAG chain. This is the device where the flash programmer looks for the Nios II JTAG debug module. JTAG devices are numbered relative to the JTAG chain, starting at 1. *(1)*
`--instance=<instance>`	Required if there are multiple Nios II CPUs with JTAG debug modules in the target design on the FPGA.	Specifies which Nios II JTAG debug module to look at in the FPGA. This is the JTAG debug module that is used for programming flash memory.
`--erase=<start>,<size>`	Optional; defaults off.	Erases a range of bytes in the flash memory.
`--erase-all`	Optional; defaults off.	Erases the entire flash memory. The erase operation occurs before programming, if an input file is provided for programming.
`--program`	Optional; defaults on if an input file is specified.	Programs flash memory from the input files.
`--no-keep-nearby`	Optional; defaults off.	Throws away partial sector data. If the data to program does not completely fill the first or last sector, the flash programmer normally preserves and reprograms the original data in those sectors. The `--no-keep-nearby` parameter disables this feature. This option speeds up the programming process, but is only appropriate if the existing flash memory contents are unimportant.
`--verify`	Optional; defaults off.	Verifies that contents of flash memory match input files.
{ *<file>* }	Optional.	Specifies the name(s) of the input file(s) to program or verify. Separate multiple file names with spaces.
`--read=<file>`	Optional; defaults off.	Reads flash memory contents into the specified file.

Table 3–2. nios2-flash-programmer Parameters (Part 2 of 2)		
Name	**Required?**	**Description**
`--read-bytes=<start>,<size>`	Optional if `--read` is specified; defaults off.	Specifies which address range to read (byte addresses).
`--go`	Optional; defaults off.	Runs the CPU from its reset vector after flash memory programming is complete.
CFI Parameters		
`--debug`	Optional; defaults off.	Prints debug information, including the flash memory's query table.
`--base=<address>`	Required.	Specifies the base address of the CFI flash memory. This is the absolute address in the target design's address space. **nios2-flash-programmer** treats addresses in the S-record files as offsets to the base address.
EPCS Parameters		
`--epcs`	Required when programming an EPCS serial configuration device; defaults off.	Specifies that the target flash memory is an EPCS serial configuration device.
`--debug`	Optional; defaults off.	Prints debug information about the physical memory inside the EPCS device.
`--base=<address>`	Required.	Specifies the base address of the EPCS device.

Notes to Table 3–2:

(1) The `--device` parameter is only needed if there are two processors in different devices with the same instance ID. If necessary, you can find the instance value in your system library project's **generated.sh** file. In the Nios II IDE's **C/C++ Projects** view, expand your system library project, and find the **Debug** or **Release** folder. **generated.sh** is in the **system_description** subfolder. Open **generated.sh**, and find the instance number next to the keyword `nios2_instance=`. The instance number is unrelated to the CPU number.

For additional parameters, type `nios2-flash-programmer --help` at a command line.

nios2-flash-programmer Command-Line Examples

```
nios2-flash-programmer --cable="Usb-blaster [USB-0]"
--base=0x200000 --program ext_flash.flash
```

Programs CFI flash memory based at address 0x200000 with input file **ext_flash.flash** using a cable named "Usb-blaster [USB-0]"

```
nios2-flash-programmer --epcs --base=0x02100000
epcs_controller.flash
```

Programs an EPCS device based at address 0x02100000 with input file **epcs_controller.flash**.

```
nios2-flash-programmer --base=0x200000 --read=current.srec
--read-bytes=0,0x10000
```

Reads 0x10000 bytes from CFI flash memory based at address 0x200000 and writes the contents to a file named current.srec

```
nios2-flash-programmer --base=0x200000 --erase=0x8000,0x10000
```

Erases address range 0x8000 to 0x10000 in CFI flash memory based at address 0x200000

```
nios2-flash-programmer --base=0x200000 --debug
```

Queries CFI flash memory based at address 0x200000 and reports the result. This dumps the flash memory's query table.

sof2flash

The **sof2flash** utility takes an FPGA configuration file in **.sof** format and translates it to an S-record file, suitable for programming into flash memory.

Table 3–3 lists the typical parameters used with **sof2flash**.

Table 3–3. sof2flash Parameters

Name	Required?	Description
General Parameters		
`--compress`	Optional; defaults off.	Turns on Stratix II compression. Not available on non-Stratix II FPGAs.
`--input=<`*file*`>`	Required.	Name of the input **.sof** file.
`--output=<`*file*`>`	Required.	Name of the output file.
CFI Parameters		
`--offset=<`*addr*`>`	Required.	Offset within the flash memory device where the FPGA configuration is to be programmed.
EPCS Parameters		
`--epcs`	Required for EPCS devices; defaults off.	Specifies that the output is intended for an EPCS device.

For additional parameters, type `sof2flash --help` at a command line.

sof2flash Command-Line Examples

```
sof2flash --offset=0x0 --input=standard.sof
--output=standard_cfi.flash
```

Converts **standard.sof** to an S-record file named **standard_cfi.flash** intended for a CFI flash memory. The S-record offset begins at 0x0.

```
sof2flash --epcs --input=standard.sof
--output=standard_epcs.flash
```

Converts **standard.sof** to an S-record file named **standard_epcs.flash** intended for an EPCS device.

elf2flash

The **elf2flash** utility takes a software executable file in **.elf** format, and translates it to an S-record file suitable for programming into flash memory.

elf2flash also inserts a boot copier into the flash file, if needed. **elf2flash** inserts the boot copier code before the application code under the following conditions:

- The CPU's reset address falls within the address range of the flash memory being programmed.
- The executable code is linked to a memory location outside of the flash memory being programmed.

If **elf2flash** inserts a boot copier, it also translates the application code **.elf** file to a boot record for use by the boot copier. This boot record contains all of the application code, but is not executable. After reset, the boot copier reads the boot record from flash memory and copies the application code to the correct linked address, and then branches to the newly-copied application code.

Table 3–4 lists the typical parameters used with **elf2flash**.

Table 3–4. elf2flash Parameters

Name	Required?	Description
General Parameters		
`--input=<file>`	Required.	The name of the input **.elf** file.
`--output=<file>`	Required.	The name of the output file.
CFI Parameters		
`--base=<addr>`	Required.	The base address of the flash memory component. **elf2flash** uses this parameter with `--end` and `--reset` to determine whether the system requires a boot copier.
`--end=<addr>`	Required.	The end address of the flash memory component. **elf2flash** uses this parameter with `--base` and `--reset` to determine whether the system requires a boot copier.
`--reset=<addr>`	Required.	The CPU reset address, which is specified in SOPC Builder. **elf2flash** uses this parameter with `--base` and `--end` to determine whether the system requires a boot copier.
`--boot=<file>`	Required under the following conditions: • The CPU's reset address falls within the address range of the flash memory being programmed. • The executable code is linked to a memory location outside of the flash memory being programmed.	Specifies the boot copier object code file. Ignored if the boot copier is not required. If **elf2flash** determines that a boot copier is required, but the `--boot` parameter is absent, **elf2flash** displays an error message. The Altera-provided boot copier is located at <*Nios II Kit Path*>/**components/altera_nios2/boot_loader_cfi.srec**.
EPCS Parameters		
`--epcs`	Required when creating files for an EPCS device; defaults off.	Specifies that the output is intended for an EPCS device.
`--after=<file>`		**elf2flash** uses this parameter to position a Nios II executable in an EPCS device along with an FPGA configuration. For further details, see "Programming Both Hardware and Software into an EPCS Device".

For additional parameters, type `elf2flash --help` at a command line.

Programming Both Hardware and Software into an EPCS Device

The `--base` parameter is not available for EPCS devices, because in EPCS devices, FPGA configuration data must start at address 0x0. However, if you are programming both an FPGA configuration and a Nios II software executable in the EPCS device, the `--after` parameter lets you position the software executable directly after the FPGA configuration data.

Convert the FPGA configuration file first using **sof2flash**. When converting the Nios II software executable, use the `--after` parameter, and specify the FPGA configuration S-record file. The S-record output for the software executable starts at the first address unused by the FPGA configuration. See the second example under "elf2flash Command-Line Examples" on page 3–8.

elf2flash does not insert the FPGA configuration into the output file. It simply leaves space, starting at offset 0x0, that is large enough for the configuration data.

elf2flash Command-Line Examples

```
elf2flash --base=0x0 --reset=0x0 --boot=boot_loader_cfi.srec
--input=myapp.elf --output=myapp.flash
```

Converts **myapp.elf** to an S-record file named **myapp.flash**, intended for a CFI flash memory based at 0x0. If required, includes a boot copier named **boot_loader_cfi.srec**.

```
elf2flash --epcs --after=standard.flash --input=myapp.elf
--output=myapp.flash --boot=boot_loader_epcs.srec
```

Converts **myapp.elf** to an S-record file named **myapp.flash**, intended for an EPCS device. The S-record output starts at the first address unused by **standard.flash**.

bin2flash

The **bin2flash** utility converts an arbitrary file to an S-record file suitable for use by the flash programmer. You can use **bin2flash** to convert read-only binary data needed by a Nios II program, such as software configuration tables.

Depending on your application, you might find it more convenient to use the Read-Only Zip Filing System, which serves the same purpose.

For details, see the Zip Read-Only File System topic in the Nios II IDE help system.

Do not use **bin2flash** to convert executable software files or FPGA configuration files. To convert Nios II software executable files, use **elf2flash**. To convert FPGA configuration files, use **sof2flash**.

Table 3–5 lists the typical parameters used with **bin2flash**.

Table 3–5. bin2flash Parameters

Name	Required?	Default	Description
`--location=<addr>`	Required	N/A	Offset within the flash memory where the data is to be programmed
`--input=<file>`	Required	N/A	Name of the input binary file being converted
`--output=<file>`	Required	N/A	Name of the output file

For additional parameters, type `elf2flash --help` at a command line.

bin2flash Command-Line Example

```
bin2flash --location=0x40000 --input=data.bin
--output=data.flash
```

Converts data.bin to an S-record file named **data.flash**. Addresses in the S-record place the data starting at offset 0x40000 from the beginning of flash memory.

Appendix A. Non-Standard Flash Memories

This section covers advanced topics to support non-standard CFI flash memory. To use the procedures in this section, you need the datasheet for the flash memory device you are using. Make sure you fully understand the CFI aspects of the device.

Some CFI flash memory devices contain missing or incorrect CFI table information. In this case, the Nios II flash programmer might fail based on the erroneous information in the CFI table. For these devices, the Nios II flash programmer provides the following methods to override the CFI table and successfully program flash memory:

- Built-in recognition and override
- Flash override files
- Width mode override

Built-in Recognition and Override

The Nios II flash programmer contains code to recognize some devices with known CFI table problems. On these devices, it automatically overrides the incorrect table entries. Always try using built-in recognition and override before trying to create an override file. To determine whether the flash programmer recognizes the device, run the flash programmer from the command line with the `--debug` option. If the flash programmer automatically overrides the CFI table, the flash programmer displays a message "Override data for this device is built in".

For details on using the flash programmer in command-line mode, see the Chapter 3, Using the Flash Programmer in Command-Line Mode.

Flash Override Files

To support newly released flash memory devices which might have problems in the CFI table, the Nios II flash programmer provides the ability to override CFI table entries with flash override files. A flash override file lets you manually override erroneous information in the CFI table, which enables the Nios II flash programmer to function correctly.

Before creating an override file, run **nios2-flash-programmer** in command-line mode with the `--debug` parameter, which lists the CFI table found in the device. Compare the debug output with the device's data sheet.

Flash Override File Format

Flash override files contain two sections for each flash memory they override. The first section declares the flash memory type. The second section is the CFI table override data. The flash override file can contain comments preceded by a '#' character.

For example, the SST 39VF800 flash memory contains three incorrect entries in its CFI table at location 0x13, 0x14, and 0x2C. The following example demonstrates how to override the values at those addresses.

```
[FLASH-00BF-2781] # Keyword FLASH, followed by the Mfgr ID and Device ID
                  # These ID values can be found in three ways:
                  # - by consulting the flash memory device's data sheet.
                  # - by using the "autoselect" command
                  # - by running nios2-flash-programmer --debug
CFI[0x13] = 0x02  # The primary command set, found at CFI table -
CFI[0x14] = 0x00  # addresses 0x13 and 0x14 is overridden to 0x02, 0x00.
CFI[0x2C] = 0x01  # The number of CFI Erase block regions, found at
                  # CFI table –address 0x2C is overridden to 0x1.
```

This example is for illustration only. **nios2-flash-programmer** recognizes the SST 39VF800 as a nonstandard CFI device and overrides its CFI table automatically. You do not need to create an override file for this particular part.

How to Use the Flash Override File

There are two ways to deploy flash override files:

1. Place the override file in *<Nios II Kit Path>***/bin**. The flash programmer searches this directory for all filenames matching the pattern `nios2-flash-override*`. The flash programmer loads all these files as override files.

2. Pass the override file to the flash programmer with the `--override` parameter. An example is shown below:

```
nios2-flash-programmer --base 0x0 --override=my_override.txt
sw.flash
```

Width Mode Override Parameter

The override procedure described in "Flash Override Files" on page A–1 assumes the flash programmer detects the correct data-width mode from the CFI query table. In some cases, a 16-bit CFI flash memory device wired in 8-bit mode might return a query table indicating 16-bit mode. This condition prevents the flash programmer from correctly interpreting the remainder of the query table. The flash programmer cannot detect this situation, because the device type is unreadable. If your flash memory device has this problem, you must program it from the command line.

In this case, override the data width on the command line with the hidden parameter `--width=8`.

This parameter is known to be necessary for only two flash memory devices: The ST Micro ST29W800 and ST29W640. Unless you are using these devices, you are unlikely to require this parameter.

Appendix B. Supported Flash Memory Devices

The Nios II flash programmer works with all CFI-compatible parallel flash memories that support programming algorithm 1, 2, or 3, and with all Altera EPCS serial configuration devices. Not all flash memory devices have been tested with the Nios II flash programmer. All the flash memory devices that Altera has verified in hardware with the Nios II development tools version 5.1 are listed below.

If you find a CFI-compliant device that does not work with the Nios II flash programmer, please report it to Altera Technical Support.

The following Altera EPCS devices have been verified to work with the Nios II flash programmer:

- Altera EPCS4
- Altera EPCS16
- Altera EPCS64

The following CFI-compliant devices have been verified to work with the Nios II flash programmer:

- AMD AM29LV128
- AMD AM29LV641
- AMD AM29LV065
- Sharp LH28F160S3T
- Intel TE28F320C3
- Intel TE28F320J3
- SST SST39VF800
- SST SST39VF400
- SST SST39VF200
- Atmel AT49BV332AT
- Atmel AT49BV162AT
- ST Micro ST29W800
- ST Micro ST29W640

Appendix C. Stand-Alone Mode

When developing your hardware and/or software, you use the Nios II flash programmer on a computer with the Quartus II software and the Nios II development tools installed. However, in a production or service environment you might want to set up a computer to program flash memory without installing the full set of Altera development tools.

In stand-alone mode, the flash programmer has limited functionality. You can program any type of CFI or EPCS flash memory. However, the **elf2flash**, **sof2flash**, and **bin2flash** utilities are not available. You must create input files for the flash programmer on a computer which has the Nios II development tools fully installed.

How to Install Nios II Stand-Alone Flash Programmer

Perform the following steps:

1. Install the Quartus II Stand-alone Programmer from the Quartus II CD. The **nios2-flash-programmer** utility requires the Quartus II Stand Alone Programmer to access the JTAG chain on the board.
2. On a computer which has the Nios II development tools fully installed, find the executable file <*Nios II Kit Path*>**/bin/nios2-flash-programmer.exe**.
3. On the stand-alone computer, copy **nios2-flash-programmer.exe** into the directory <*Quartus II Stand-alone Programmer Path*>**/bin**.
4. Open a command prompt on the stand-alone computer. Change directories to the location of the files you wish to program into flash memory.
5. Run the **nios2-flash-programmer** utility as described in Chapter 3, Using the Flash Programmer in Command-Line Mode.

Appendix D. Troubleshooting

This chapter lists troubleshooting tips for the Nios II flash programmer. Each section below describes a common issue you might run into when using the Nios II flash programmer.

Program Flash Button Grayed Out in the IDE

In the **Flash Programmer** dialog in the Nios II IDE, the **Program Flash** button is grayed out.

Probable Cause

You have not fully specified the required parameters in the **Flash Programmer** dialog.

Suggested Actions

- Make sure that your JTAG cable settings are correct. Specify the JTAG settings on the **Target Connection** tab.
- Make sure that you have selected a file to program to the flash memory.

"No Nios II processors available" Error

When you run the flash programmer, you get the error: "There are no Nios II processors available which match the values specified. Please check that your PLD is correctly configured, downloading a new **.sof** file if necessary."

Probable Cause

The flash programmer is unable to connect with a Nios II JTAG debug module inside the FPGA.

Suggested Actions

- Make sure that the FPGA is running a valid flash programmer target design. If not, you need to configure the FPGA using the Quartus II programmer. See "Flash Programmer Target Design" on page 1–3.

- If you are developing on a custom board, make sure that your SOPC Builder board description contains the correct device index for the flash programmer. See the *Board Description Editor* chapter of the *Quartus II Handbook, Volume 4: SOPC Builder*.
- If using the flash programmer in command-line mode, ensure you have specified the proper `--device`, `--cable`, and `--instance` parameter values. See Chapter 3, Using the Flash Programmer in Command-Line Mode for details.

"No CFI table found" Error

When you run the flash programmer to program CFI flash memory, you get the error: "No CFI table found at address <base address of flash memory device>"

Probable Cause

The flash programmer was able to connect with a Nios II JTAG debug module in the FPGA, but it could not successfully execute a query to a flash memory at the base address specified.

Suggested Actions

- If you are using **nios2-flash-programmer** in command-line mode, make sure you specified the correct base address for the CFI device. You can find the flash memory's base address in SOPC Builder.
- Run **nios2-flash-programmer** in command-line mode with the `--debug` parameter. This dumps the flash memory's query table. Compare the output with the flash memory device's data sheet. For further details, see Chapter 3, Using the Flash Programmer in Command-Line Mode.
- Ensure your flash memory hardware is correctly connected to place it at the base address specified in SOPC Builder. Verify this is by running the "Test Flash" routine in the "Memory Test" software template provided in the Nios II IDE. If the test fails, there is a problem with your memory connection. There are two places to look for the problem:
  - The physical connection on your target board
  - The pin assignments on the top-level FPGA design
- If all else fails, make sure the flash memory device you are using does not require an override file. See Appendix A, Non-Standard Flash Memories for details.

"No EPCS registers found" Error

When you run the flash programmer to program an EPCS device, you get the error: "No EPCS registers found: tried looking at addresses...."

Probable Cause

The flash programmer was able to connect with a Nios II JTAG debug module in the FPGA, but it could not successfully find an EPCS device located at the specified base address.

Suggested Actions

- Reconfigure the FPGA with a valid target design via JTAG using the Quartus II programmer. If the FPGA is configured by another method, such as by a configuration controller, the pins that connect to the EPCS device might be disabled.
- If you are using **nios2-flash-programmer** in command-line mode, make sure you specified the correct base address for your EPCS device. You can find the flash memory's base address in SOPC Builder.
- Ensure that the EPCS device is correctly connected to the FPGA on the board. Verify this by running the "Test EPCS" routine in the "Memory Test" software template in Nios II IDE. If the test fails, there is a problem with your memory connection. There are two places to look for the problem:
  - The physical connection on your target board
  - The pin assignments on the top-level FPGA design
- Use the Quartus II Programmer to program the EPCS device directly via a JTAG download cable, and verify that the EPCS device successfully configures the FPGA.
- Run **nios2-flash-programmer** in command-line mode with the `--epcs` parameter. This displays information about the flash memory in the EPCS device. For further details, see Chapter 3, Using the Flash Programmer in Command-Line Mode.

"System does not have any flash memory" Error

When you run the flash programmer, you get the error: "The SOPC Builder system does not have any flash memory."

Probable Cause

The FPGA is not currently configured with a valid flash programmer target design.

Suggested Actions

If practical, upgrade your FPGA design to meet the criteria for a flash programmer target design. See "Flash Programmer Target Design" on page 1–3 for details.

"Reading System ID at address 0x<address>: FAIL" Error

When you run the flash programmer in IDE mode, you get the error: "Reading System ID at address 0x<address>: FAIL"

Probable Cause

The FPGA is not currently configured with the target design that corresponds to the system library project for the C/C++ application in the IDE.

Suggested Actions

Use the Quartus II Programmer to download the correct FPGA configuration file to the FPGA, then try using the Nios II flash programmer again.