Nios® II Flash Programmer User Guide

Online Version
Send Feedback

UG-20103

ID: 683118
Version: 2017.11.06

Contents

1. Overview of the Nios® II Flash Programmer

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 Intel® Quartus® Prime Programmer . Its purpose is to program data into a flash memory device connected to an FPGA. The Nios II Flash Programmer sends file contents over a download cable, such as theIntel FPGA Download Cable, to a Nios II system running on the FPGA, and instructs the Nios II system to write the data to flash memory.

The Nios II Flash Programmer can program three types of content to flash memory:

- Nios II software executable files—Many systems use flash memory to store non-volatile program code, or firmware. Nios II systems can boot from 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.

The Nios II Flash Programmer can be used to program the following types of memory:

- Common flash interface (CFI)-compliant flash memory—CFI is an industry standard that provides a common, vendor-independent interface to flash memory devices.

  For more information about the CFI specification, refer to the JEDEC Common Flash Interface standard JESD68.01 and JEDEC publications JEP137x, available on the JEDEC Solid State Technology Association standards organization website (www.jedec.org).
- Intel EPCQ, EPCQL, and EPCQA serial configuration devices store FPGA configuration data and Nios II executable software.

The Nios II processor supports the following two boot options using Intel Serial Flash:

- The Nios II processor application executes in place from EPCQ flash.
- The Nios II processor application is copied from EPCQ flash to RAM using a boot copier.

You can boot a Nios II processor from Intel EPCQ flash memory (EPCQx1, EPCQx4) using an Intel FPGA Serial Flash Controller. The Intel FPGA Serial Flash Controller with Avalon® interface allows Nios II processor systems to access an Intel EPCQ flash memory, which supports standard, quad and single-I/O mode. The Nios II processor Software Build Tools (SBT) supports the Nios II booting from the Intel FPGA Serial Flash Controller.

For more information about how to build a bootable system from EPCS and device eram for a Nios II processor application, refer to *Nios II Configuration and Booting Solutions* chapter of *Embedded Design Handbook*.

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

Related Information

Embedded Design Handbook

1.1. Prerequisites

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

- *Nios II Hardware Development Tutorial*
- "Getting Started with the Graphical User Interface" chapter of the *Nios II Gen2 Software Developer's Handbook*.

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.

For more information, refer to *AN346:Using the Nios II Configuration Controller Reference Designs*, or to the reference manual for your specific development board.

Related Information

- Nios II Gen2 Hardware Development Tutorial
- Nios II Gen2 Software Developer's Handbook
- Using the Nios II Configuration Controller Reference Designs

1.2. Nios II Flash Programmer GUI and Command-Line Utilities

You can run the Nios II Flash Programmer from a GUI or from the command line. The GUI displays the command-line equivalents of the actions you direct it to perform.

For more information about the flash programmer GUI, refer to "Using the Flash Programmer GUI" chapter.

For more information about using the flash programmer command-line utilities, refer to "Using the Flash Programmer from the Command Line" chapter.

The following tools allow you to run the Nios II Flash Programmer:

- Nios II Software Build Tools for Eclipse™ – The Nios II Software Build Tools (SBT) for Eclipse provides easy access to the Nios II Flash Programmer GUI. The flash programmer GUI is an easy-to-use interface that allows you to control the flash programmer features using settings you can store and reuse. This access method is suitable for most flash programming needs.
- Nios II Command Shell – The Nios II Command Shell provides commands that control the flash programmer features. You might have to calculate some parameters, manually. You can also start the Nios II Flash Programmer GUI from the command line.

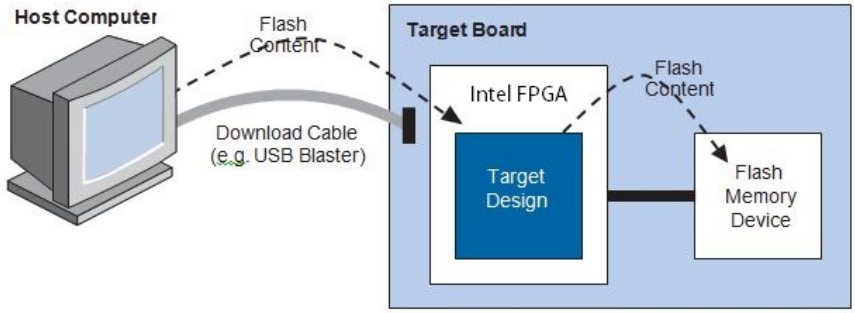
Related Information

- Using the Flash Programmer GUI on page 9
- Using the Flash Programmer from the Command Line on page 16

1.3. How the Flash Programmer Works

The Nios II Flash Programmer has two parts, the host and the target, as shown in the following figure. The host portion runs on your computer. It sends flash programming files and programming instructions over a download cable to the target. The target portion is a hardware design, running in the FPGA. The target portion accepts the programming data—flash content and required information about the target flash memory device—sent by the host, and follows the instructions to write data to the flash memory device.

Figure 1. How the Nios II Flash Programmer Works



1.3.1. Flash Programmer Target Design

To use the Nios II Flash Programmer, you must have a valid flash programmer target design downloaded to your board. A valid target design contains a Platform Designer system with at least the Platform Designer components shown in the "Minimum Component Set for the Flash Programmer Target Design" table.

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. "Minimum Component Set for the Flash Programmer Target Design" lists the minimum component set for programming each kind of flash memory.

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

Component	Flash Memory to Program			
	CFI	EPCS/EPCQx1	EPCQ	On-Chip Flash (Intel MAX® 10)
Nios II processor, with JTAG debug module level1 or greater	Required	Required	Required	Required
System ID peripheral	Recommended	Recommended(1)	Recommended(1)	Optional
Generic Tri-State Controller	Required			
Tristate Conduit Bridge	Required(3)			
Legacy EPCS/EPCQx1 Flash Controller		Required		
Intel FPGA Generic Quad SPI Controller			Required(4)	
Intel FPGA Generic Quad SPI Controller II			Required(4)	
Intel FPGA Serial Flash Controller			Required(4)	
Intel FPGA Serial Flash Controller II			Required(4)	
Intel On-Chip Flash				Required

(1) If present, a System ID Peripheral component allows the Nios II Flash Programmer to validate the target design before programming the flash memory.

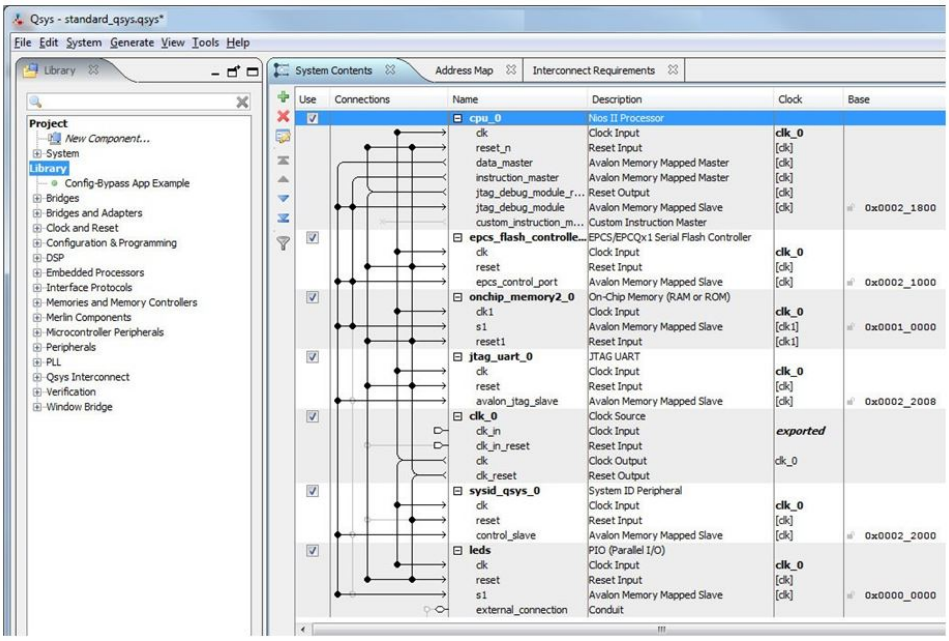
(2) A Nios II system can interface with more than one CFI flash memory device. The system must contain one Flash Memory (Common Flash Interface) component for each flash memory device on the board.

(3) Tri-state Conduit Bridge is needed to drive the CFI flash memory signals.

(4) Only one of these components is required. Use these controllers to communicate with an EPCQ device in ASx4 mode.

Figure 2. Example Target Design Containing the Minimum Component Set

This is an example of a Platform Designer system with the minimum component set for a system composed of an EPCS serial configuration device. The system also includes other components which relate to the purpose of the system, not to the flash programmer.



Note: The data master and instruction master of the Nios II Processor must be connected to the same address space as the On-Chip memory.

Hardware example designs capable of programming the flash memory are provided with development boards. If you are developing for a custom board, consider using one of these example designs as a starting point for your flash programmer target design.

1.4. Document Revision History for Overview of the Nios II Flash Programmer

Table 2. Document Revision History for Overview of the Nios II Flash Programmer

Date	Version	Changes
November 2017	2017.11.06	• Added the component support for Intel FPGA Generic Quad SPI Controller, Intel FPGA Generic Quad SPI Controller II, Intel FPGA Serial Flash Controller, and Intel FPGA Serial Flash Controller II in *Table: Minimum Component Set for the Flash Programmer Target Design*.
May 2017	2017.05.08	Updated: • Table 1 on page 7
September 2015	2015.09.21	Maintenance release.
July 2015	2015.07.01	• Added info for how to build a bootable .pof file including Nios II code for boot from EPCS or device eram.

2. Using the Flash Programmer GUI

Note: The GUI uses the legacy `nios2-flash-programmer` tool instead of `quartus_pgm --nios 2`. Intel recommends you to use the Quartus Prime Programmer.

The Nios II Flash Programmer GUI, available in Intel Quartus Prime Standard Edition, is an easy-to-use graphical interface that allows you to:

- automate the process of programming flash memory.
- control the programming parameters
- program any combination of software, hardware, and binary data in flash memory in one operation.
- generate flash files for your future use, and store them without programming the flash memory. Generating flash files or programming flash memory from the flash programmer GUI generates a script for future use from the command line.

You start the Nios II Flash Programmer GUI from the Nios II SBT for Eclipse or from the command line.

Alternatively, you can use the flash programmer from the command line. *Using the Flash Programmer from the Command Line* chapter describes the flash programmer command-line utilities.

Note: It is recommended that you use the Nios II Flash Programmer GUI to generate automated scripts, and use the scripts to automate the Nios II flash programming process.

Related Information

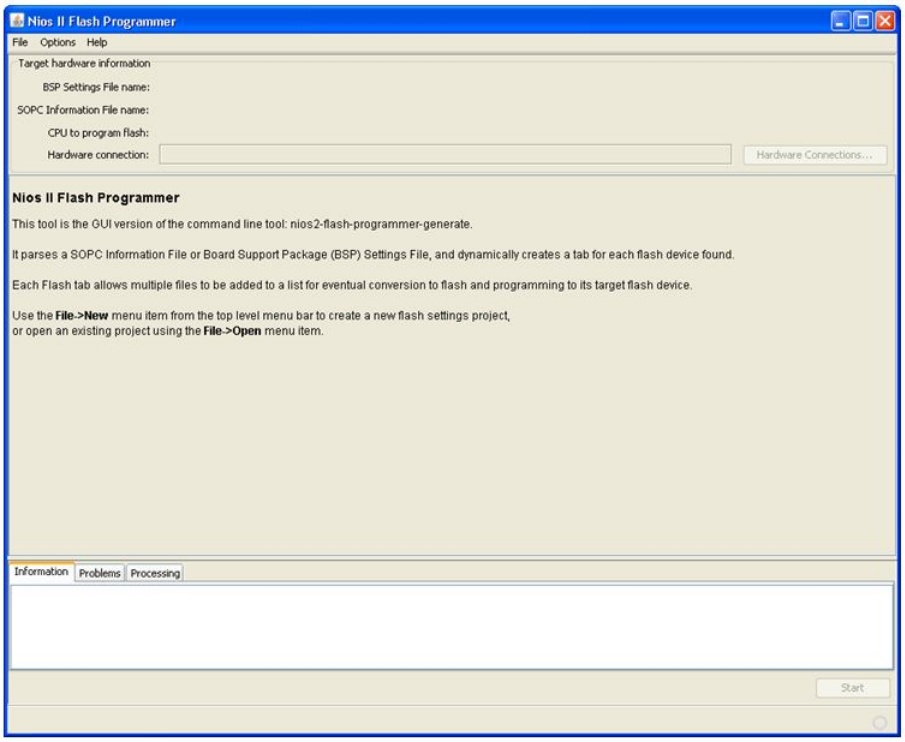
- Nios II Flash Programmer GUI and Command-Line Utilities on page 5
- Using the Flash Programmer from the Command Line on page 16
- Nios II Gen2 Software Developer's Handbook
  For more information about the Nios II Command Shell, refer to the "Getting Started from the Command Line" chapter in the *Nios II Gen2 Software Developer's Handbook*.

2.1. Starting the Flash Programmer GUI

There are two ways to start the Nios II Flash Programmer GUI:

- From the Nios II Command Shell:
  - type `nios2-flash-programmer-gui r`
- From the Nios II SBT for Eclipse:
  - On the **Nios II** menu, click **Flash Programmer** and the **Flash Programmer** dialog box appears.

Figure 3. Flash Programmer Dialog Box



2.2. Specifying your Flash Programmer Settings

Before writing data to flash memory, you must determine the flash programmer settings.

To create a new set of flash programmer settings, complete the following steps:

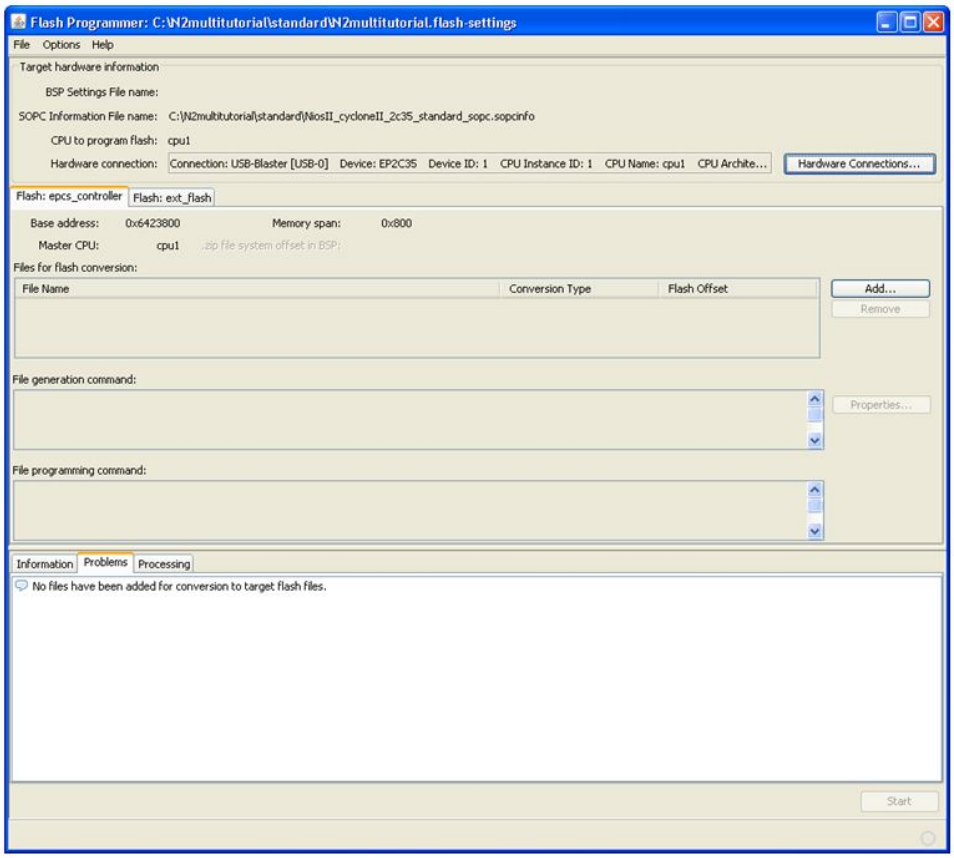
1. On the **File** menu, click **New**. The **New Flash Programmer Settings File** dialog box appears.
2. Select **Get flash programmer system details from BSP Settings File** or **Get flash programmer system details from SOPC Information File**.
3. Browse to locate your BSP Settings File (**.bsp**) or SOPC Information File (**.sopcinfo**).
4. For a multiprocessor system, select the processor. If you specify a **.bsp** file, the processor is already specified.
5. Click **OK**. The **New Flash Programmer Settings File** dialog box closes and the Nios II Flash Programmer GUI populates with your processor selection, if relevant, and the information from the **.bsp** or **.sopcinfo** file.

Depending on your selection in step 2, your flash programmer settings may not include information about a BSP settings file. A new set of flash programmer settings based on a BSP settings file includes the **.sopcinfo** file name, but a new set of flash programmer settings based on a **.sopcinfo** file does not have information about a BSP settings file to which it corresponds. In that case you must identify the **.bsp** file

explicitly. The **.bsp** file contains the information about your processor selection, for example, whereas the **.sopcinfo** file knows about the available processors, but not which one you selected.

Figure 4. Flash Programmer Dialog Box with Flash Programmer Settings

This figure shows the flash programmer with a new set of flash programmer settings based on the **.sopcinfo** file for a design with multiple processors and two flash memory components.



2.3. Working with Flash Programmer Settings Files

The Nios II Flash Programmer GUI is a tool with many options. It is recommended saving your flash programmer settings in a Nios II Flash Programmer Settings File (**.flash-settings**) for future use.

The **.flash-settings** file includes the settings described in the following sections, such as the script and flash files directory paths, whether to perform system ID and system timestamp checking, and whether to generate flash files or program flash memory.

To save your current flash programmer settings, on the **File** menu, click **Save** or **SaveAs** to update or create a **.flash-settings** file. After you save the file, you can continue to edit it in the Nios II Flash Programmer GUI.

To open a pre-existing flash programmer settings file, on the **File** menu, click **Open**, and navigate to the location of the existing **.flash-settings** file.

2.4. Setting the Hardware Connection

This section describes how to select the correct download cable, device, and processor to program flash memory. If your system has only a single download cable and a single processor, the process is simple. This section describes all the steps for a system with multiple download cables, processors, and devices.

Before you can program flash memory on your board, you must configure your FPGA with a flash programmer target design that contains at least the minimum component set specified in the "Minimum Component Set for the Flash Programmer Target Design" table in the "Flash Programmer Target Design" chapter.

Note: For instructions to configure the FPGA, refer to the "Intel Quartus Prime Programmer" chapter in volume 3 of the *Intel Quartus Prime Handbook*.

After you load the target design on your FPGA, you can set the hardware connection for programming flash memory.

To set the **Hardware Connection**, perform the following steps:

1. Click **Hardware Connections**. The **Hardware Connections** dialog box appears.
2. In the **Hardware Connections** dialog box, click **Refresh Connections**.
3. If you are reusing an **.flash-settings** file, and the Intel Quartus Prime project has been recompiled since the **.flash-settings** file was created or the **Name** column entries in the **Processors** list are blank, perform the following steps:
   a. Under **JTAG Debugging Information File name**, browse to locate your project JTAG Debugging Information File (**.jdi**).
   b. Click **Resolve Names**. The flash programmer uses the **.jdi** file to ensure the available connection information is accurate.
4. If your design has multiple download cables, select the appropriate cable.
5. If your design has multiple processors, select the Nios II processor that corresponds to the **CPU to program flash** value under **Target hardware information** in the **Nios II Flash Programmer** dialog box.
6. Click **Close**.

Related Information

- Quartus Prime Handbook Volume 3: Verification
- Flash Programmer Target Design on page 6

2.5. Checking System ID and System Timestamp

If your flash programmer target design includes a System ID component, the Nios II Flash Programmer can perform system ID and system timestamp checking before programming flash memory. If the flash programmer performs system ID checking, system timestamp checking, or both, and the expected system is not configured in the FPGA, the flash programmer does not program the flash memory.

Note: The system ID and system timestamp checking are enabled by default. Intel recommends you keep these settings and that your FPGA target design include a system ID component.

To disable checking for system ID or system timestamp, perform the following steps:

1. Click **Hardware Connections**. The **Hardware Connections** dialog box appears.
2. To disable system ID checking, turn on **Ignore mismatched system ID**.
3. To disable system timestamp checking, turn on **Ignore mismatched system timestamp**.
4. Click **Close**.

After the hardware connections are set, you can confirm system ID and system timestamp matching by performing the following steps:

1. Click **Hardware Connections**.
2. In the **Hardware Connections** dialog box, click **System ID Properties**.
3. Check that the **Expected system ID** and **Actual system ID** values match, and that the **Expected system timestamp** and **Actual system timestamp** values match.
4. In the **System ID Properties** dialog box, click **Close**.
5. In the **Hardware Connections** dialog box, click **Close**.

For additional information about the system ID and system timestamp, refer to the "General Parameters" table in the "quartus_pgm --nios 2 Parameters" section.

Regardless of the System ID component values, you cannot program flash memory if the hardware design configured in the FPGA is not a valid flash programmer target design that contains at least the minimum component set. Refer to *Table: Minimum Component Set for the Flash Programmer Target Design* in the *Flash Programmer Target Design chapter*.

Related Information

quartus_pgm --nios2 Parameters on page 17

2.6. Generating Flash Files and Programming Flash Memory

The Nios II Flash Programmer can generate flash files, program the flash memory with a flash file, or both. The flash programmer can generate flash files from the following different file types:

- SRAM Object File (**.sof**) — Contains FPGA configuration data
- Executable and Linking Format File (**.elf**)— Contains your executable application software
- Intel FPGA Zip Read-Only File System File (**.zip**) — Contains a read-only zip file system associated with your Nios II software application project
- An arbitrary binary file

The Nios II EDS provides the Intel FPGA Zip Read-Only File System software component, which is an easy-to-use tool for storing and accessing data in flash memory. Depending on your application, you might find it more convenient to use the Zip Read-Only File System, rather than storing raw binary data in flash memory.

For information about the Intel FPGA Zip Read-Only File System, refer to the "Read-Only Zip File System" chapter in the *Nios II Gen2 Software Developer's Handbook*.

To generate flash files or to write to flash memory using your flash programmer settings, perform the following steps:

1. On the **Options** menu, turn on the actions you wish to perform. The following actions are available:
   - **Generate Files**—Generates flash files.
   - **Program Files**—Programs flash memory with the generated flash files.
   - **Erase Flash Before Programming**—Erases the entire flash memory before writing each flash file to it.
   - **Run From Reset After Programming**—Runs the processor from its reset vector after flash memory programming is complete.(5)
2. To specify the directories where you want the flash programmer to store the generated flash files and script, click **Staging Directories** in the **Options** menu and specify the script and flash-files directory paths.
3. On the tab for the flash memory device you wish to target, under **Files for flash conversion**, click **Add**.

   The Nios II Flash Programmer adds the file to the **Files for flash conversion** list and derives the flash offset from the file. Depending on your current **Options** settings, the **File generation command** and **File programming command** boxes populate with the command-line commands that generate the **.flash** file and program it to your flash memory, respectively, providing a convenient way to learn about the command-line utilities.
4. If you want to pass any additional arguments to the file generation command, under **File generation command**, click **Properties**.
5. If necessary, edit the **Conversion Type** settings for the files listed in the **Files for flash conversion table**, by clicking the relevant table cells. Allowed conversion types are **ELF**, **SOF**, and **BINARY**.
6. If necessary, edit the **Flash Offset** settings for the files listed in the **Files for flash conversion** table, by double-clicking the relevant table cells.

   Note: Do not specify a flash offset for files of conversion type **ELF**.
7. Click **Start**. The Nios II Flash Programmer performs the actions specified by the commands. It stores the generated flash files to the specified flash directory, and captures the commands it runs to the **flash_programmer.sh** bash shell script in the specified script directory.

   Note: You can use this script to duplicate the same actions again in the future, and to learn about the command-line utilities.

Related Information

Nios II Gen2 Software Developer's Handbook

For information about the Intel FPGA Zip Read-Only File System, refer to the "Read-Only Zip File System" chapter in the *Nios II Gen2 Software Developer's Handbook*.

(5) Relevant only if you turn on **Program Files**.

2.7. Document Revision History for Using the Flash Programmer GUI

Table 3. Document Revision History for Using the Flash Programmer GUI

Date	Version	Changes
September 2015	2015.09.21	Maintenance release.

3. Using the Flash Programmer from the Command Line

The Nios II development tools provide 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.

Alternatively, you can use the flash programmer GUI.

For more information about the flash programmer GUI, refer to the "Using the Flash Programmer GUI" chapter.

The Nios II Flash Programmer GUI programs flash memory 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. The GUI-generated script is particularly helpful if you need to use the `--instance` parameter listed in the **xref** table.

After you successfully program flash memory using the Nios II Flash Programmer GUI, you can find the flash programmer script in the directory you specified in the **Staging Directories** dialog box, available on the **Options** menu. The flash programmer script is a file named **flash_programmer.sh**.

Table 4. Flash Programmer Command-Line Utilities

Command-Line Utility	Description
`quartus_pgm --nios2`	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 SRAM Object File (**.sof**) to an S-record file.
`elf2flash`	Converts a Nios II Executable and Linking Format File (**.elf**) to an S-record file.
`bin2flash`	Converts an arbitrary data file to an S-record file.

The main utility for programming flash memory from the command line is `quartus_pgm --nios2`. It requires industry-standard S-record input files. These utilities ensure that the input is compatible with the Nios II Flash Programmer. Input file names for all utilities must include an explicit extension, such as **.elf** or **.flash**.

When you launch the Nios II Command Shell, in a Windows® operating system, the flash programmer utilities are available in your default search path.

For more information about the Nios II Command Shell, refer to the "Getting Started from the Command Line" chapter of the *Nios II Software Developer's Handbook*.

The following sections list the utilities and their functions.

Related Information

- Nios II Flash Programmer GUI and Command-Line Utilities on page 5
- Using the Flash Programmer GUI on page 9
- Nios II Gen2 Software Developer's Handbook
  For more information about the Nios II Command Shell, refer to the "Getting Started from the Command Line" chapter in the *Nios II Gen2 Software Developer's Handbook*.

3.1. quartus_pgm --nios2

The **quartus_pgm --nios2** 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**.

quartus_pgm --nios2 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 Flash Programmer GUI creates flash programmer files with a **.flash** extension.

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

The **quartus_pgm --nios2** command-line syntax is as follows:

```
quartus_pgm --nios2 [--help] [--cable=<cable name>]
[--device=<device index>] [--instance=<instance>]
[--sidp=<address>] [--id=<id>] [--timestamp=<time>] [--accept-bad-sysid] --
base=<address> [--epcs | --epcq | --onchip] <file> [--go]
```

Note: Before you can program flash memory on your board, you must configure your FPGA with a flash programmer target design. The design must contains at least the minimum component set specified in the *Table: Minimum Component Set for the Flash Programmer Target Design* in the *Flash Programmer Target Design chapter*.

Related Information

- Using the Flash Programmer GUI on page 9
- Flash Programmer Target Design on page 6
- Quartus Prime Handbook, Volume 3: Verification
  For instructions on how to configure the FPGA, refer to the "Quartus Prime Programmer" chapter in volume 3 of the *Quartus Prime Handbook*.

3.1.1. quartus_pgm --nios2 Parameters

The following tables contain the General, CFI and EPCS parameters for **quartus_pgm --nios2**.

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

3.1.1.1. General Parameters

Table 5. General Parameters

Name	Required	Description
`--cable=<cable name>`	Required if there are multiple download cables connected to the host computer.	Specifies which download cable to use. (6)
`--device=<device index>`	Required if there are multiple devices in the JTAG chain.	Specifies the FPGA's device number in the JTAG chain. The device index specifies 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.(7)
`--instance=<instance>`	Required if there are multiple Nios II processors with JTAG debug modules in the target design on the FPGA.	Specifies which Nios II JTAG debug module to look at in the FPGA. The instance ID specifies the JTAG debug module that is used for programming flash memory.(8)
`--sidp=<address>`	Optional; required for system ID validation.	Contains the base address of the System ID component in your system. This value is in hexadecimal format (for example, `0x01000000`).(9)
`--id=<id>`	Optional; required for system ID validation.	Contains the ID value programmed into the System ID component in your system. This value is randomly selected each time you regenerate your Platform Designer system. This value is in unsigned decimal format (for example, `2056847728u`).(10)
`--timestamp=<time>`	Optional; required for system timestamp validation.	Contains the timestamp value programmed into the System ID component in your system. Platform Designer sets this value based on the time of system generation. This value is in unsigned decimal format (for example, `1177105077u`). Turning this parameter on is the same as turning off the **Ignore mismatched system**

continued...

(6) The `--cable` parameter is only needed if there are multiple download cables connected to the host computer. To determine the cable names, run `jtagconfig`.

(7) The `--device` parameter is only needed if there are two or more processors in different devices with the same instance ID. To determine the JTAG device index, run `jtagconfig`.

(8) There are two ways to find the correct value of the instance ID for a processor. The easiest is to use the Nios II Flash Programmer GUI to create a sample flash programmer script. For more information, refer to the "Using the Flash Programmer GUI" chapter. Alternatively, open **<Intel Quartus Prime project name>.jdi**, in the Intel Quartus Prime project directory. Locate the Nios II processor node by finding a value of `hpath` containing "<processor module name>=". The instance ID is specified as `instance_id`.

(9) In **system.h** and in your board support package (BSP), the system ID base address is specified by SYSID_BASE.

(10) In **system.h** and in your BSP the system ID value is specified by SYSID_ID.

Name	Required	Description
		timestamp check box in the Nios II Flash Programmer GUI **Hardware Connections** dialog box.(11)
`--accept-bad-sysid`	Optional; defaults off.	Bypasses the system ID validation. Forces the flash programmer to download a flash image. Turning this parameter on is the same as turning on the **Ignore mismatched system ID** check box in the Nios II Flash Programmer GUI **Hardware Connections** dialog box.
`--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.
`--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 processor from its reset vector after flash memory programming is complete.

(11) In **system.h** and in your BSP, the system ID time stamp is specified by **SYSID_TIMESTAMP**.

3.1.1.2. Device Parameters

Table 6. Device Parameters

Name	Required	Description
`--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 parameter is the absolute address in the target design's address space. **quartus_pgm --nios2** treats addresses in the S-record files as offsets to the base address.
`--epcs | --epcq`	Required when programming an EPCS (`--epcs`)/EPCQ/EPCQL/EPCQA (`--epcq`) serial configuration device; defaults off.	Specifies that the target flash memory is an EPCS/EPCQ/EPCQL/EPCQA serial configuration device.
`--debug`	Optional; defaults off.	Prints debug information about the physical memory inside the EPCS/EPCQ device.
`--base=<address>`	Yes	Specifies the base address of the EPCS/EPCQ device.
`--onchip`	Yes	Specifies that the target flash memory is on-chip flash (MAX 10 only).
`--override=<file>`	Yes	Specifies the override file.
`--width=<width>`	Yes	Specifies the data-width override.
`--dualdie`	Yes	Specifies that the target flash memory is dual-die.

3.1.2. quartus_pgm --nios2 Command-Line Examples

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

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

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

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

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

```
quartus_pgm --nios2 --base=0x200000 --read=current.srec \ --read-
bytes=0,0x10000
```

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

```
quartus_pgm --nios2 --base=0x200000 --erase=0x8000,0x10000
```

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

```
quartus_pgm --nios2 --base=0x200000 --debug
```

3.2. sof2flash

The **sof2flash** utility takes an SRAM Object File and translates it to an S-record file, suitable for programming into flash memory.

3.2.1. sof2flash Parameters

The following tables contain the General, CFI, EPCS, and Device-specific parameters for **sof2flash**.

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

3.2.1.1. General Parameters

Table 7. General Parameters

Name	Required	Description
`--compress`	Optional	Turns on compression. Available for Cyclone® II, Cyclone III, Stratix® II, and Stratix III devices.
`--input=<file>`	Required	Name of the input SRAM Object File.
`--output=<file>`	Required	Name of the output file.

3.2.1.2. Device Parameters

Table 8. Device Parameters

Name	Required	Description
`--offset=<addr>`	Required	Offset within the flash memory device where the FPGA configuration is to be programmed.
`--epcs`	Required for EPCS devices; defaults off.	Offset within the flash memory device where the FPGA configuration is to be programmed.
`--epcq`	Required for EPCQ devices of size 256 Mb.	Specifies that the output is intended for an EPCQ device of size 256 Mb.
`--epcq128`	Required for EPCQ devices of size 128 Mb or less.	Specifies that the output is intended for an EPCQ device of size 128 Mb or less.

(12) You must add the `--epcq128` switch to allow user to specify that they have a 128 Mb flash or smaller.

3.2.1.3. Configuration-Specific Parameters

Table 9. Device-Specific Parameters

Name	Required	Description
`--activeparallel`	Optional	Creates parallel flash contents compatible with active-parallel configuration mode. Only available on FPGAs which support active-parallel configuration.
`--pfl`	Required if your FPGA configuration uses the PFL	Specifies that the flash programmer use the parallel flash loader. Required if your FPGA configuration uses the parallel flash loader (PFL).
`--optionbit=<optionbitaddr>`[13]	Required if your FPGA configuration uses the PFL	Specifies the option bit address in your flash memory device. When you use this option, the sof2flash command generates both a **.flash** file and a **.map.flash** file. When you program the flash memory with the **.map.flash** file, it overwrites the default option bits. In almost all cases, the default option bits are appropriate and you should not program this file to flash.

Related Information

AN386: Using the Parallel Flash Loader with the Quartus

3.2.2. sof2flash Command-Line Examples

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

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

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

- Converts standard.sof to an S-record file named standard_epcq.flash intended for an EPCQ device.

```
sof2flash --epcq --input=standard.sof --output=standard_epcq.flash
```

- Converts **standard.sof** to an S-record file named **standard.flash** for use with the parallel flash loader, and generates an option-bits overwrite file **standard.map.flash** for option bits at offset `0x18000` on the flash memory device.

```
sof2flash --optionbit=0x18000 --pfl --input=standard.sof \ --
output=standard.flash --offset=0x640000
```

(13) Using the `--pfl` and `--optionbits` command-line options slows down sof2flash generation noticeably. For more information about when to use the `--pfl` and `--optionbits` command-line options, refer to *AN386: Using the Parallel Flash Loader with the Intel Quartus Prime Software*.

Note: It is recommended that you not use the **standard.map.flash** file even if you generate it. The `--optionbit` command-line option is required for correct functioning of the --pfl option, but the resulting overwrite file should be ignored.

3.3. elf2flash

The **elf2flash** utility takes an Executable and Linking Format File, 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 processor'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 executable and linking format 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.

3.3.1. elf2flash Parameters

The following tables contain the General, CFI, and EPCS parameters for **elf2flash**.

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

3.3.1.1. General Parameters

Table 10. General Parameters

Name	Required	Description
`--input=<file>`	Required	The name of the input Executable and Linking Format File.
`--output=<file>`	Required	The name of the output file.

3.3.1.2. CFI Parameters

Table 11. CFI Parameters

Name	Required	Description
`--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 processor reset address, which is specified in Platform Designer. **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 processor'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 boot copier resides at *<Nios II EDS install path>*/**components/Intel FPGA_nios2/boot_loader_cfi.srec**.

3.3.1.3. EPCS Parameters

Table 12. EPCS Parameters

Name	Required	Description
`--epcs`	Required when creating files for an EPCS/EPCQ device; defaults off.	Specifies that the output is intended for an EPCS/EPCQ device.
`--after=<file>`	Required when programming both hardware and software into an EPCS/EPCQ device.	**elf2flash** uses this parameter to position a Nios II executable in an EPCS/EPCQ device along with an FPGA configuration. For more information, refer to the "Programming Both Hardware and Software into an EPCS/EPCQ Device".

3.3.2. Programming Both Hardware and Software into an EPCS/EPCQ Device

The `--base` parameter is not available for EPCS/EPCQ devices, because in EPCS/EPCQ 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/EPCQ 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. Refer to the second example under the "elf2flash Command-Line Examples" chapter.

Note: **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.

Note: In Intel Quartus Prime software version 13.1 and onwards, the `-epcs/--epcq` option in sof2flash generates .flash file with a SOF header, which contains the SOF length. This change is required for V-series devices and above for new SOF format, and to allow for future SOF format variations. The Nios II bootcopier loads the Nios II software executable from EPCS/EPCQ devices based on the SOF length. For more information about how to program EPCS/EPCQ devices, refer to the "KDB Solution rd11192013_118" webpage.

Related Information

- elf2flash Command-Line Examples on page 25
- KDB Solution rd11192013_118

3.3.3. elf2flash Command-Line Examples

- Converts **myapp.elf** to an S-record file named **myapp.flash**, intended for a CFI flash memory based at `0x0`. Includes a boot copier (from **boot_loader_cfi.srec**), which is required in this example because --base and --reset are equal.

```
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 an EPCS device. The S-record output starts at the first address unused by **standard.flash**.

```
elf2flash --epcs --after=standard.flash --input=myapp.elf \ --
output=myapp.flash
```

3.4. 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 Intel FPGA Read-Only Zip Filing System, which serves the same purpose.

Note: 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**.

Related Information

- Nios II Classic Software Developer's Handbook
  For more information about the Intel FPGA Zip Read-Only File System, refer to the "Read-Only Zip File System" chapter in the *Nios II Classic Software Developer's Handbook*.
- Nios II Gen2 Software Developer's Handbook
  For more information about the Intel FPGA Zip Read-Only File System, refer to the "Read-Only Zip File System" chapter in the *Nios II Gen2 Software Developer's Handbook*.

3.4.1. bin2flash Parameters

The following tables contain the General, CFI, EPCS, and Device-specific parameters for **bin2flash**.

For additional parameters, type `bin2flash --help` at a command line.

Table 13. bin2flash Parameters

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

3.4.2. bin2flash Command-Line Example

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

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

3.5. Document Revision History for Using the Flash Programmer from the Command Line

Table 14. Document Revision History for Using the Flash Programmer from the Command Line

Date	Version	Changes
November 2017	2017.11.06	Merged *CFI Parameters* and *EPCS Parameters* into *Device Parameters*.
May 2017	2017.05.08	• `nios2-flash-programmer` is now `quartus_pgm --nios2` • Table 6 on page 20
September 2015	2015.09.21	Updated the EPCS parameters section to better define EPCQ256 and EPCQ128.

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 must refer to the data sheet for the flash memory device you are using. Ensure you fully understand the CFI aspects of the device.

Some CFI flash memory devices contain missing or incorrect CFI table information. In such cases, 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

A.1. 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 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 overrides the CFI table, the flash programmer displays a message "Override data for this device is built in".

Related Information

Using the Flash Programmer from the Command Line on page 16
For more information on using the flash programmer from the command line, refer to Chapter 3, Using the Flash Programmer from the Command Line.

A.2. 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 **quartus_pgm --nios2** from the command line with the `--debug` parameter, which lists the CFI table found in the device. Compare the debug output with the device's data sheet.

A.2.1. 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 quartus_pgm --nios2 --debug
CFI[0x13] = 0x02 # The primary command set, found at CFI table -
CFI[0x14] = 0x00 # addresses 0x13 and 0x14 are overridden to
# 0x02, 0x00.
CFI[0x2C] = 0x01 # The number of CFI Erase block regions, found at
# CFI table –address 0x2C is overridden to 0x1.
```

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

A.2.2. How to Use the Flash Override File

There are two ways to deploy flash override files:

- Place the override file in *<Nios II EDS install path>***/bin**. The Nios II Flash Programmer searches this directory for all filenames matching the pattern `nios2-flash-override*`. The flash programmer loads all these files as override files.
- Pass the override file to the flash programmer with the `--override` parameter. The following example illustrates this parameter:

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

A.3. Width Mode Override Parameter

The override procedure described in the "Flash Override Files" chapter assumes the Nios II 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.

Related Information

Flash Override Files on page 27

A.4. Document Revision History for Non-Standard Flash Memories

Table 15. Document Revision History for Non-Standard Flash Memories

Date	Version	Changes
September 2015	2015.09.21	Maintenance release.

B. Troubleshooting

B.1. Overview

This chapter lists troubleshooting tips for the Nios II Flash Programmer. Each section in this chapter describes a common issue you might run into when using the Nios II Flash Programmer.

B.2. Start Button Grayed Out in the Flash Programmer GUI

In the Nios II Flash Programmer GUI, even after a flash programmer configuration is opened, the **Start** button appears grayed out.

B.2.1. Probable Cause

You have not fully specified the required parameters for programming flash memory.

B.2.2. Suggested Actions

- Check the **Problems** tab for any information about what may be missing or incorrect.
- Ensure that your JTAG cable settings are correct. Specify the connection in the Hardware Connections dialog box, by clicking **Refresh Connections** and **Resolve Names**, and selecting the Nios II processor that matches the value in **CPU to program flash**.
- Ensure that you have selected a file to program to the flash memory, and that it appears with the correct **Conversion Type** and **Flash Offset** values.

B.3. "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."

B.3.1. Probable Cause

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

B.3.2. Suggested Actions

- Ensure that the FPGA is running a valid flash programmer target design. If not, you must configure the FPGA using the Intel Quartus Prime programmer. For more information, refer to the "Flash Programmer Target Design" chapter.
- If using the flash programmer from the command line, ensure you have specified the proper `--device`, `--cable`, and `--instance` parameter values. Refer to "Using the Flash Programmer from the Command Line" for details. For more information, refer to the "Using the Flash Programmer from the Command Line" chapter.

Related Information

- Flash Programmer Target Design on page 6
- Using the Flash Programmer from the Command Line on page 16

B.4. "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*>"

B.4.1. Probable Cause

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

B.4.2. Suggested Actions

- If you are using **quartus_pgm --nios2** from the command line, you must specify the correct base address for the CFI device. You can find the flash memory's base address in Platform Designer.
- Run **quartus_pgm --nios2** from the command line with the `--debug` parameter. This command dumps the flash memory's query table. Compare the output with the flash memory device's data sheet. For more information, refer to the "Using the Flash Programmer from the Command Line".
- Ensure your flash memory hardware is correctly connected to place it at the base address specified in Platform Designer. Verify the base address by running the "Test Flash" routine in the "Memory Test" software template provided in the Nios II EDS. 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, ensure the flash memory device you are using does not require an override file. For more information, refer to Appendix A, Non-Standard Flash Memories.

Related Information

- Using the Flash Programmer from the Command Line on page 16
- Non-Standard Flash Memories on page 27

B.5. "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...."

B.5.1. Probable Cause

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

B.5.2. Suggested Actions

- Reconfigure the FPGA with a valid target design via JTAG using the Intel Quartus Prime 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 **quartus_pgm --nios2** from the command line, ensure you specified the correct base address for your EPCS device. You can find the flash memory's base address in Platform Designer.
- Ensure that the EPCS device is correctly connected to the FPGA on the board. Verify the EPCS connection by running the "Test EPCS" routine in the "Memory Test" software template provided by the Nios II EDS. 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 Intel Quartus Prime Programmer to program the EPCS device directly via a JTAG download cable, and verify that the EPCS device successfully configures the FPGA.
- Run **quartus_pgm --nios2** from the command line with the `--epcs` parameter. This command displays information about the flash memory in the EPCS device. For more information, refer to the "Using the Flash Programmer from the Command Line" chapter.

Related Information

Using the Flash Programmer from the Command Line on page 16

B.6. "System Does Not Have Any Flash Memory" Error

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

B.6.1. Probable Cause

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

B.6.2. Suggested Actions

If practical, upgrade your FPGA design to meet the criteria for a flash programmer target design. For more information, refer to the "Flash Programmer Target Design" chapter.

Related Information

Flash Programmer Target Design on page 6

B.7. "Reading System ID at Address *0x<address>*: FAIL" Error

When you run the Nios II Flash Programmer GUI, you get the error: "Reading System ID at address *0x<address>*: FAIL"

B.7.1. Probable Cause

The FPGA is not currently configured with the target design that corresponds to the BSP project for the software application in the Nios II Software Build Tools for Eclipse.

B.7.2. Suggested Actions

Use the Intel Quartus Prime Programmer to download the correct FPGA configuration file to the FPGA, then try using the Nios II Flash Programmer again.

B.8. "Base Address Not Aligned on Size of Device" Error

When you run the flash programmer, you get the error message "Base address not aligned on size of device".

B.8.1. Probable Cause

The flash device base address being passed to the Flash Programmer is not a multiple of the flash device's size.

B.8.2. Suggested Actions

- Ensure that the flash device is mapped to a base address in Platform Designer that is a multiple of the flash size listed in its CFI table.
- If in command line mode, ensure that the `--base` parameter you are passing to **nios2-flash-programmer** is the correct base address of the flash device in your Platform Designer system.

B.9. Document Revision History for Troubleshooting

Table 16. Document Revision History for Troubleshooting

Date	Version	Changes
September 2015	2015.09.21	Maintenance release.

C. Document Revision History

Table 17. Document Revision History for Nios II Flash Programmer User Guide

Date	Version	Changes
November 2017	2017.11.06	• Added the component support for Intel FPGA Generic Quad SPI Controller, Intel FPGA Generic Quad SPI Controller II, Intel FPGA Serial Flash Controller, and Intel FPGA Serial Flash Controller II in *Table: Minimum Component Set for the Flash Programmer Target Design*. • Changed the document part number from **UG-NIOSIIFLSHPROG** to **UG-20103**.
May 2017	2017.05.08	• `nios2-flash-programmer` is now `quartus_pgm --nios2` • The Stand-Alone Mode chapter has been removed
March 2014	2.2	• In chapter 1, added cross-reference to the Quad-Serial Configuration (EPCQ) Devices Datasheet • In chapter 1, updated the "Minimum Component Set for the Flash Programmer Target Design" table and updated the table notes. • In chapter 1, updated the "Example Target Design Containing the Minimum Component Set" figure. • In chapter 3, updated the "**sof2flash** Parameters" table. • In chapter 3, added the EPCQ command line example. • In chapter 3, updated the "**elf2flash** Parameters" table. • In chapter 3, added a note about how to program EPCS/EPCQ devices. • Replaced SOPC Builder with Qsys. • Replaced "EPCS" with "EPCS/EPCQ."
February 2010	2.1	Documented the two **sof2flash** options `--pfl` and `--optionbit`.
January 2010	2.0	• Documented the new Nios II Flash Programmer GUI. • Revised entire document to use Nios II Software Build Tools for Eclipse instead of Nios II IDE. • Removed out-of-date list of tested flash memory devices.
May 2008	1.6	In chapter 3, corrected error in the "**sof2flash** Parameters" table. The default mode for the compression parameter is on. Compression is available for Cyclone II, Cyclone III, Stratix II, and Stratix III devices.
November 2007	1.5	• In chapter 1, added note that HardCopy® II devices also support programming CFI Flash using Nios II Flash programmer. • In chapter 3, documented command-line support for active parallel configuration.
May 2007	1.4	• In chapter 1, removed mention of board description files (no longer implemented) • In chapter 1, corrected and updated discussion of --instance and --device command line parameters • In chapter 1, updated SOPC Builder screen shot • In chapter 3, added descriptions of **nios2-flash-programmer** options `--sidp`, `--id`, `--timestamp`, `--accept-bad-sysid` • In appendix C, corrected missing installation step • In appendix C, removed mention of board description files (no longer implemented)

continued...

Date	Version	Changes
November 2006	1.3	Updates for the Nios II version 6.1 release. Includes improvements to the flash programmer user interface.
October 2005	1.2	Updates for the Nios II version 5.1 release. Includes major changes to the flash programmer target design.
December 2004	1.1	Updates for the Nios II version 1.1 release.
May 2004	1.0	Initial release.