Introduction

This chapter describes the Nios® II command-line tools that are provided with the Nios II Embedded Development Suite (EDS). The chapter describes both the Altera® tools and the GNU tools. Most of the commands are located in the **bin** and **sdk2** subdirectories of your Nios II EDS installation.

The Altera command line tools are useful for a range of activities, from board and system-level debugging to programming an FPGA configuration file (**.sof**). For these tools, the examples expand on the brief descriptions of the Altera-provided command-line tools for developing Nios II programs in "Altera-Provided Embedded Development Tools" in the *Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook*. The Nios II GCC toolchain contains the GNU Compiler Collection, GNU Binary Utilities (binutils), and newlib C library.

☞ All of the commands described in this chapter are available in the Nios II command shell. For most of the commands, you can obtain help in this shell by typing
<command name> `--help` ↵

To start the Nios II command shell on Windows platforms, on the Start menu, click **All Programs**. On the All Programs menu, on the Altera submenu, on the Nios II EDS *<version>* submenu, click **Nios II** *<version>* **Command Shell**.
On Linux platforms, type the following command:
<Nios II EDS install path>`/nios2_command_shell.sh`↵

The command shell is a Bourne-again shell (bash) with a pre-configured environment.

Altera Command-Line Tools for Board Bringup and Diagnostics

This section describes Altera command-line tools useful for Nios development board bringup and debugging.

jtagconfig

This command returns information about the devices connected to your host PC through the JTAG interface, for your use in debugging or programming. Use this command to determine if you configured your FPGA correctly.

Many of the other commands depend on successful JTAG connection. If you are unable to use other commands, check whether your JTAG chain differs from the simple, single-device chain used as an example in this chapter.

Type `jtagconfig --help` from a Nios II command shell to display a list of options and a brief usage statement.

jtagconfig Usage Example

To use the `jtagconfig` command, perform the following steps:

1. Open a Nios II command shell.

2. In the command shell, type the following command:
   jtagconfig -n ↵

Example 4–1 shows a typical system response to the `jtagconfig -n` command.

Example 4–1. jtagconfig Example Response

```
[SOPC Builder]$ jtagconfig -n
1) USB-Blaster [USB-0]
   020050DD   EP1S40/_HARDCOPY_FPGA_PROTOTYPE
      Node 11104600
      Node 0C006E00
```

The information in the response varies, depending on the particular FPGA, its configuration, and the JTAG connection cable type. Table 4–1 describes the information that appears in the response in Example 4–1.

Table 4–1. Interpretation of jtagconfig Command Response

Value	Description
USB-Blaster [USB-0]	The type of cable. You can have multiple cables connected to your workstation.
EP1S40/_HARDCOPY_FPGA_PROTOTYPE	The device name, as identified by silicon identification number.
Node 11104600	The node number of a JTAG node inside the FPGA. The appearance of a node number between 11104600 and 11046FF, inclusive, in this system's response confirms that you have a Nios II processor with a JTAG debug module.
Note 0C006E00	The node number of a JTAG node inside the FPGA. The appearance of a node number between 0C006E00 and 0C006EFF, inclusive, in this system's response confirms that you have a JTAG UART component.

The device name is read from the text file **pgm_parts.txt** in your Quartus® II installation. In Example 4–1, the name is EP1S40/_HARDCOPY_FPGA_PROTOTYPE because the silicon identification number on the JTAG chain for the FPGA device is 020050DD, which maps to the names EP1S40<*device-specific name*>, a couple of which end in the string _HARDCOPY_FPGA_PROTOTYPE. The internal nodes are nodes on the system-level debug (SLD) hub. All JTAG communication to an Altera FPGA passes through this hub, including advanced debugging capabilities such as the SignalTap® II embedded logic analyzer and the debugging capabilities in the Nios II EDS.

Example 4–1 illustrates a single cable connected to a single-device JTAG chain. However, your computer can have multiple JTAG cables, connected to different systems. Each of these systems can have multiple devices in its JTAG chain. Each device can have multiple JTAG debug modules, JTAG UART modules, and other kinds of JTAG nodes. Use the `jtagconfig -n` command to help you understand the devices with JTAG connections to your host PC and how you can access them.

nios2-configure-sof

This command downloads the specified **.sof** and configures the FPGA according to its contents. At a Nios II command shell prompt, type `nios2-configure-sof --help` for a list of available command-line options.

☞ You must specify the cable and device when you have more than one JTAG cable (USB-Blaster™ or ByteBlaster™ cable) connected to your computer or when you have more than one device (FPGA) in your JTAG chain. Use the `--cable` and `--device` options for this purpose.

nios2-configure-sof Usage Example

To use the `nios2-configure-sof` command, perform the following steps:

1. Open a Nios II command shell.

2. In the command shell, change to the directory in which your **.sof** is located. By default, the correct location is the top-level Quartus II project directory.

3. In the command shell, type the following command:
   `nios2-configure-sof` ↵

   The Nios II EDS searches the current directory for a **.sof** and programs it through the specified JTAG cable.

system-console

The `system-console` command starts a Tcl-based command shell that supports low-level JTAG chain verification and full system-level validation.This tool is available in the Nios II EDS starting in version 8.0.

This application is very helpful for low-level system debug, especially when bringing up a system. It provides a Tcl-based scripting environment and many features for testing your system.

The following important command-line options are available for the `system-console` command:

■ The `--script=<your script>.tcl` option directs the System Console to run your Tcl script.

■ The `--cli` option directs the System Console to open in your existing shell, rather than opening a new window.

■ The `--debug` option directs the System Console to redirect additional debug output to **stderr**.

■ The `--project-dir=`*`<project dir>`* option directs the System Console to the location of your hardware project. Ensure that you're working with the project you intend—the JTAG chain details and other information depend on the specific project.

■ The `--jdi=`*`<JDI file>`* option specifies the name-to-node mapping for the JTAG chain elements in your project.

For System Console usage examples and a comprehensive list of system console commands, refer to *Analyzing and Debugging Designs with the System Console* in volume 3 of the *Quartus II Handbook*. On-line training is available at http://www.altera.com/training.

Altera Command-Line Tools for Hardware Development

This section describes Altera command-line tools useful for hardware project development. They are useful for all projects created with SOPC Builder, whether or not the project includes a Nios II processor.

quartus_cmd and sopc_builder

These commands create scripts that automate generation of SOPC Builder systems and compilation of the corresponding Quartus II projects.

You can use these commands to create a flow that maintains only the minimum source files required to build your Quartus II project. If you copy an existing project to use as the basis for development of a new project, you should copy only this minimum set of source files. Similarly, when you check in files to your version control system, you want to check in only the minimum set required to reconstruct the project.

To reconstruct an SOPC Builder system, the following files are required:

■ *<project>*.**qpf** (Quartus II project file)

■ *<project>*.**qsf** (Quartus II settings file)

■ *<SOPC Builder system>*.**sopc** (SOPC Builder system description)

■ The additional HDL, BDF, or BSF files in your existing project

If you work with the hardware design examples that are provided with the Quartus II installation, Altera recommends that you copy each set of source files to a working directory to avoid modifying the original source files inadvertently. Run the script on the new working directory.

To create a flow that maintains only the minimum source files, perform the following steps:

1. Copy the required source files to a working directory, maintaining a correct copy of each source file elsewhere.

2. Change to this working directory.

3. To generate a **.sof** to configure your FPGA, type the following command sequence:

```
sopc_builder --no_splash –s --generate ↵
quartus_cmd <project>.qpf -c <project>.qsf ↵
```

The shell script in Example 4–2 illustrates these commands. This script automates the process of generating SOPC Builder systems and compiling Quartus II projects across any number of subdirectories. The script is an example only, and may require modification for your project. If you want to compile the Quartus II projects, set the COMPILE_QUARTUS variable in the script to 1.

Example 4–2. Script to Generate SOPC Builder System and Compile Quartus II Projects   (Part 1 of 2)

```
#!/bin/sh
COMPILE_QUARTUS=0
#
# Resolve TOP_LEVEL_DIR, default to PWD if no path provided.
#
if [ $# -eq 0 ]; then
    TOP_LEVEL_DIR=$PWD
else
    TOP_LEVEL_DIR=$1
fi
echo "TOP_LEVEL_DIR is $TOP_LEVEL_DIR"
echo
#
# Generate SOPC list...
#
SOPC_LIST=`find $TOP_LEVEL_DIR -name "*.sopc"`
#
# Generate Quartus II project list.
#
PROJ_LIST=`find $TOP_LEVEL_DIR -name "*.qpf" | sed s/\.qpf//g`
#
# Main body of the script.  First "generate" all of the SOPC Builder
# systems that are found, then compile the Quartus II projects.
#
#
# Run SOPC Builder to "generate" all of the systems that were found.
#
for SOPC_FN in $SOPC_LIST
do
    cd `dirname $SOPC_FN`
    if [ ! -e `basename $SOPC_FN .sopc`.vhd -a ! -e `basename $SOPC_FN .sopc`.v ]; then
        echo; echo
        echo "INFO:  Generating $SOPC_FN SOPC Builder system."
        sopc_builder -s --generate=1 --no_splash
        if [ $? -ne 4 ]; then
            echo; echo
            echo "ERROR:  SOPC Builder generation for $SOPC_FN has failed!!!"
            echo "ERROR:  Please check the SOPC file and data " \
                    "in the directory `dirname $SOPC_FN` for errors."
        fi
    else
        echo; echo
        echo "INFO:  HDL already exists for $SOPC_FN, skipping Generation!!!"
    fi
    cd $TOP_LEVEL_DIR
done
#
# Continued...
#
```

Example 4–2. Script to Generate SOPC Builder System and Compile Quartus II Projects   (Part 2 of 2)

```
#
# Now, generate all of the Quartus II projects that were found.
#
if [ $COMPILE_QUARTUS ]; then
    for PROJ in $PROJ_LIST
    do
        cd `dirname $PROJ`
        if [ ! -e `basename $PROJ`.sof ]; then
            echo; echo
            echo "INFO:  Compiling $PROJ Quartus II Project."
            quartus_cmd `basename $PROJ`.qpf -c `basename $PROJ`.qsf
            if [ $? -ne 4]; then
                echo; echo
                echo "ERROR:  Quartus II compilation for $PROJ has failed!!!."
                echo "ERROR:  Please check the Quartus II project " \
                          "in `dirname $PROJ` for details."
            fi
        else
            echo; echo
            echo "INFO:  SOF already exists for $PROJ, skipping compilation."
        fi
        cd $TOP_LEVEL_DIR
    done
fi
```

> ⚠ **CAUTION** The commands and script in Example 4–2 are provided for example purposes only. Altera does not guarantee the functionality for your particular use.

Altera Command-Line Tools for Flash Programming

This section describes the command-line tools for programming your Nios II-based design in flash memory.

When you use the Nios II EDS to program flash memory, the Nios II EDS generates a shell script that contains the flash conversion commands and the programming commands. You can use this script as the basis for developing your own command-line flash programming flow.

> For more details about the Nios II EDS and command-line usage of the Nios II Flash Programmer and related tools, refer to the *Nios II Flash Programmer User Guide*.

nios2-flash-programmer

This command programs common flash interface (CFI) memory. Because the Nios II flash programmer uses the JTAG interface, the `nios2-flash-programmer` command has the same options for this interface as do other commands. You can obtain information about the command-line options for this command with the `--help` option.

nios2-flash-programmer Usage Example

You can perform the following steps to program a CFI device:

1. Follow the steps in "nios2-download" on page 4–9, or use the Nios II EDS, to program your FPGA with a design that interfaces successfully to your CFI device.

2. Type the following command to verify that your flash device is detected correctly:

```
nios2-flash-programmer –debug –base=<base address>↵
```

where *<base address>* is the base address of your flash device. The base address of each component is displayed in SOPC Builder. If the flash device is detected, the flash memory's CFI table contents are displayed.

3. Convert your file to flash format (**.flash**) using one of the utilities `elf2flash`, `bin2flash`, or `sof2flash` described in the section "elf2flash, bin2flash, and sof2flash".

4. Type the following command to program the resulting **.flash** file in the CFI device:

```
nios2-flash-programmer –base=<base address> <file>.flash↵
```

5. Optionally, type the following command to reset and start the processor at its reset address:

```
nios2-download –g –r↵
```

elf2flash, bin2flash, and sof2flash

These three commands are often used with the `nios2-flash-programmer` command. The resulting **.flash** file is a standard **.srec** file.

The following two important command-line options are available for the `elf2flash` command:

■ The `-boot=`*<boot copier file>*`.srec` option directs the `elf2flash` command to prepend a bootloader S-record file to the converted ELF file.

■ The `-after=`*<flash file>*`.flash` option places the generated **.flash** file—the converted ELF file—immediately following the specified **.flash** file in flash memory.

The `-after` option is commonly used to place the **.elf** file immediately following the **.sof** in an erasable, programmable, configurable serial (EPCS) flash device.

⚠ **CAUTION** If you use an EPCS device, you must program the hardware image in the device before you program the software image. If you disregard this rule your software image will be corrupted.

Before it writes to any flash device, the Nios II flash programmer erases the entire sector to which it expects to write. In EPCS devices, however, if you generate the software image using the `elf2flash -after` option, the Nios II flash programmer places the software image directly following the hardware image, not on the next flash sector boundary. Therefore, in this case, the Nios II flash programmer does not erase the current sector before placing the software image. However, it does erase the current sector before placing the hardware image.

When you use the flash programmer through the Nios II IDE, you automatically create a script that contains some of these commands. Running the flash programmer creates a shell script (**.sh**) in the **Debug** or **Release** target directory of your project. This script contains the detailed command steps you used to program your flash memory.

Example 4–3 shows a sample auto-generated script.

Example 4–3. Sample Auto-Generated Script:

```
#!/bin/sh
#
# This file was automatically generated by the Nios II IDE Flash Programmer.
#
# It will be overwritten when the flash programmer options change.
#

cd <full path to your project>/Debug

# Creating .flash file for the FPGA configuration
#"<Nios II EDS install path>/bin/sof2flash" --offset=0x400000 \
                --input="full path to your SOF" \
                --output="<your design>.flash"

# Programming flash with the FPGA configuration
#"<Nios II EDS install path>/bin/nios2-flash-programmer" --base=0x00000000 \
                --sidp=0x00810828 --id=1436046714 \
                --timestamp=1169569475 --instance=0 "<your design>.flash"
#
# Creating .flash file for the project
"<Nios II EDS install path>/bin/elf2flash" --base=0x00000000 --end=0x7fffff \
                --reset=0x0 \
                --input="<your project name>.elf" --output="ext_flash.flash" \
                --boot="<path to the bootloader>/boot_loader_cfi.srec"

# Programming flash with the project
"<Nios II EDS install path>/bin/nios2-flash-programmer" --base=0x00000000 \
                --sidp=0x00810828 --id=1436046714 \
                --timestamp=1169569475 --instance=0 "ext_flash.flash"

# Creating .flash file for the read only zip file system
"<Nios II EDS install path>/bin/bin2flash" --base=0x00000000 --location=0x100000\
                --input="<full path to your binary file>" --output="<filename>.flash"

# Programming flash with the read only zip file system
"<Nios II EDS install path>/bin/nios2-flash-programmer" --base=0x00000000 \
                --sidp=0x00810828 --id=1436046714 \
                --timestamp=1169569475 --instance=0 "<filename>.flash"
```

The paths, file names, and addresses in the auto-generated script change depending on the names and locations of the files that are converted and on the configuration of your hardware design.

bin2flash Usage Example

To program an arbitrary binary file to flash memory, perform the following steps:

1. Type the following command to generate your **.flash** file:

   ```
   bin2flash --location=<offset from the base address> \
           -input=<your file> --output=<your file>.flash ↵
   ```

2. Type the following command to program your newly created file to flash memory:

   ```
   nios2-flash-programmer -base=<base address> <your file>.flash ↵
   ```

Altera Command-Line Tools for Software Development and Debug

This section describes Altera command-line tools that are useful for software development and debugging.

nios2-terminal

This command establishes contact with **stdin**, **stdout**, and **stderr** in a Nios II processor subsystem. **stdin**, s**tdout**, and **stderr** are routed through a UART (standard UART or JTAG UART) module within this system.

The `nios2-terminal` command allows you to monitor **stdout**, **stderr,** or both, and to provide input to a Nios II processor subsystem through **stdin**. This command behaves the same as the `nios2-configure-sof` command described in "nios2-configure-sof" on page 4–3 with respect to JTAG cables and devices. However, because multiple JTAG UART modules may exist in your system, the `nios2-terminal` command requires explicit direction to apply to the correct JTAG UART module instance. Specify the instance using the `-instance` command-line option. The first instance in your design is `0` (`-instance "0"`). Additional instances are numbered incrementally, starting at `1` (`-instance "1"`).

nios2-download

This command parses Nios II **.elf** files, downloads them to a functioning Nios II processor, and optionally runs the **.elf** file.

As for other commands, you can obtain command-line option information with the `--help` option. The `nios2-download` command has the same options as the `nios2-terminal` command for dealing with multiple JTAG cables and Nios II processor subsystems.

nios2-download Usage Example

To download (and run) a Nios II **.elf** program:

1. Open a Nios II command shell.

2. Change to the directory in which your **.elf** file is located. If you use the Nios II IDE for development, the correct location is often the **Debug** or **Release** subdirectory of your top-level project.If you use the Nios II SBT, the correct location is the **app** folder.

3. In the command shell, type the following command to download and start your program:
   `nios2-download -g <project name>.elf` ↵

4. Optionally, use the `nios2-terminal` command to connect to view any output or provide any input to the running program.

nios2-stackreport

This command returns a brief report on the amount of memory still available for stack and heap from your project's **.elf** file.

This command does not help you to determine the amount of stack or heap space your code consumes during runtime, but it does tell you how much space your code has to work in.

Example 4–4 illustrates the information this command provides.

Example 4–4. nios2-stackreport Command and Response

```
[SOPC Builder]$ nios2-stackreport <your project>.elf
Info: (<your project>.elf) 6312 KBytes program size (code + initialized data).
Info:                      10070 KBytes free for stack + heap.
```

nios2-stackreport Usage Example

To use the `nios2-stackreport` command, perform the following steps:

1. Open a Nios II command shell.

2. Change to the directory in which your **.elf** file is located.

3. In the command shell, type the following command:

    `nios2-stackreport` <*your project*>`.elf` ↵

validate_zip

The Nios II EDS uses this command to validate that the files you use for the Read Only Zip Filing System are uncompressed. You can use it for the same purpose.

validate_zip Usage Example

To use the `validate_zip` command, perform the following steps:

1. Open a Nios II command shell.

2. Change to the directory in which your **.zip** file is located.

3. In the command shell, type the following command:

    `validate_zip` <*file*>`.zip` ↵

    If no response appears, your **.zip** file is not compressed.

nios2-ide

On Linux and Windows systems, you can type `nios2-ide` in a command shell to launch the Nios II IDE. On Windows systems, you can also use the Nios II IDE launch icon in SOPC Builder.

The `nios2-ide` command does not call the executable file directly. Instead, it runs a simple Bourne shell wrapper script, which calls the **nios2-ide** executable file. The Linux and Windows platform versions of the wrapper script follow.

Linux wrapper script

```
#!/bin/sh
# This is the linux-gtk version of the nios2-ide launcher script
# set the default workspace location for linux
WORKSPACE="$HOME/nios2-ide-workspace-7.2"
WORKSPACE_ARGS="-data $WORKSPACE"
# if -data is already passed in, we can't specify it
```

```
# again when calling nios2-ide
for i in $*
do
  if [ "x$i" = "x-data" ]; then
        WORKSPACE_ARGS=""
  fi
done
exec <Nios II EDS install path>/bin/eclipse/nios2-ide -configuration
$HOME/.nios2-ide-6.1 $WORKSPACE_ARGS "$@"
```

Windows wrapper script

```
#!/bin/sh
# This is the win32 version of the nios2-ide launcher script
# It simply invokes the binary with the same arguments as
# passed in.
# By doing this, the user will default to the same workspace as
# when launched using the Windows shortcut, as "persisted"
# in the configuration/.settings/org.eclipse.ui.ide.prefs file.
cd "<Nios II EDS install path>/bin/eclipse"
exec ./nios2-ide-console "$@"
```

nios2-gdb-server

This command starts a GNU Debugger (GDB) JTAG conduit that listens on a specified TCP port for a connection from a GDB client, such as a `nios2-elf-gdb` client.

Occasionally, you may have to terminate a GDB server session. If you no longer have access to the Nios II command shell session in which you started a GDB server session, or if the offending GDB server process results from an errant Nios II IDE debugger session, you should stop the **nios2-gdb-server.exe** process on Windows platforms, or type the following command on Linux platforms:

```
pkill -9 -f nios2-gdb-server ↵
```

nios2-gdb-server Usage Example

The Nios II IDE and most of the other available debuggers use the `nios2-gdb-server` and `nios2-elf-gdb` commands for debugging. You should never have to use these tools at this low level. However, in case you prefer to do so, this section includes instructions to start a GDB debugger session using these commands, and an example GDB debugging session.

You can perform the following steps to start a GDB debugger session:

1. Open a Nios II command shell.

2. In the command shell, type the following command to start the GDB server on the machine that is connected through a JTAG interface to the Nios II system you wish to debug:

```
nios2-gdb-server --tcpport 2342 --tcppersist ↵
```

If the transfer control protocol port 2342 is already in use, use a different port.

Following is the system response:

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: OK
Listening on port 2342 for connection from GDB:
```

Now you can connect to your server (locally or remotely) and start debugging.

3. Type the following command to start a GDB client that targets your **.elf** file:
nios2-elf-gdb *<file>*.elf ↵

Example 4–5 shows a sample session.

Example 4–5. Sample Debugging Session

```
GNU gdb 6.1
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=nios2-elf"...
(gdb) target remote <your_host>:2342
Remote debugging using <your_host>:2342
OS_TaskIdle (p_arg=0x0) at sys/alt_irq.h:127
127      {
(gdb) load
Loading section .exceptions, size 0x1b0 lma 0x1000020
Loading section .text, size 0x3e4f4 lma 0x10001d0
Loading section .rodata, size 0x4328 lma 0x103e6c4
Loading section .rwdata, size 0x2020 lma 0x10429ec
Start address 0x10001d0, load size 281068
Transfer rate: 562136 bits/sec, 510 bytes/write.
(gdb) step
.
.
.
(gdb) quit
```

Possible commands include the standard debugger commands `load`, `step`, `continue`, `run`, and `quit`. Press Ctrl+c to terminate your GDB server session.

nios2-debug

This command is a wrapper around the Tcl/Tk-based Insight GDB GUI, which installs with the Altera-specific GNU GDB distribution that is part of the Nios II EDS.

The command-line option `-save-gdb-script` saves the session script, and the option `-command=`*<GDB script name>* restores a previous GDB session by executing its previously saved GDB script. Use this option to restore break and watch points.

For more information about the Insight GDB GUI, refer to the Insight documentation available at sources.redhat.com.

nios2-debug Usage Example

After you generate the **.elf** file manually or using the Nios II ISBT or the Nios II IDE, perform the following steps to open an Insight debugger session:

1. Open a Nios II command shell.

2. Change to the directory in which your **.elf** file is located.

   If you use the Nios II IDE for development, the correct location is often the **Debug** or **Release** subdirectory of your top-level project.

3. In the command shell, type the following command:

```
nios2-debug <file>.elf ↵
```

Your **.elf** file is parsed and downloaded to memory in your Nios II subsystem, and the main debugger window opens, with the first executable line in the `main()` function highlighted. This debugger window displays your Insight debugging session. Simply click on the **Continue** menu item to run your code, or set some breakpoints to experiment.

Altera Command-Line Nios II Software Build Tools

The Nios II software build tools are command-line utilities available from a Nios II command shell that enable you to create application, board support package (BSP), and library software for a particular Nios II hardware system. Use these tools to create a portable, self-contained makefile-based project that can be easily modified later to suit your build flow.

Unlike the Nios II IDE-based flow, proficient use of these tools requires some expertise with the GNU make-based software build flow. Before you use these tools, refer to the *Introduction to the Nios II Software Build Tools* and the *Using the Nios II Software Build Tools* chapters of the *Nios II Software Developer's Handbook*.

The following sections summarize the commands available for generating a BSP for your hardware design and for generating your application software. Many additional options are available in the Nios II software build tools.

For an overview of the tools summarized in this section, refer to the *Introduction to the Nios II Software Build Tools* chapter of the *Nios II Software Developer's Handbook.*

For information on the many additional options available to you in the Nios II software build tools, refer to the *Introduction to the Nios II Software Build Tools*, *Using the Nios II Software Build Tools*, and *Nios II Software Build Tools Reference* chapters of the *Nios II Software Developer's Handbook*, and the *Developing Nios II Software* chapter of the *Embedded Design Handbook*.

BSP Related Tools

Use the following command-line tools to create a BSP for your hardware design:

- `nios2-bsp-create-settings` creates a BSP settings file.
- `nios2-bsp-update-settings` updates a BSP settings file.
- `nios2-bsp-query-settings` queries an existing BSP settings file.
- `nios2-bsp-generate-files` generates all the files related to a given BSP settings file.
- `nios2-bsp` is a script that includes most of the functionality of the preceding commands.
- `create-this-bsp` is a high-level script that creates a BSP for a specific hardware design example.

Application Related Tools

Use the following commands to create and manipulate Nios II application and library projects:

■ `nios2-app-generate-makefile` creates a makefile for your application.

■ `nios2-lib-generate-makefile` creates a makefile for your application library.

■ `nios2-c2h-generate-makefile` creates a makefile fragment for the C2H compiler.

■ `create-this-app` is a high-level script that creates an application for a specific hardware design example.

GNU Command-Line Tools

The Nios II GCC toolchain contains the GNU Compiler Collection, the GNU binutils, and the newlib C library. You can follow links to detailed documentation from the Nios II EDS documentation launchpad found in your Nios II EDS distribution. To start the launchpad on Windows platforms, on the Start menu, click **All Programs**. On the All Programs menu, on the Altera submenu, on the Nios II EDS *<version>* submenu, click **Literature**. On Linux platforms, open
*<Nios II EDS install dir>***/documents/index.htm** in a web browser. In addition, more information about the GNU GCC toolchain is available on the World Wide Web.

nios2-elf-addr2line

This command returns a source code line number for a specific memory address. The command is similar to but more specific than the `nios2-elf-objdump` command described in "nios2-elf-objdump" on page 4–20 and the `nios2-elf-nm` command described in "nios2-elf-nm" on page 4–19.

Use the `nios2-elf-addr2line` command to help validate code that should be stored at specific memory addresses. Example 4–6 illustrates its usage and results:

Example 4–6. nios2-elf-addr2line Utility Usage Example

```
[SOPC Builder]$ nios2-elf-addr2line --exe=<your project>.elf 0x1000020
<Nios II EDS install path>/components/altera_nios2/HAL/src/alt_exception_entry.S:99
```

nios2-elf-addr2line Usage Example

To use the `nios2-elf-addr2line` command, perform the following steps:

1. Open a Nios II command shell.

2. In the command shell, type the following command:

   ```
   nios2-elf-addr2line <your project>.elf <your_address_0>,\
   <your_address_1>,...,<your_address_n> ↵
   ```

   If your project file contains source code at this address, its line number appears.

nios2-elf-gdb

This command is a GDB client that provides a simple shell interface, with built-in commands and scripting capability. A typical use of this command is illustrated in the section "nios2-gdb-server" on page 4–11.

nios2-elf-readelf

Use this command to parse information from your project's **.elf** file. The command is useful when used with **grep**, **sed**, or **awk** to extract specific information from your **.elf** file.

nios2-elf-readelf Usage Example

To display information about all instances of a specific function name in your **.elf** file, perform the following steps:

1. Open a Nios II command shell.

2. In the command shell, type the following command:

   nios2-elf-readelf -symbols *<project>*.elf | grep *<function name>* ↵

Example 4–7 shows a search for the http_read_line() function in a **.elf** file.

Example 4–7. Search for the http_read_line Function Using nios2-elf-readelf

```
[SOPC Builder]$ nios2-elf-readelf.exe –s my_file.elf | grep http_read_line
1106: 01001168   160 FUNC    GLOBAL DEFAULT    3 http_read_line
```

Table 4–2 lists the meanings of the individual columns in Example 4–7.

Table 4–2. Interpretation of nios2-elf-readelf Command Response

Value	Description
1106	Symbol instance number
0100168	Memory address, in hexadecimal format
160	Size of this symbol, in bytes
FUNC	Type of this symbol (function)
GLOBAL	Binding (values: GLOBAL, LOCAL, and WEAK)
DEFAULT	Visibility (values: DEFAULT, INTERNAL, HIDDEN, and PROTECTED)
3	Index
http_read_line	Symbol name

You can obtain further information about the ELF file format online. Each of the ELF utilities has its own man page.

nios2-elf-ar

This command generates an archive (**.a**) file containing a library of object (**.o**) files. The Nios II IDE uses this command to archive the System Library project.

nios2-elf-ar Usage Example

To archive your object files using the nios2-elf-ar command, perform the following steps:

1. Open a Nios II command shell.

2. Change to the directory in which your object files are located.

3. In the command shell, type the following command:

    `nios2-elf-ar q <archive_name>.a <object files>`

Example 4–8 shows how to create an archive of all of the object files in your current directory. In Example 4–8, the `q` option directs the command to append each object file it finds to the end of the archive. After the archive file is created, it can be distributed for others to use, and included as an argument in linker commands, in place of a long object file list.

Example 4–8. nios2-elf-ar Command Response

```
[SOPC Builder]$ nios2-elf-ar q <archive_name>.a *.o
nios2-elf-ar: creating <archive_name>.a
```

Linker

Use the `nios2-elf-g++` command to link your object files and archives into the final executable format, ELF.

Linker Usage Example

To link your object files and archives into a **.elf** file, open a Nios II command shell and call `nios2-elf-g++` with appropriate arguments. The following example command line calls the linker:

```
nios2-elf-g++ -T'<linker script>' -msys-crt0='<crt0.o file>' \
-msys-lib=<system library> -L '<The path where your libraries reside>' \
-DALT_DEBUG -O0 -g -Wall -mhw-mul -mhw-mulx -mno-hw-div \
-o  <your project>.elf <object files> -lm ↵
```

The exact linker command line to link your executable may differ. When you build a project in the Nios II SBT or the Nios II IDE, you can see the command line used to link your application. To turn on this option in the Nios II IDE, on the Window menu, click **Preferences**, select the **Nios II** tab, and enable **Show command lines when running make**. You can also force the command lines to display by running `make` without the `-s` option from a Nios II command shell.

☞ Altera recommends that you not use the native linker `nios2-elf-ld` to link your programs. For the Nios II processor, as for all softcore processors, the linking flow is complex. The `g++` (`nios2-elf-g++`) command options simplify this flow. Most of the options are specified by the `-m` command-line option, but the options available depend on the processor choices you make.

nios2-elf-size

This command displays the total size of your program and its basic code sections.

nios2-elf-size Usage Example

To display the size information for your program, perform the following steps:

1. Open a Nios II command shell.

2. Change to the directory in which your **.elf** file is located.

3. In the command shell, type the following command:

    `nios2-elf-size <project>.elf`

Example 4–9 shows the size information this command provides.

Example 4–9. nios2-elf-size Command Usage

```
[SOPC Builder]$ nios2-elf-size my_project.elf
text    data     bss     dec    hex filename
272904    8224 6183420 6464548  62a424 my_project.elf
```

nios2-elf-strings

This command displays all the strings in a **.elf** file.

nios2-elf-strings Usage Example

The command has a single required argument:

```
nios2-elf-strings <project>.elf
```

nios2-elf-strip

This command strips all symbols from object files. All object files are supported, including ELF files, object files (**.o**) and archive files (**.a**).

nios2-elf-strip Usage Example

```
nios2-elf-strip <options> <project>.elf
```

nios2-elf-strip Usage Notes

The `nios2-elf-strip` command decreases the size of the **.elf** file.

This command is useful only if the Nios II processor is running an operating system that supports ELF natively. If ELF is the native executable format, the entire **.elf** file is stored in memory, and the size savings matter. If not, the file is parsed and the instructions and data stored directly in memory, without the symbols in any case.

Linux is one operating system that supports ELF natively; uClinux is another. uClinux uses the flat (FLT) executable format, which is translated directly from the ELF.

nios2-elf-gdbtui

This command starts a GDB session in which a terminal displays source code next to the typical GDB console.

The syntax for the `nios2-elf-gdbtui` command is identical to that for the `nios2-elf-gdb` command described in "nios2-elf-gdb" on page 4–14.

☞ Two additional GDB user interfaces are available for use with the Nios II GDB Debugger. CGDB, a cursor-based GDB UI, is available at www.sourceforge.net. The Data Display Debugger (DDD) is highly recommended.

nios2-elf-gprof

This command allows you to profile your Nios II system.

For details about this command and the Nios II EDS-based results GUI, refer to *AN 391: Profiling Nios II Systems*.

nios2-elf-insight

The `nios2-debug` command described in "nios2-debug" on page 4–12 uses this command to start an Insight debugger session on the supplied **.elf** file.

nios2-elf-gcc and g++

These commands run the GNU C and C++ compiler, respectively, for the Nios II processor.

Compilation Command Usage Example

The following simple example shows a command line that runs the GNU C or C++ compiler:

```
nios2-elf-gcc(g++) <options> -o <object files> <C files>
```

More Complex Compilation Example

Example 4–10 is a Nios II EDS-generated command line that compiles C code in multiple files in many directories.

Example 4–10. Example nios2-elf-gcc Command Line

```
nios2-elf-gcc -xc -MD -c \
-DSYSTEM_BUS_WIDTH=32 -DALT_NO_C_PLUS_PLUS -DALT_NO_INSTRUCTION_EMULATION \
-DALT_USE_SMALL_DRIVERS -DALT_USE_DIRECT_DRIVERS -DALT_PROVIDE_GMON \
-I.. -I/cygdrive/c/Work/Projects/demo_reg32/Designs/std_2s60_ES/software/\
reg_32_example_0_syslib/Release/system_description \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_timer/HAL/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_timer/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_jtag_uart/HAL/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_jtag_uart/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_pio/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_lcd_16207/HAL/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_lcd_16207/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_sysid/HAL/inc \
-I/cygdrive/c/altera/70_b31/ip/sopc_builder_ip/altera_avalon_sysid/inc \
-I/cygdrive/c/altera/70_b31/nios2eds/components/altera_nios2/HAL/inc \
-I/cygdrive/c/altera/70_b31/nios2eds/components/altera_hal/HAL/inc \
-DALT_SINGLE_THREADED -D__hal__ -pipe -DALT_RELEASE -O2 -g -Wall\
-mhw-mul -mhw-mulx -mno-hw-div -o obj/reg_32_buttons.o ../reg_32_buttons.c
```

nios2-elf-c++filt

This command demangles C++ mangled names. C++ allows multiple functions to have the same name if their parameter lists differ; to keep track of each unique function, the compiler mangles, or decorates, function names. Each compiler mangles functions in a particular way.

For a full explanation, including more details about how the different compilers mangle C++ function names, refer to standard reference sources for the C++ language compilers.

nios2-elf-c++filt Usage Example

To display the original, demangled function name that corresponds to a particular symbol name, you can type the following command:

```
nios2-elf-c++filt -n <symbol name> ↵
```

For example,

```
nios2-elf-c++filt -n _Z11my_functionv ↵
```

More Complex nios2-elf-c++filt Example

The following example command line causes the display of all demangled function names in an entire file:

```
nios2-elf-strings <file>.elf | grep ^_Z | nios2-elf-c++filt -n
```

In this example, the `nios2-elf-strings` operation outputs all strings in the **.elf** file. This output is piped to a `grep` operation that identifies all strings beginning with `_Z`. (GCC always prepends mangled function names with `_Z`). The output of the `grep` command is piped to a `nios2-elf-c++filt` command. The result is a list of all demangled functions in a GCC C++ **.elf** file.

nios2-elf-nm

This command list the symbols in a **.elf** file.

nios2-elf-nm Usage Example

The following two simple examples illustrate the use of the `nios2-elf-nm` command:

- `nios2-elf-nm <project>.elf` ↵
- `nios2-elf-nm <project>.elf | sort -n` ↵

More Complex nios2-elf-nm Example

To generate a list of symbols from your **.elf** file in ascending address order, use the following command:

```
nios2-elf-nm <project>.elf | sort -n > <project>.elf.nm
```

The *<project>*.**elf.nm** file contains all of the symbols in your executable file, listed in ascending address order. In this example, the `nios2-elf-nm` command creates the symbol list. In this text list, each symbol's address is the first field in a new line. The `-n` option for the `sort` command specifies that the symbols be sorted by address in numerical order instead of the default alphabetical order.

nios2-elf-objcopy

Use this command to copy from one binary object format to another, optionally changing the binary data in the process.

Though typical usage converts from or to ELF files, the `objcopy` command is not restricted to conversions from or to ELF files. You can use this command to convert from, and to, any of the formats listed in Table 4–3.

Table 4–3. -objcopy Binary Formats

Command (…-objcopy)	Comments
elf32-littlenios2, elf32-little	Header little endian, data little endian, the default and most commonly used format
elf32-bignios2, elf32-big	Header big endian, data big endian
srec	S-Record (SREC) output format
symbolsrec	SREC format with all symbols listed in the file header, preceding the SREC data
tekhex	Tektronix hexadecimal (TekHex) format
binary	Raw binary format Useful for creating binary images for storage in flash on your embedded system
ihex	Intel hexadecimal (ihex) format

You can obtain information about the TekHex, ihex, and other text-based binary representation file formats on the World Wide Web. As of the initial publication of this handbook, you can refer to the www.sbprojects.com knowledge-base entry on file formats.

nios2-elf-objcopy Usage Example

To create an SREC file from an ELF file, use the following command:

```
nios2-elf-objcopy –O srec <project>.elf <project>.srec
```

ELF is the assumed binary format if none is listed. For information about how to specify a different binary format, in a Nios II command shell, type the following command:

```
nios2-elf-objcopy --help ↵
```

nios2-elf-objdump

Use this command to display information about the object file, usually an ELF file.

The `nios2-elf-objdump` command supports all of the binary formats that the `nios2-elf-objcopy` command supports, but ELF is the only format that produces useful output for all command-line options.

nios2-elf-objdump Usage Description

The Nios II EDS uses the following command line to generate object dump files:

```
nios2-elf-objdump –D –S –x <project>.elf > <project>.elf.objdump
```

nios2-elf-ranlib

Calling `nios2-elf-ranlib` is equivalent to calling `nios2-elf-ar` with the `-s` option (`nios2-elf-ar -s`).

For further information about this command, refer to "nios2-elf-ar" on page 4–15 or type `nios2-elf-ar --help` in a Nios II command shell.

Document Revision History

Table 4–4 shows the revision history for this document.

Table 4–4. Document Revision History

Date	Version	Changes
July 2011	2.2	■ Fixed outdated references. ■ Clarified `jtagconfig -n` result node numbers vary by device, system configuration, and so on.
April 2009	2.1	Fixed outdated reference.
November 2008	2.0	Added System Console.
March 2008	1.0	Initial release.