EXCALIBUR™

Nios Embedded Processor

Software Development Reference

Manual
Version 1.1
March 2001

ALTERA®

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

A-MNL-NIOSPROG-01

About this Manual

This document provides information for programmers developing software for the Nios™ embedded soft core processor. Primary focus is given to code written in the C programming language; however, several sections discuss the use of assembly code as well.

The terms Nios processor or Nios embedded processor are used when referring to the Altera® soft core microprocessor in a general or abstract context.

The term Nios CPU is used when referring to the specific block of logic, in whole or part, that implements the Nios processor architecture.

Table 1 below shows the programmer's reference manual revision history.

Table 1 .Revision History

Revision	Date	Description
Version 1.1	March 2001	Nios Embedded Processor Software Development Reference Manual - printed

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at http://www.altera.com.

For additional information about Alter aproducts, consult the sources shown in Table 2.

Table 2 .How to Contact Altera

Information Type	Access	USA & Canada	All Other Locations
Altera Literature Services	Electronic mail	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-7606	(408) 544-7606
Technical support	Telephone hotline	(800) 800-EPLD (6:00 a.m. to 6:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:30 a.m. to 5:30 p.m. Pacific Time)
	Fax	(408) 544-6401	(408) 544-6401 *(1)*
	Electronic mail	telecom@altera.com	telecom@altera.com
	FTP site	ftp.altera.com	ftp.altera.com
General product information	Telephone	(408) 544-7104	(408) 544-7104 *(1)*
	World-wide web site	http://www.altera.com	http://www.altera.com

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

Typographic Conventions

The ***Nios Embedded Processor Programmer's Reference Manual*** uses the typographic conventions shown in Table 3.

Table 3 . Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file.
Bold italic type	Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Device Data Book***.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75 (High-Speed Board Design).*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of Quartus II and MAX+PLUS II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable."
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `_n`, e.g., `reset_n`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\max2work\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

Notes:

Contents

Notes:

Overview

Introduction

The Nios embedded processor is a soft core CPU optimized for programmable logic and system-on-a-programmable-chip (SOPC) designs. SOPC designs are created using the MegaWizard® Plug-In Manager included in the Quartus II™ development software. When the Nios system builder generates a design, several results occur:

1. The system memory map is checked for consistency. Peripheral addresses and interrupt priorities are verified to be unique, and fall within the range of valid entries for the CPU. If not, appropriate errors are reported and corrections must be made before continuing.

2. A custom software development kit (SDK) is generated for the new Nios system. The SDK consists of a compiled library of software routines for the SOPC design, a Makefile for rebuilding the library, and C header files containing structures for each peripheral.

3. The system hardware is synthesized, placed, routed, and output in a file format suitable to configure an Altera programmable logic device.

This document covers the SDK, generated in step 2 above. All directories and files mentioned are assumed to be part of the SDK unless otherwise specified.

Project Considerations

Many design scenarios are possible in Nios processor-based systems. Before beginning development, it is helpful to make some decisions based on application requirements. The following issues should be considered before starting the SOPC design:

- Memory Model
  Application code can reside in on-chip RAM or ROM or external memory devices. Applications that require internal memory resources will typically be limited to <20K of code space. Consequently, they may require hand-optimized assembly language to remain small.

  External memory allows larger code space at the cost of memory devices (RAM, ROM, flash, etc.).

- CPU Footprint
  The Nios CPU can be configured with a 32-bit or 16-bit data path. The 16-bit version uses fewer logic elements (LEs), can access a narrower range of addresses, and runs faster than the 32-bit implementation.
- Software Acceleration
  Multiplication-intensive software runs faster when a hardware multiplier unit is added to the CPU core. Adding a multiplier unit uses additional LEs.

Development Flow

The following outline describes a typical development flow used when creating a Nios processor-based design from scratch. It is assumed that initial development will be accomplished using the development board and software tools included in the Nios development kit.

Developing applications using the Nios embedded processor is slightly different than that of traditional processors since the designer is allowed to configure the processor architecture and specify the peripheral content. In other words, a designer can build a microcontroller according to system design requirements, as opposed to selecting a pre-built microcontroller with a fixed set of peripherals, on-chip memory, and external interfaces.

The Nios development board included in the kit comes with a 32-bit reference design (processor, on-chip memory with monitor, and peripherals), and application code pre-loaded in on-board flash memory. This reference design will help you quickly familiarize yourself with the development tools prior to starting your custom design (see the *Nios Embedded Processor System Builder Tutorial*). If possible, begin your software design using the Nios development board as your target hardware platform.

Step 1: Define the Processor

Based on your system needs, decide the following:

CPU data path

How wide a data path will your application require, 32-bit or 16-bit? If a 32-bit data path is not needed, choosing 16-bit data path will generate a smaller, faster CPU core.

Data Path	Logic Elements Used	Address Range
16-bits	1100	128 K
32-bits	1700	4 GB

Multiplier

If your code performs few multiplication operations, does not contain time critical multiplication, or you are trying to make the CPU core as small as possible, use the software math libraries included with the C compiler. If, on the other hand, your code performs numerous multiplication operations or needs to be optimized for speed, choose one of the dedicated hardware multipliers (MSTEP or MUL).

Multiplication Option	Additional Logic Elements Used	Clock Cycles 16x16>32	Clock Cycles 32x32>32
None (software	0	80	250
MSTEP	+200	18	80
MUL	+400	2	16

On-Chip Memory

Decide how much on-chip ROM and RAM your system will require. The Nios processor uses embedded system blocks (ESBs) for on-chip memory. There are practical limits to the number of ESBs used for on-chip memory (see the *Altera Device Data Book* for details on the number of ESBs available in particular devices). The Nios system builder software imposes a maximum limit of 20K per on-chip memory device.

Off-Chip Memory

Interfaces to off-chip memory are provided for flash memory and SRAM. The GERMS monitor, included in the development kit, contains software routines for writing to, and erasing Advanced Micro Devices (AMD) flash devices.

Peripherals

Decide the type and number of peripherals to be connected to the Nios processor. A number of peripherals come with the Nios development kit. Interface to off-chip or custom on-chip peripherals using either the parallel input/output peripheral (PIO), or user-defined interface. Below is a list of peripherals included with the Nios development kit:

Peripheral	Description
UART	Universal Asynchronous Receiver Transmitter
PIO	Parallel Input/Output
Timer	General-purpose timer
SPI	Serial Peripheral Interface
User Defined Interface	Custom interface to on-chip and off-chip peripherals
Off-chip shared bus	Shared interface to off-chip peripherals and memory

Step 2: Build the Processor

Using the Quartus development software and the MegaWizard Plug-In manager, generate a custom processor system based on the choices you made in Step 1. As you build the processor, you will:

- Choose the width of the processor data path.
- Specify the processor boot address.
- Assign peripheral memory addresses and alignment.
- Assign interrupt priorities for peripherals and external interfaces as needed.
- Specify peripheral setup and hold requirements as needed.
- Assign peripheral and memory wait states as needed.
- Enable dynamic bus-sizing to narrow memory (or peripheral) interfaces as needed.
- Assign code (or data) files for on-chip ROM and/or RAM.

Once the Nios system has been created, download the processor configuration ("sof" or "pof" file) to the APEX device on the development board using the Quartus II software and the ByteBlasterMV™ download cable.

A monitor program, called GERMS, is included in the Nios development kit. GERMS allows you to run executable code, read from, and write to memory, download blocks of code (or data) to memory, and erase flash (see the GERMS Monitor section for details). By assigning the GERMS monitor to the processor boot address (typically on-chip ROM), you can immediately begin code development, download, and debug.

See the *Nios Embedded Processor System Builder Tutorial* for step-by-step instructions on creating a Nios processor-based SOPC design.

Step 3: Save the Processor Configuration to FLASH

The programmable logic device configuration (hardware design) on the development board is volatile and is overwritten by the contents of flash memory when the RESET button (SW2) is pressed or power is cycled. The development board contains logic that supports a dual configuration scheme as follows:

By default, the APEX device is configured from a "User" section of flash memory (address range `0x180000 - 0x1BFFFF`). If the APEX device fails to configure due to corrupt or empty "User" section, it is automatically configured from the "Factory" section of flash memory (address range `0x1C0000 - 0x1FFFFF`). When jumper JP2 is shorted, and the RESET button is pressed, the APEX device is forced to configure from the "Factory" section of flash memory.

During development, it is recommended that you always store your new design to the "User" section of flash memory. By doing this, if a hardware bug occurs you can reconfigure the APEX device with the known good reference design stored in the "Factory" section of flash memory. The factory section of flash memory is loaded by Altera with a 32-bit Nios system design.

See hexout2flash on page 36 of this document, or the Programming section of the *Nios Embedded Processor System Builder Tutorial* for details on downloading device configuration files to flash memory.

Step 4: Create the Application Code

Using a text editor (xemacs and vi editors are included) write and compile your application code.

For small- to medium-sized software projects, use nios-build to generate executable code (see nios-build in the *Nios Software Utilities* section of this document for details).

For large projects, use hand-crafted make files. Refer to the online GNU documentation by choosing **Programs > Cygwin > Cygwin Documentation > Using make** (Windows Start Menu) for details on using make.

Step 5: Download the Executable Code

Use nios-run to download and run the application on the development board (see nios-run in the Nios Software Utilities section of this document for details).

Step 6: Debug the Code

If you choose to debug your code using printf(), your messages will be sent to the STDIO (e.g. UART). The nios-run utility acts as a dumb terminal to display these messages on your development system terminal.

If more sophisticated debugging is called for, rebuild your code with debugging set ON, and use the GNU debugger (GDB) to step through the code, examine memory and register contents, etc. See nios-elf-gdb in the Nios Software Utilities section of this document for details.

Step 7: Transition to Auto-Booting Code

Code in On-Chip Memory

Change on-chip RAM to on-chip ROM and rebuild the design (Step 2) using your code to initialize ROM (GERMS monitor is removed completely).

Code in Off-Chip Memory

Store program in flash memory so that the GERMS monitor will automatically execute it after initialization. Use srec2flash to add a routine that copies the executable code from flash memory to SRAM at start time (see srec2flash in the Nios Software Utilities section of this document for details).

Or

Remove the GERMS monitor entirely, and change the reset address to point to the program in flash memory. Use srec2flash to add a routine that copies the executable code from flash memory to SRAM at start time.

Step 8: Transition Design From Nios Development Board to Target Hardware

If possible, use the GERMS monitor to download code to the target hardware. Having the ability to iterate software without burning a new ROM or recompiling the hardware design is very useful.

GERMS Monitor

The GERMS monitor is included in the default reference design, loaded in flash mrmory of the development board. On power-up, the GERMS monitor is the first code to execute and controls the boot process. Once booted, it provides a way to read and write memory.

"GERMS" is an acronym for remembering the rather minimal command set of the monitor program included in the Nios development kit:

G Go (run a program)
E Erase flash
R Relocate next download
M Memory set and dump
S Send S-records
: Send I-Hex records

Monitor Commands

When the monitor is running, it is always waiting for commands. Commands consist of a letter, followed by an address. Some commands take two addresses, separated by a hyphen. The write command takes an address followed by a colon, followed by data to write.

Commands are executed as they are typed. If you are writing to memory, for example, each word is stored as soon as you enter it. There is no backspace. The only "line editing" available is the ability to restart the monitor immediately, by pressing the ESC key.

All numbers and addresses entered into the monitor are in hexadecimal.

Syntax	Example	Description
G<base address>	G40000	GO—Execute a CALL instruction to the specified address.
E<base address>	E180000	Erase flash memory. If the address is within the range of the "flash" ROM, the sector containing that address will be erased.
R<from address>-<to address>	R0-180000	Offset the next download. The next S-record or I-Hex record downloaded will be stored offset by the range specified.
M<address>	M50000	Display memory starting from the address.
M<address>-<address>	M40000-40100	Display a range of memory. Pressing <CR> again will show the same number of bytes, starting where the last M command ended.
M<address>:<value> <value>…	M50000:1 2 3 4	Write successive 16-bit words to memory, until the end of line.
M<address>-<address>:<value>	M50000-50100:AA55	Fill a range of memory with a 16-bit word.
<CR>	<CR>	Display the next 64 bytes of memory.
S<S-record data>	S21840000…	Write S-record to next memory location.
:<I-hex record data>	:80000004…	Write I-hex record to next memory location.
<ESC>	<ESC>	Restart the monitor.

Boot Process

The monitor is located at address zero, 0x0000, in the default configuration of the Nios development board.

There are several ways the monitor might come to be executed. When the default design is downloaded, execution begins at address zero, which is the monitor. Later on, if any TRAP or interrupt occurs, and the vector table has not been altered, the monitor will be executed.

When the monitor starts running, it performs some system initialization:

1. Turns off interrupts on the UART, timer, and switch PIO.
2. Sets current window pointer (CWP) to HI_LIMIT.
3. Sets interrupt priority (IPR) to 63.
4. Sets %sp to 0x080000 (NA_RAMTop).

It then looks for code to execute out of flash memory:

5. Examines the two bytes at 0x14000C (NA_FlashBase + 0x04000C).

6. Examines button 0 on the switch PIO (SW4).

7. If the button is not pressed, and the two bytes contain 'N' and 'i' , then the monitor executes a CALL to location 0x140000 (NA_FlashBase + 0x040000).

If the code is not executed in step 7, or that code returns, the following steps occur:

8. Prints an 8-digit version number to STDOUT, of the form "#vvvvPPPP" followed by a carriage return, where "vvvv" is a monitor pseudo-version—it will be different but not necessarily consecutive for different builds of the monitor—, and PPPP is the processor version number, as retrieved from processor register CTL 6.

9. Wait for user commands from STDIN.

Booting From Flash Memory

Programs can be stored in flash memory and caused to execute on power-up or reset. This is particularly useful when developing application code targeted for flash memory.

The GERMS monitor checks for the existence of application software in flash memory (Boot Process, Step 5). If found, the processor immediately executes the code. The software utility, srec2flash, should be used to prepare programs for this style of operation (see srec2flash in the Nios Software Utilities section of this document). Srec2flash adds a small piece of code to the beginning of the program that will copy the application code from flash (slow memory) to SRAM (fast memory) then run from SRAM.

To return program execution to the GERMS monitor (i.e., avoid running code stored in flash memory) perform the following steps:

1. Hold down SW4.

2. Press then release the RESET button (SW2).

3. Release SW4.

Overview Of The SDK Tree

The SDK will be generated as a subdirectory of your Quartus (or MAX+PLUS® II) project. It will be named with the name of the Nios system, with "_sdk" appended. The 32-bit reference design (ref_32_system), for example, has the following directory structure:

```
.../ref_32_system_sdk/
      |
      +--- inc/
      |
      +--- lib/
      |
      +--- src/
```

The Include ("inc") Directory

```
[bash] ...inc/: ls -l
total 17
-rw-r--r--  1 niosuser Administ    281 Jan 24 15:19 nios.h
-rw-r--r--  1 niosuser Administ    245 Jan 24 15:19 nios.s
-rw-r--r--  1 niosuser Administ   8990 Jan 24 15:19 nios_macros.s
-rw-r--r--  1 niosuser Administ   3853 Jan 24 15:19 nios_map.h
-rw-r--r--  1 niosuser Administ   3877 Jan 24 15:19 nios_map.s
-rw-r--r--  1 niosuser Administ   5267 Jan 24 15:19 nios_peripherals.h
-rw-r--r--  1 niosuser Administ   3755 Jan 24 15:19 nios_peripherals.s
```

The SDK include directory, called "inc", contains several files intended to be included from your application programs. These files define the peripheral addresses, interrupt priorities, register structures, and other useful constants and macros.

Files are included in both C and assembly language. Each file of your program should include nios.h if the file is written in C or C++, or nios.s if the file is written in assembly language.

nios.h (and nios.s)

Includes the other relevant include files described below:

nios_macros.s

Includes various useful assembly language macros. See *Assembly Macros* in Appendix B for more details.

nios_peripherals.h (and nios_peripherals.s)

Contains register maps for each peripheral in your system. Additionally, nios_peripherals.h contains C prototypes for library routines available for each peripheral.

For C programs, the register maps are provided as structures. For example, the timer peripheral's structure is as follows:

```
typedef volatile struct
{
int np_timerstatus;  // read only, 2 bits (any write to clear TO)
int np_timercontrol; // write/readable, 4 bits
int np_timerperiodl; // write/readable, 16 bits
int np_timerperiodh; // write/readable, 16 bits
int np_timersnapl;   // read only, 16 bits
int np_timersnaph;   // read only, 16 bits
} np_timer;

enum
{
np_timerstatus_run_bit    = 1, // timer is running
np_timerstatus_to_bit     = 0, // timer has timed out

np_timercontrol_stop_bit  = 3, // stop the timer
np_timercontrol_start_bit = 2, // start the timer
np_timercontrol_cont_bit  = 1, // continuous mode
np_timercontrol_ito_bit   = 0, // enable time out interrupt

np_timerstatus_run_mask    = (1<<1), // timer is running
np_timerstatus_to_mask     = (1<<0), // timer has timed out

np_timercontrol_stop_mask  = (1<<3), // stop the timer
np_timercontrol_start_mask = (1<<2), // start the timer
np_timercontrol_cont_mask  = (1<<1), // continous mode
np_timercontrol_ito_mask   = (1<<0)  // enable time out interrupt
};
```

Each register is included as an integer (int) structure field, so that it can be used on a 16-bit or a 32-bit Nios processor interchangeably.

For the registers that have sub-fields or control bits, additional constants are defined to reference those fields, by both mask and bit number. (The bit numbers are useful for the Nios assembly language instructions SKP0 and SKP1).

nios_map.h (and nios_map.s)

This file provides addresses for all your peripherals, interrupt numbers, and other useful constants. Here is an excerpt from the reference design's nios_map.h:

```
#define na_timer1                   ((np_timer *) 0x00000440)
#define na_timer1_irq                               25
#define na_led_pio                  ((np_pio *)    0x00000460)
#define na_button_pio               ((np_pio *)    0x00000470)
#define na_button_pio_irq                           27
#define nasys_printf_uart           ((np_uart *)   0x00000400)
#define nasys_printf_uart_irq                       26
```

The name na_timer1 is derived from the peripheral's name "timer", with "na_" added to the beginning, standing for "Nios address". It is defined as a number cast to the type of "np_timer *". This allows the symbol "na_timer1" to be treated as a pointer to a timer structure. The following is an example of code written to access the timer:

```
int status = na_timer1->np_timerstatus;     /* get status of timer1 */
```

The Library ("lib") Directory

```
[bash] ...lib/: ls -l
total 119
-rw-r--r--   1 niosuser Administ     3177 Jan 25 12:46 Makefile
-rw-r--r--   1 niosuser Administ    95944 Jan 25 12:46 libnios32.a
-rw-r--r--   1 niosuser Administ     3067 Jan 24 01:19 nios_cstubs.s
   ...

drwxr-xr-x   2 niosuser Administ    12288 Jan 25 12:46 obj32/
-rw-r--r--   1 niosuser Administ     5871 Jan 24 01:19 pio_lcd16207.c
   ...

-rw-r--r--   1 niosuser Administ      803 Jan 24 01:19 uart_txhex32.s
-rw-r--r--   1 niosuser Administ      699 Jan 24 01:19 uart_txstring.s
[bash] ...lib/:
```

The SDK library directory, called "lib", contains a Makefile, an archive file, source, and object files for libraries usable by your Nios system.

Some of the source files are in assembly language, and others are in C. The archive contains assembled (or compiled) versions of routines from each file, suitable for linking to your program. The routines are described in detail in the Nios Library Routines section of this document.

The command line tools "nios-build" uses the appropriate library directory, either "libnios32.a" or "libnios16.a", depending whether it is building for a 32-bit or Nios 16-bit system, respectively.

The Makefile contains instructions for rebuilding the archive file. The beginning of the Makefile contains several settings to enable or disable various features of the Nios library. Here is an excerpt from a typical Nios library Makefile.

```
#
# Nios SDK Generated Makefile
# 2001.01.24 01:19:30
# //d/niosbuild/srctree/Delta/SWDev/bin/nios_reference32.ptf
#
NIOS_USE_MSTEP = 1  # CPU option (shift, test, & add)
NIOS_USE_MULTIPLY = 0  # CPU option (16x16->32)
NIOS_MONITOR = nios_germs_monitor
NIOS_SYSTEM_NAME = nios_system_module
NIOS_USE_CONSTRUCTORS = 1  # Call c++ static constructors smaller
footprint
NIOS_USE_CWPMGR = 1  # Turn off to disable underflow handling
(dangerous)
NIOS_USE_FAST_MUL = 1  # Faster but larger int multiply routine
NIOS_USE_SMALL_PRINTF = 1  # Smaller non-ANSI printf formats

M = 32  # Nios 32
```

You can change each of these settings to customize the Nios library. After changing a setting, type "make -s all" from the command line to rebuild the library.

Below is an explanation of each setting:

NIOS_USE_MSTEP

If NIOS_USE_MSTEP is set to 1, then the Nios library will override the standard multiplication routine with a faster one that uses the MSTEP instruction. This is set to 1 automatically if the MSTEP feature is selected in the system builder software. (This setting must be used in conjunction with NIOS_USE_FAST_MUL).

NIOS_USE_MULTIPLY

If NIOS_USE_MULTIPLY is set to 1, then the Nios library will override the standard multiplication routine with a faster one that uses the MUL instruction. (This runs even faster than MSTEP multiplication.) This is set to 1 automatically if the MULTIPLY feature is selected in the system builder software. (This setting must be used in conjunction with NIOS_USE_FAST_MUL.)

NIOS_MONITOR

This is a short string used by the GERMS monitor. The monitor prints this string to the STDIO when it starts up.

NIOS_SYSTEM_NAME

This is a string with the name of the Nios system.

NIOS_USE_CONSTRUCTORS

If NIOS_USE_CONSTRUCTORS is set to 1, then the Nios library will contain startup code to call any initializing code for statically allocated C++ classes. By default, this is set to 1. Changing this setting to 0 will slightly reduce the code footprint of the compiled software if static initialization of C++ classes is not needed. (Useful for small software ROM sizes.)

NIOS_USE_CWPMGR

If NIOS_USE_CWPMGR is set to 1, then the Nios library will contain code for handling register window underflows. Changing this setting to 0 will reduce the code footprint of the compiled software. This should only be done if the code does not call to a subroutine depth that exceeds the register file size. See the *Nios Programmers Reference Manual* for more details.

NIOS_USE_FAST_MUL

To instruct the library to perform integer multiplications with either optional instruction MUL or MSTEP, NIOS_USE_FAST_MUL must be 1. If this setting is 1 and neither MUL nor MSTEP are enabled, then a hand-optimized integer multiplication routine will be linked into the Nios library.

NIOS_USE_SMALL_PRINTF

The standard printf() routine in the GNU libraries takes about 40k of Nios code. It contains support for the complete ANSI printf() specification, including floating point numbers. If NIOS_USE_SMALL_PRINTF is 1, then a more minimal implementation is linked into the Nios library, which takes about 1k of Nios code. This "small printf" supports only integers, and only the formats of %c, %s, %d, %x, and %X.

M

This will be set to either 16 or 32, to match the width of the Nios CPU. Also, nios-build looks at this value to set the appropriate compiler and assembler options when building.

Nios Program Structure

In the typical case of a C program built with nios-build, the following table shows the memory layout that is represented in the resultant S-record file.

Address, ascending	Contents
Base + 0x00	A simple preamble, consisting of a JUMP instruction to the symbol "_start", and the four characters 'N','i','o','s'. This is guaranteed to be at the beginning of the S-record output file. It comes from the library file "nios_jumptostart.o".
Base + 0x10	Your program's "main()" will be in here somewhere, as well as all your other routines, in order. The command that nios-build issues to the GNU linker has "nios_jumptostart.o" as its first file, and your C program as its second.
(A higher address)	A routine with the label "_start". This comes from the library file "nios_setup.o". It does some initialization, and then calls "main()".
(A higher address)	Two routines for handling "register window underflow" and "register window overflow", which are required by the Nios embedded processor to execute calling chains that are arbitrarily deep. These come from the library file "nios_cwpmanager.o".
(A higher address)	Any other Nios library routines that your program references. The linker extracts only those that are used from the file "libnios32.a", and includes them in the final program.
(A higher address)	Any read-only data from your program, such as strings or numeric constants.
(A higher address)	Any static variables in your program.

Nios Library Routines

The SDK for your Nios system will have a library built called libnios32.a (for 32-bit Nios system) or libnios16.a (for a 16-bit Nios system); either will be referred to in this document as the Nios library. The routines available in it will vary depending on the peripherals in the particular Nios system. This section describes the routines that are always present, as well as the optional peripheral routines.

C Runtime Support

Before a compiled program is run, certain initializations must take place. When nios-build is used to compile and link a program, the first routine executed is "_start", which performs this initialization and then calls the "main()" routine. Furthermore, the standard C libraries rely on several low-level platform-specific routines.

The following table lists the low-level C runtime support provided by the Nios library, always present in the Nios library:

Routine	Source File	Description
_start	nios_setup.s	Performs initialization prior to calling main().
_exit	nios_cstubs.s	Execute a JMP to "nasys_reset_address".
_sbrk	nios_cstubs.s	Increments "RAMLimit" by the requested amount and returns the previous value for it, unless the new value would be within 256 bytes of the current stack pointer, in which case it returns 0. This is the low-level routine used by malloc() to allocate more heap space.
Isatty	nios_cstubs.s	Returns "1", indicating to the C library that there is a tty.
_close	nios_cstubs.s	Returns "0"; not used by Nios software without a file system, but necessary to link.
_fstat	nios_cstubs.s	Returns "0"; not used by Nios software without a file system, but necessary to link.
_kill	nios_cstubs.s	Returns "0"; not used by Nios software without a file system, but necessary to link.
_getpid	nios_cstubs.s	Returns "0"; not used by Nios software without a file system, but necessary to link.
_read	nios_cstubs.s	Calls nr_uart_rxchar() to read a single character from a UART. The "fd" parameter is treated as the base address of a UART.
_write	nios_cstubs.s	Call nr_uart_txchar() to print characters to a UART. The "fd" parameter is treated as the base address of a UART. This has the useful effect of allowing the routine fprintf() to print to any UART, by passing a UART address in place of the file handle argument.
__mulsi3	nios_math1.s	This routine overrides the standard signed 32-bit multiplication routine in the GNU C library. It is faster than the standard routine, and uses the MUL or MSTEP instructions (if present), and does not use a register window level. It uses more code space than the standard routine.
__mulhi3	nios_math1.s	This routine overrides the standard unsigned 32-bit multiplication routine in the GNU C library. It is faster than the standard routine, and uses the MUL or MSTEP instructions (if present), and does not use a register window level. It uses more code space than the standard routine.

_start

The first code executed by a Nios program is the preamble's jump to _start. The second code executed is the _start code. Before a C program can run, various initialization must be performed. The _start code does this. The initialization consists of the following steps:

1. Initialize the stack pointer to “nasys_stack_top”.
2. Zero program storage between “__bss_start” and “_end”.
3. Set an internal variable named “RAMLimit” to “_end” (malloc claims memory upwards from here).
4. Optionally install the CWP Manager.
5. Optionally call the C++ static constructors.
6. Executes a CALL to the routine “main()”, which normally is the main entry point of your C routine.
7. If “main()” should happen to return, its return value is ignored, and a TRAP 0 is executed. This usually results in restarting the monitor.

System-Level Services

The following system-level service routines are always present in the Nios library, and are called automatically unless disabled in the Makefile.

Interrupt Service Routine Handler

The Nios processor allows up to 64 prioritized, vectored interrupts (numbered 0 to 63). The lower the interrupt number the higher the priority. Interrupt vectors 0 through 15 are reserved for system services, leaving 48 interrupt vectors for user applications.

For details on Nios CPU exception handling, refer to the “Exceptions” section of the *Nios Embedded Processor Programmer's Reference Manual*.

nr_installuserisr

Syntax:	void nr_installuserisr(int trapNumber, void *ISRProcedure, int context);

Parameters:	trapNumber - the exception number to be associated with a user service routine
	
	ISRProcedure - a routine you supply, which has a prototype of:
	
	`typedef void (*nios_isrhandlerproc)(int context);`
	
	`context - a value that will be passed to the routine specified by isrProcedure.`

Description:	This routine installs an interrupt service routine for a specific exception number. If nr_installuserisr() is used to set up the exception handler, then the exception handler can be an ordinary C routine. Note: If you manipulate the vector table directly, you must completely understand the mechanisms of the Nios register window, control registers, etc. The exception handler will receive the context value as its only argument when called. The trap handler is still responsible for clearing any interrupt condition for a peripheral that it services.
Include:	nios.h

Current Window Pointer Manager

A detailed understanding the current window pointer (CWP) Manager is not required to write Nios software, but it becomes part of the final program, and is briefly described in the following section.

The Nios embedded processor contains 128, 256, or 512 general-purpose registers. Of these, exactly 32 are visible to the software at any particular moment. They are named %r0-%r31, and can also be referred to as %g0-%g7 (global), %o0-%o7 (out), %L0-%L7 (local), and %i0-%i7 (in).

Which 32 registers are visible is determined by the CWP bits of the Nios STATUS register (%ctl0, readable via the RDCTL instruction). See the *Nios Programmers Reference Manual* for more details.

Subroutines execute a SAVE instruction, which decrements the CWP by one, revealing 16 "new" registers. The “caller’s” %o registers are visible to the “callee” as %i registers. Eventually, however, there are no more registers to reveal, and the CWP is pointing to the lowest registers.

This is where the CWP manager comes in: when a SAVE is executed, it induces a software exception that is handled by the CWP manager's underflow handler. This handler saves every register onto the stack, and repositions the CWP back to the top.

Conversely, subroutines execute a RESTORE instruction when they are ready to return. If the CWP is already at the top of the register file, a trap is induced, which is handled by the CWP Manager's overflow handler. This handler restores the register contents from when they were saved by the corresponding underflow condition.

nr_installcwpmanager

Syntax:	void nr_installcwpmanager(void);
Parameters:	none
Description:	This routine is called automatically by _start() if the library was built with NIOS_USE_CWPMGR = 1. It installs service routines for the Nios CPU underflow and overflow exceptions.
Include:	nios.h

General-Purpose System Routines

The following routines perform general-purpose operations.

nr_delay

Syntax:	void nr_delay(int milliseconds);
Parameters:	milliseconds - Length of time, in milliseconds, for program execution to be suspended.
Description:	Causes program execution to pause for the number of milliseconds specified in milliseconds. It executes a tight countdown loop during this time.
Include:	nios.h

nr_zerorange

Syntax:	void nr_zerorange(char *rangestart, int rangeByteCount);
Parameters:	rangestart - first byte to set to zero rangeByteCount - number of consecutive byte to set to zero
Description:	Writes zero to range of bytes in memory starting at rangeStart and counting up to rangeByteCount
Include:	nios.h

High-Level C Support

These routines are always present in the Nios library, unless disabled in the Makefile.

Routine	Source File	Description
Printf	nios_printf.c	This version of the standard C printf() function omits all support for floating point numbers, and supports only the %d, %x, %X, %c, and %s formats. The Nios library includes this version of printf() because the standard library routine takes about 40k of Nios code. This large footprint is primarily for floating point support, and the Nios CPU is often used for applications that do not require floating point. The Nios library version of printf() is about 1k of Nios code.
Sprintf	nios_printf.s	Uses the Nios library's version of printf() to print to a string in memory.

Notes:

Nios Peripheral Routines

The tables below contain lists of C (or assembly) call-able peripheral routines that are automatically added to the custom SDK library when the corresponding peripherals are included in the Nios system design.

PIO Routine	Description
nr_pio_showhex	Sends low byte to PIO named na_seven_seg_pio.

SPI Routine	Description
nr_spi_rxchar	Reads a character from the SPI peripheral whose address is passed as an argument.
nr_spi_txchar	Sends a single character to the SPI peripheral whose address is passed as an argument.

Timer Routine	Description
nr_timer_milliseconds	Installs an interrupt service routine and returns zero the first time it is called. For each subsequent call, it returns the number of milliseconds that have elapsed since the first call.

UART Routine	Description
nr_uart_rxchar	Reads a character from the UART whose address is passed as an argument.
nr_uart_txcr	Sends a carriage return and line feed to the UART at address *nasys_printf_UART*.
nr_uart_txchar	Sends a single character to the UART whose address is passed as an argument.
nr_uart_txhex	Prints an integer value, in hexadecimal, to the UART at address *nasys_printf_UART*.
nr_uart_txhex16	Prints the value of a short integer, in hexadecimal, to the UART at address *nasys_printf_UART*.
uart_txchar32	Prints the value of a long integer, in hexadecimal, to the UART at address *nasys_printf_UART*.
nr_uart_txstring	Prints a null-terminated string to the UART at address *nasys_printf_UART*.

Nios PIO

Register Map		
A1..A0	**Register Name**	**Variable Size 1..32 bits**
0	Data-in[1]	Data Value currently on PIO inputs (read).
	Data-out[3,a]	New value to drive on PIO outputs (write).
1	DataDir[2]	Data Direction (optional): Individual control for each port bit. 1=out, 0=in.
2	Int Mask[2]	Interrupt Mask (optional): Per-bit IRQ enable/disable.
3	Edge Capture[3,b]	Edge Capture (optional): Per-bit synchronous edge detect-and-hold.

Notes
(1) Read-only value.
(2) Host-written control value. Can be read back at any time.
(3) Write-event register. A write operation to this address causes an event in the device.
(a) A write-operation to the Data-out register changes the value on the PIO output pins, if any.
(b) A write-operation to the Edge Capture register clears all bits in the register 0.

Software Data Structure

```
typedef volatile struct
{
  int np_piodata;           // read/write, up to 32 bits
  int np_piodirection;      // write/readable, up to 32 bits, 1->output
                            // bit
  int np_piointerruptmask;  // write/readable, up to 32 bits, 1->enable
                            // interrupt
  int np_pioedgecapture;    // read, up to 32 bits, cleared by any write
} np_pio;
```

Example: Direct access to PIO

```
void TurnOnLEDs(void)
{
  // the reference design has a PIO named na_led_pio

  na_led_pio->np_piodirection = 3; // Set direction: output
  na_led_pio->np_piodata = 0;      // both LEDs off
  nr_delay(1000);                  // wait 1 second
  na_led_pio->np_piodata = 1;      // turn on first led
  nr_delay(1000);                  // wait 1 second
  na_led_pio->np_piodata = 3;      // both LEDs on
}
```

PIO Peripheral Routines

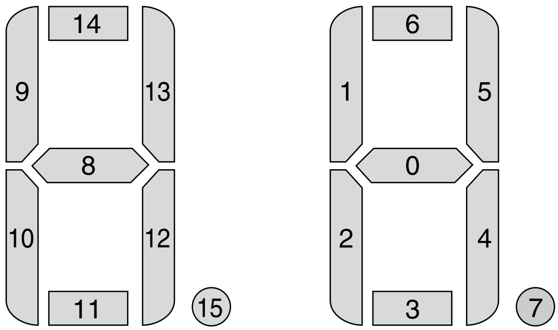
The PIO routines are present in the Nios library if there are one or more PIOs present in the Nios system.

nr_pio_showhex

Syntax: void nr_pio_showhex(int value);

Parameters: value - Data to be sent to seven-segment display.

Description: This routine assumes that a 16-bit wide PIO named "na_seven_seg_pio" is attached to a two-digit seven-segment display, in which segments are illuminated when the corresponding bits are driven low (zero). PIO bits are assigned to the seven-segment display elements as shown below:



Include: nios.h

Example:

```
#include "nios.h"

void main(void)
{
    int c;

    printf("Please enter a character:\n");

    while((c = nr_uart_rxchar(0)) == -1);
    //wait for valid input

    printf("Your character is:\t%c\n", c);
}
```

Nios SPI

Register Map (Master)

A2..A0	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RxData[1]	Rx Data(n-1..0)															
1	TxData[2]	TxData(n-1..0)															
2	Status[3]								E*	RRDY	TRDY	TMT	TOE*	ROE*			
3	Control[4]								iE*	iRRDY	iTRDY	iTMT	iTOE*	iROE*			
4	Reserved																
5	Select[5]	Slave Select Mask															

Notes
(1) Read-only value.
(2) Write-event register. A write operation to this address causes an event in the device.
(3) A write operation to the Status register clears the following bits: ROE, TOE, E.
(4) Nios CPU-written control value. Can be read back at any time.
(5) Write/read register. Bit mask for slave addressing.

Register Map (Slave)

A2..A0	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RxData[1]	Rx Data(n-1..0)															
1	TxData[2]	TxData(n-1..0)															
2	Status[3]								E*	RRDY	TRDY	TMT	TOE*	ROE*			
3	Control[4]								iE*	iRRDY	iTRDY	iTMT	iTOE*	iROE*			

Notes
(1) Read-only value.
(2) Write-event register. A write operation tothis address causes an event in the device.
(3) A write operation tot he Status register clears the following bits: ROE, TOE, E.
(4) Nios CPU-written control value. Can be read back at any time.

Software Data Structure

```
typedef volatile struct
{
     int np_spirxdata;       // Read-only, 1-16 bit
     int np_spitxdata;       // Write-only, 1-16 bit
     int np_spistatus;       // Read-only, 9-bit
     int np_spicontrol;      // Read/Write, 9-bit
     int np_spireserved;     // reserved
     int np_spislaveselect;  // Read/Write, 1-16 bit, master only
} np_spi;
```

SPI Routines

The Serial Peripheral Interface (SPI) routines are present in the Nios library if there are one or more SPI peripherals present in the Nios system.

nr_spi_rxchar

Syntax:	int nr_spi_rxchar(np_spi *pSPI);
Parameters:	pSPI - Pointer to the SPI peripheral.
Description:	Reads a character from the SPI peripheral whose address is passed as pSPI. If there is no character waiting, returns -1. If zero is passed for the peripheral address, reads a character from the default SPI memory location *nasys_printf_uart* (defined in nios.h).
Include:	nios.h

nr_spi_txchar.

Syntax:	int nr_spi_txchar(int i, np_spi *pSPI);
Parameters:	i - character to be sent pSPI - Pointer to the SPI peripheral.
Description:	Sends a single character, i, to the SPI peripheral whose address is passed as pSPI. If zero is passed for the peripheral address, sends character to the SPI peripheral at the default memory location *nasys_printf_uart* (defined in nios.h).
Include:	nios.h

Nios Timer

Register Map																	
A2..A0	**Register Name**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Status[1,2]															Run[3]	TO*
1	Control													Stop	Start	Cont	iTO
2	Period(L)	Timeout Period (bits15 : 0)[4]															
3	Period(H)	Timeout Period (bits 31 : 16)															
4	Snap(L)5	Timeout Counter Snapshot (bits 15 : 0)															
5	Snap(H)5	Timeout Counter Snapshot (bits 31 : 16)															

Notes

(1) Read-only value.

(2) Host-written control value. Can be read back at any time.

(3) Write-event register. A write operation to this address causes an event in the device.

(4) A write-operation tothe Status register clears the TO bit.

(5) A write-operation to either the Sanp(L) or Snap(H) registers update both registers with a coherent snapshot of the current internal-counter value.

Software Data Structure

```
typedef volatile struct
{
    int np_timerstatus;  // read only, 2 bits (any write to clear TO)
    int np_timercontrol; // write/readable, 4 bits
    int np_timerperiodl; // write/readable, 16 bits
    int np_timerperiodh; // write/readable, 16 bits
    int np_timersnapl;   // read only, 16 bits
    int np_timersnaph;   // read only, 16 bits
} np_timer;
```

Example: Direct access to Timer

```
#include "nios.h"

int main(void)
        {
        int t = 0;

        // Set timer for 1 second
        na_timer1->np_timerperiodl = (short)(nasys_clock_freq & 0x0000ffff);
        na_timer1->np_timerperiodh = (short)((nasys_clock_freq >> 16) & 0x0000ffff);

        // Set timer running, looping, no interrupts
        na_timer1->np_timercontrol = np_timercontrol_start_mask + np_timercontrol_cont_mask;

        // Poll timer forever, print once per second
        while(1)
                {
                if(na_timer1->np_timerstatus & np_timerstatus_to_mask)
                        {
                        printf("A second passed! (%d)\n",t++);

                        // Clear the to (timeout) bit
                        na_timer1->np_timerstatus = 0; // (any value)
                        }
                }
        }
```

Timer Peripheral Routines

The timer routines are present in the Nios library if there is one or more timer peripheral present in the Nios system.

nr_timer_milliseconds

Syntax:	int nr_timer_milliseconds(void);
Parameters:	None
Description:	This routine requires the existence of a timer called `timer1`, with a base address defined by *na_timer1* and an interrupt number defined by *na_timer1_irq*. The first time this routine is called, it installs an interrupt service routine for the timer, and returns zero. For each subsequent call, the number of milliseconds that have elapsed since the first call is returned.
Include:	nios.h

Nios UART

Register Map																	
A2..A0	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RxData[1]									Rx Data							
1	TxData[2]									TxData							
2	Status[3]								E*	RRDY	TRDY	TMT	TOE*	ROE*	BRK*	FE*	PE*
3	Control[4]								iE*	iRRDY	iTRDY	iTMT	iTOE*	iROE*	iBRK*	iFE*	iPE*
4	Divisor	Baud Rate Divisor (optional)															

Notes

(1) Read-only value.
(2) Write-event register. A write operation to this address causes an event in the device.
(3) A write-operation to the Status register clears these bits: E, TOE, ROE, BRK, FE, PE.
(4) Host-written control value. Can be read back at any time.

Software Data Structure:

```
typedef volatile struct
{
  int np_uartrxdata;  // Read-only, 8-bit
  int np_uarttxdata;  // Write-only, 8-bit
  int np_uartstatus;  // Read-only, 9-bit
  int np_uartcontrol; // Read/Write, 9-bit
  int np_uartdivisor; // Read/Write, 16-bit, optional
} np_uart;
```

UART Peripheral Routines

The UART routines are present in the Nios library if there are one or more UARTs present in the Nios system.

nr_uart_rxchar

Syntax: int nr_uart_rxchar(np_uart *uartBase);

Parameters: uartBase - Pointer to the UART peripheral.

Description: Reads a character from the UART peripheral whose address is passed in uartBase. If there is no character waiting, return -1. If zero is passed for the peripheral address, reads a character from the UART at location nasys_printf_uart (nios_map.h).

Include: nios.h

Example:

```
#include "nios.h"

void main(void)
{
    int c;

    printf("Please enter a character:\n");

   while((c = nr_uart_rxchar(0)) == -1); //wait for valid
    input

    printf("Your character is:\t%c\n", c);
}
```

nr_uart_txchar.

Syntax: int nr_uart_txchar(int c, np_uart *uartBase);

Parameters: c - Character to be sent.

uartBase - Pointer to the UART peripheral.

Description: Sends a single character, c, to the UART peripheral whose address is passed as *uartBase*. If zero is passed for the peripheral address, sends character to the UART at location *nasys_printf_uart* (defined in nios.h).

Include: nios.h

Example:

```
#include "nios.h"

#define kLineWidth 77
#define kLineCount 100

void SendLots(void)
{
    char c;
    int i,j;
    int mix;

    printf("\n\nPress character, or <space> for mix: ");
    while((c = nr_rxchar(0)) < 0);

    printf("%c\n\n",c);

    // Don't show unprintables

    if(c < 32)
        c = '.';

    mix = c==' ';

    for(i = 0; i < kLineCount; i++)
    {
        for(j = 0; j < kLineWidth; j++)
        {
            if(mix)
            {
                c++;
                if(c >= 127)
                   c = 33;
            }
            nr_uart_txchar(c,0);
            //send character to UART
        }
        nr_uart_txcr();
        //send carriage return and new line

    }
    printf("\n\n");
}
```

nr_uart_txcr

Syntax:	int nr_uart_txcr(void);
Parameters:	None
Description:	Sends a carriage return and line feed to the UART at location *nasys_printf_uart* (defined in nios.h).
Include:	nios.h

nr_uart_txhex

Syntax:	int nr_uart_txhex(int x);
Parameters:	x - Integer value to be sent to UART.
Description:	Prints the integer value of *x* in hexadecimal to the UART at location *nasys_printf_uart* (defined in nios.h). This will be 4 characters (`0000` to `FFFF`) if run on a 16-bit Nios CPU, and 8 characters (`00000000` to `FFFFFFFF`) if run on a 32-bit Nios CPU.
Include:	nios.h

nr_uart_txhex16

Syntax: int nr_uart_txhex16(short x);

Parameters: x - 16-bit integer value to be sent to UART.

Description: Prints the 16-bit value of *x* in hexadecimal to the UART at location *nasys_printf_uart* (defined in nios.h). This will be 4 characters (`0000` to `FFFF`).

Include: nios.h

nr_uart_txhex32

Syntax: int nr_uart_txhex32(long x);

Parameters: x - 32-bit integer value to be sent to UART.

Description: Prints the 32-bit value of *x* in hexadecimal to the UART at location *nasys_printf_uart* (defined in nios.h). This will be 8 characters (`00000000` to `FFFFFFF`). This routine is not available on a 16-bit Nios CPU.

Include: nios.h

nr_uart_txstring

Syntax: int nr_uart_txstring(char *s);

Parameters: s - Pointer to null-terminated character string.

Description: Prints the null-terminated string *s* to the UART at location *nasys_printf_uart* (defined in nios.h).

Include: nios.h

Nios Software Development Utilities

The GNUPro software tools, included in the Nios development kit, contain a number of general-purpose software development utilities, including the bash command line shell. Bash is the environment in which Nios software is developed. For details on using bash, run bash and type "man bash" from the shell prompt.

Additionally, many Nios-specific utilities are included in the development kit for generating and debugging software. The following sections provide detailed descriptions of these utilities:

Nios Utility	Description
hexout2flash	Perl script that readies a Quartus II software .hexout file for writing to Nios development board flash memory
nios_bash	A startup script to set the bash environment for Nios development (bash shell)
nios-build	Perl script that performs compilation and assembly of source files, links to Nios library, generates .srec file
nios-convert	Perl script that converts .srec files to .mif or .dat file format
nios_csh	A startup script to set the bash environment for Nios development (C shell)
nios-elf-as	GNU assembler for Nios
nios-elf-gcc	GNU C/C++ compiler for Nios
nios-elf-gdb	GNU debugger for Nios
nios-elf-ld	GNU linker for Nios
nios-elf-nm	GNU tool to extract symbols from Nios object files
nios-elf-objcopy	GNU utility that converts linker output (.out) to S-records (.srec)
nios-elf-objdump	GNU tool to disassemble Nios object files
nios-elf-size	This tool produces a report of object file size, for code (text), data (data), and uninitialized storage (bss).
nios-run	A specialized terminal program for communicating with the Nios development board
nios-vimrc	A vim setup compatible with DOS files under Cygwin
srec2flash	Perl script that readies a .srec file for writing to Nios development board flash

Note

On-line documentation for the Cygwin GNUPro tools is available by choosing **Programs > Cygwin > Cygwin Documentation** (Windows Start Menu).

hexout2flash

Description: The Quartus II software and Max+Plus II software generate design files for download to an Altera® complex programmable logic device (CPLD). One design file format generated by these tools is a .hexout file. The hexout2flash script converts a .hexout file to a .flash file, suitable for writing to the flash device on the Nios GERMS monitor commands to erase a section of flash memory, and relocate the .hexout file to the erased section.

Refer to the *Nios Development Board Reference Manual* for details about the Nios development board.

Usage:

```
hexout2flash [options] <filename>[.hexout]
```

Options:

```
-b <base address> :Location in flash to write file,
                  :(default 0x180000)
--help            :Print help
```

Example: If your file is called “my_design.hexout”, you would execute the following commands:

```
hexout2flash my_design.hexout
```

hexout2flash converts my_design.hexout to my_design.flash

Download the .flash file to the development board by typing the following command:

```
nios-run my_design.flash
```

This step writes the design into flash memory at location 0x180000, and becomes the default booting design for the development board.

nios_bash

Description: A startup script that properly sets the bash shell environment for software development using nios-build. Nios-build requires two shell variables to exist and be exported. A normal Windows install sets this up for you automatically. The two shell variables are as follows:

```
set niosgnu = <location of Nios GNU tools>
```
By default this is /usr/altera/excalibur.

```
set niossdk = <location of Nios SDK>
```
By default this is /usr/altera/excalibur/nios-sdk.

Usage: Source this script from the .bash_profile at shell startup time. It adds a few paths and shell variables needed to use the Nios tools.

nios-build

Description: nios-build is a Perl script that invokes the appropriate tools to compile, assemble, and link Nios source code. It ensures that the standard C libraries and standard Nios libraries are linked against, and that the associated “include” paths are available. Most programs should compile with no command line options at all; reasonable defaults are in effect

nios-build will produce a file with the base name of the last source file on the command line, and the suffix “.srec”, ready for downloading to the GERMS monitor running on the Nios development board.

Source files are listed on the command line following the options. If only one source file is specified, nios-build will search the same directory for files with the same base name, and underscore extensions.

Files ending with .s or .asm are passed to nios-elf-asm. Files ending with .c are passed to nios-elf-gcc, and files ending with .o are passed to nios-elf-ld.

Usage:

```
nios-build [options] <sourcefile>.[sco]
```

Options:

```
-b <base address>    :Set base address of code
-m16                 :Generate code for Nios 16
-m32                 :Generate code for Nios 32 (default)
-as <quoted string> :Pass command line options to
assembler
-cc <quoted string> :Pass command line options to
                     :compiler
-ld <quoted string> :Pass command line options to linker
-d                   :Set NIOS_GDB=1 and generate
                     :debug script
-s                   :Silent mode (only print errors)
-l <file name>       :Include system library
-o <file name>       :Output file name
--help               :Print help
--help 1             :Print more help
```

Example:

```
nios-build foo.c bar.s
```

Multiple files listed in the command line, as shown above, will generate an executable file named bar.srec

```
nios-build helloworld.c
```

If there are files named “helloworld_2.c” and “helloworld_3.s” in the same directory, they will be included in the build, and the result will be named helloworld.srec.

nios-convert

Description: Perl script that converts files from one format to another. Source files can be `.srec` or `.mif`; destination files can be `.mif` or `.dat`.

Destination files will be named the same as the source file if no destination file name is specified.

Usage: `nios-convert [options] <sourceFile> [destFile]`

Options:

```
--lanes=x      :break into multiple output files
               :lane_0 .. _lane_(x-1) appended
--width=x      :set output width to 8, 16, or 32
--oformat=f    :format can be mif or dat
--comments=b   :comments in mif file enabled(1) or
               :disabled(0).
               :(default is enabled)
--help
```

Example: `nios-convert bootcode.srec bootcode.mif`

Converts file `bootcode.srec` to `bootcode.mif`.

nios_csh

Description: A startup script that properly sets the C shell environment for software development using nios-build.

Usage: Source this script from the .login at shell startup time.

Example: `source /usr/altera/excalibur/nios-sdk/nios_csh`

If the .../altera/ directory is at some location other than /usr/altera, you must assign that location to the shell variables “altera” as follows:

```
set altera = /downloads/altera
source /downloads/altera/excalibur/nios-sdk/nios_bash
```

nios-elf-as

Description: Nios assembler. Produces a relocatable object file from assembly language source code. The object file contains the binary code and debug symbols.

If you use nios-build to generate executable code from assembly source, nios-elf-as is invoked automatically. It may be useful, however, to have a working knowledge of the assembler command line options to help optimize your assembly source code.

Usage: `nios-elf-as [option...] [asmfile...]`

Options:

```
-a[sub-option...]       :turn on listings Sub-options
                        :[default hls]
  c                     :omit false conditionals
  d                     :omit debugging directives
  h                     :include high-level source
  l                     :include assembly
  m                     :include macro expansions
  n                     :omit forms processing
  s                     :include symbols
  L                     :include line debug statistics
  =file                 :set listing file name
                        :(must be last sub-option)
-D                      :produce assembler debugging
                        :messages
--defsym SYM=VAL        :define symbol SYM to given value
-f                      :skip whitespace and comment
                        :preprocessing
--gstabs                :generate STABS debugging
                        :information
--gdwarf2               :generate DWARF2 debugging
                        :information
--help                  :show this message and exit
-I DIR                  :add DIR to search list for
                        :.include directives
-J                      :don't warn about signed
                        :overflow
-K                      :warn when differences altered
                        :for long displacements
-L,--keep-locals        :keep local symbols
                        :(e.g. starting with 'L')
-M,--mri                :assemble in MRI compatibility
                        :mode
--MD FILE               :write dependency information in
                        :FILE (default none)
-nocpp                  :ignored
-o OBJFILE              :name the object-file output
                        :OBJFILE (default a.out)
-R                      :fold data section into text
                        :section
--statistics            :print various measured
                        :statistics from execution
--strip-local-absolute :strip local absolute symbols
```

Options (con't):

```
--traditional-format   :use same format as native
                       :assembler when possible
--version              :print assembler version number
                       :and exit
-W  --no-warn          :suppress warnings
--warn                 :don't suppress warnings
--fatal-warnings       :treat warnings as errors
--itbl INSTTBL         :extend instruction set to
                       :include instructions matching
                       :the specifications defined in
                       :file INSTTBL
-w                     :ignored
-X                     :ignored
-Z                     :generate object file even
                       :after errors
--listing-lhs-width    :set the width in words of the
                       :output data column of the
                       :listing
--listing-lhs-width2   :set the width in words of the
                       :continuation lines of the
                       :output data column; ignored
                       :if smaller than the width of
                       :the first line
--listing-rhs-width    :set the max width in characters
                       :of the lines from the source
                       :file
--listing-cont-lines   :set the maximum number of
                       :continuation lines used for the
                       :output data column of the
                       :listing

NIOS specific command line options:

-m16                    :Nios-16 processor (16-bit)
-m32                    :Nios-32 processor (32-bit)
```

Help: For more details on using the GNU assembler refer to the on-line documentation by choosing **Programs** > **Cygwin** > **Cygwin Documentation** > **Using** as (Windows Start Menu).

nios-elf-gcc

Description: The GNU compiler invokes the necessary utilities as follows:

cpp
C preprocessor that processes all the header files and macros that the target requires.

gcc
The compiler that produces assembly language code from the processed C files.

as
The assembler that produces binary code from the assembly language source code and puts it in an object file.

ld
The linker that binds the code to addresses, links the startup file and libraries to the object code, and produces the executable binary image.

If you use nios-build to generate executable code, nios-elf-gcc is invoked automatically. It may be useful, however, to have a working knowledge of the C compiler command line options to help optimize your C code.

Usage: `nios-elf-gcc [options] file…`

Options:

```
-pass-exit-codes         :Exit with highest error code
                         :from a phase
--help                   :Display this information
                         :(Use '-v --help' to display
                         :command
                         :line options of sub-processes)
-dumpspecs               :Display all of the built in
                         :spec strings
-dumpversion             :Display the version of the
                         :compiler
-dumpmachine             :Display the compiler's target
                         :processor
-print-search-dirs       :Display the directories in the
                         :compiler's search path
-print-libgcc-file-name  :Display the name of the
                         :compiler's companion library
-print-file-name=<lib>   :Display the full path to
                         :library <lib>
-print-prog-name=<prog>  :Display the full path to
                         :compiler component <prog>
-print-multi-directory   :Display the root directory for
                         :versions of libgcc
```

Options (con't):

```
-print-multi-lib          :Display the mapping between
                          :command line options and
                          :multiple library search
                          :directories
-Wa,<options>             :Pass comma-separated <options>
                          :on to the assembler
-Wp,<options>             :Pass comma-separated <options>
                          :on to the preprocessor
-Wl,<options>             :Pass comma-separated <options>
                          :on to the linker
-Xlinker <arg>            :Pass <arg> on to the linker
-save-temps               :Do not delete intermediate
                          :files
-pipe                     :Use pipes rather than
                          :intermediate files
-time                     :Time the execution of each
                          :subprocess
-specs=<file>             :Override builtin specs with
                          :the contents of <file>
-std=<standard>           :Assume that the input sources
                          :are for <standard>
-B <directory>            :Add <directory> to the
                          :compiler's search paths
-b <machine>              :Run gcc for target <machine>,
                          :if installed
-V <version>              :Run gcc version number
                          :<version>, if installed
-v                        :Display the programs invoked
                          :by the compiler
-E                        :Preprocess only; do not
                          :compile, assemble or link
-S                        :Compile only; do not assemble
                          :or link
-c                        :Compile and assemble, but do
                          :not link
-o <file>                 :Place the output into <file>
-x <language>             :Specify the language of the
                          :following input files
                          :Permissable languages include
                          :c c++ assembler none
                          :'none' means revert to the
                          :default behaviour of guessing
                          :the language based on the
                          :file's extension

Options starting with -g, -f, -m, -O or -W are
automatically passed on to the various sub-processes
invoked by nios-elf-gcc. In order to pass other options
on to these processes the -W<letter> options must be used.
```

Help: For more details on using the GNU compiler refer to the on-line documentation by choosing **Programs > Cygwin > Cygwin Documentation > Using GNU CC** (Windows Start Menu).

nios-elf-gdb

Description: The GNU debugger, GDB, lets you see what is going on inside another program while it executes-or what another program was doing at the moment it stopped. GDB can do four main kinds of things to debug software.

- Start the program and specifying anything that might affect its behavior.
- Stop the program based on a set of specific conditions.
- Examine what has happened once the program has stopped.
- Change the program to fix bugs and continue testing.

You can use GDB to debug programs written in assembly, C and C++.

Usage: To debug a program using nios-build and nios-elf-gdb, you must do two things:

1. Add a line with “NIOS_GDB_SETUP” as the first statement in your main() routine.
2. Use nios-build with the “-d” command line option.

nios-build produces a file with the extention “.gdb”. This file is a shell script for downloading your program, and then running nios-elf-gdb.

Options:

```
--[no]async             :Enable (disable) asynchronous
                        :version of CLI
-b BAUDRATE             :Set serial port baud rate used
                        :for remote debugging.
--batch                 :Exit after processing options.
--cd=DIR                :Change current directory to DIR.
--command=FILE          :Execute GDB commands from FILE.
--core=COREFILE         :Analyze the core dump COREFILE.
--dbx                   :DBX compatibility mode.
--directory=DIR         :Search for source files in DIR.
--epoch                 :Output information used by epoch
                        :emacs-GDB interface.
--exec=EXECFILE         :Use EXECFILE as the executable.
--fullname              :Output information used by
                        :emacs-GDB interface.
--help                  :Print this message.
--interpreter=INTERP    :Select a specific
                        :interpreter/user interface
--mapped                :Use mapped symbol files if
                        :supported on this system.
--nw                    :Do not use a window interface.
--nx                    :Do not read gdb.ini file.
--quiet                 :Do not print version number on
                        :startup.
--readnow               :Fully read symbol files on first
                        :access.
```

Options (con't)

```
--se=FILE               :Use FILE as symbol file and
                        :executable file.
--symbols=SYMFILE       :Read symbols from SYMFILE.
--tty=TTY               :Use TTY for input/output by the
                        :program being debugged.
--version               :Print version information and
                        :then exit.
-w                      :Use a window interface.
--write                 :Set writing into executable and
                        :core files.
--xdb                   :XDB compatibility mode.

For more information, type "help" from within GDB, or
consult the GDB manual (available as on-line info or a
printed manual).
```

Help

For more details on using the GNU compiler refer to the on-line documentation by choosing **Programs** > **Cygwin** > **Cygwin Documentation** > **Debugging with GDB** (Windows Start Menu).

nios-elf-ld

Description: The GNU linker resolves the code addresses and debug symbols, links the startup code and additional libraries to the binary code, and produces an executable binary image.

If you use nios-build to generate executable code, nios-elf-ld is invoked automatically. It may be useful, however, to have a working knowledge of the linker command line options.

Usage: `nios-elf-ld [options] file…`

Options:

```
-a KEYWORD                       :Shared library control
                                 :for HP/UX compatibility
-A ARCH, --architecture ARCH     :Set architecture
-b TARGET, --format TARGET       :Specify target for
                                 :following input files
-c FILE, --mri-script FILE       :Read MRI format linker
                                 :script
-d, -dc, -dp                     :Force common symbols to
                                 :be defined
-e ADDRESS, --entry ADDRESS      :Set start address
-E, --export-dynamic             :Export all dynamic
                                 :symbols
-EB                              :Link big-endian objects
-EL                              :Link little-endian
                                 :objects
-f SHLIB, --auxiliary SHLIB      :Auxiliary filter for
                                 :shared object symbol
                                 :table objects
-F SHLIB, --filter SHLIB         :Filter for shared object
                                 :symbol table
-g                               :Ignored
-G SIZE, --gpsize SIZE           :Small data size (if no
                                 :size, same as --shared)
-h FILENAME, -soname FILENAME    :Set internal name of
                                 :shared library
-l LIBNAME, --library LIBNAME    :Search for library
                                 :LIBNAME
-L DIRECTORY, --library-path DIRECTORY
                                 :Add DIRECTORY to library
                                 :search path
-m EMULATION                     :Set emulation
-M, --print-map                  :Print map file on
                                 :standard output
-n, --nmagic                     :Do not page align data
-N, --omagic                     :Do not page align data,
                                 :do not make text read
                                 :only
-o FILE, --output FILE           :Set output file name
-O                               :Optimize output file
-Qy                              :Ignored for SVR4
                                 :compatibility
-r, -i, --relocateable           :Generate relocateable
                                 :output
```

Options (con't):

```
-R FILE, --just-symbols FILE  :Just link symbols (if
                              :directory, same as
                              :--rpath)
-s, --strip-all               :Strip all symbols
-S, --strip-debug             :Strip debugging symbols
-t, --trace                   :Trace file opens
-T FILE, --script FILE        :Read linker script
-u SYMBOL, --undefined SYMBOL :Start with undefined
                              :reference to SYMBOL
-Ur                           :Build global
                              :constructor/destructor
                              :tables
-v, --version                 :Print version
                              :information
-V                            :Print version and
                              :emulation information
-x, --discard-all             :Discard all local symbols
-X, --discard-locals          :Discard temporary local
                              :symbols
-y SYMBOL, --trace-symbol SYMBOL
                              :Trace mentions of SYMBOL
-Y PATH                       :Default search path for
                              :Solaris compatibility
-z KEYWORD                    :Ignored for Solaris
                              :compatibility
-(, --start-group             :Start a group
-), --end-group               :End a group
-assert KEYWORD               :Ignored for SunOS
                              :compatibility
-Bdynamic, -dy, -call_shared  :Link against shared
                              :libraries
-Bstatic, -dn, -non_shared, -static
                              :Do not link against
                              :shared libraries
-Bsymbolic                    :Bind global references
                              :locally
--check-sections              :Check section addresses
                              :for overlaps (default)
--no-check-sections           :Do not check section
                              :addresses for overlaps
--cref                        :Output cross reference
                              :table
--defsym SYMBOL=EXPRESSION    :Define a symbol
--demangle                    :Demangle symbol names
--dynamic-linker PROGRAM      :Set the dynamic linker
                              :to use
--embedded-relocs             :Generate embedded relocs
--errors-to-file FILE         :Save errors to FILE
                              :instead of printing to
                              :stderr
-fini SYMBOL                  :Call SYMBOL at
                              :unload-time
--force-exe-suffix            :Force generation of file
                              :with .exe suffix
--gc-sections                 :Remove unused sections
                              :(on some targets)
```

Options (con'td):

```
--no-gc-sections              :Don't remove unused
                              :sections (default)
--help                        :Print option help
-init SYMBOL                  :Call SYMBOL at load-time
-Map FILE                     :Write a map file
--no-demangle                 :Do not demangle symbol
                              :names
--no-keep-memory              :Use less memory and more
                              :disk I/O
--no-undefined                :Allow no undefined
                              :symbols
--no-warn-mismatch            :Don't warn about
                              :mismatched input files
--no-whole-archive            :Turn off --whole-archive
--noinhibit-exec              :Create an output file
                              :even if errors occur
--oformat TARGET              :Specify target of output
                              :file
-qmagic                       :Ignored for Linux
                              :compatibility
--relax                       :Relax branches on
                              :certain targets
--retain-symbols-file FILE    :Keep only symbols listed
                              :in FILE
-rpath PATH                   :Set runtime shared
                              :library search path
-rpath-link PATH              :Set link time shared
                              :library search path
-shared, -Bshareable          :Create a shared library
--sort-common                 :Sort common symbols by
                              :size
--split-by-file               :Split output sections for
                              :each file
--split-by-reloc COUNT        :Split output sections
                              :every COUNT relocs
--stats                       :Print memory usage
                              :statistics
--task-link SYMBOL            :Do task level linking
--traditional-format          :Use same format as
                              :native linker
-Tbss ADDRESS                 :Set address of .bss
                              :section
-Tdata ADDRESS                :Set address of .data
                              :section
-Ttext ADDRESS                :Set address of .text
                              :section
--verbose                     :Output lots of
                              :information during link
--version-script FILE         :Read version information
                              :script
--version-exports-section SYMBOL
                              :Take esport symbols list
                              :from .exports, using
                              :SYMBOL as the version.
--warn-common                 :Warn about duplicate
                              :common symbols
```

Options (con'td):

```
--warn-constructors          :warn if global
                             :constructors/destructors
                             :are seen
--warn-multiple-gp           :Warn if the multiple GP
                             :values are used
--warn-once                  :Warn only once per
                             :undefined symbol
--warn-section-align         :Warn if start of section
                             :changes due to alignment
--whole-archive              :Include all objects from
                             :following archives
--wrap SYMBOL                :Use wrapper functions
                             :for SYMBOL
--mpc860c0 [=WORDS]          :Modify problematic
                             :branches in last WORDS
                             :(1-10, default 5) words
                             :of a page

nios-elf-ld: supported targets:
  elf32-nios
  elf32-little
  elf32-big
  srec
  symbolsrec
  tekhex
  binary
  ihex

nios-elf-ld: supported emulations:
  elfnios16
  elfnios32

nios-elf-ld: emulation specific options:
  no emulation specific options.
```

Help: For more details on using the GNU linker refer to the on-line documentation by choosing **Programs** > **Cygwin** > **Cygwin Documentation** > **Using ld** (Windows Start Menu).

nios-elf-nm

Description: Lists public symbols and their values from object files.

Usage: nios-elf-nm [options] [file...]

Options:

```
[-aABCDglnopPrsuvV]
[-t radix]
[--radix=radix]
[--target=bfdname]
[--debug-syms]
[--extern-only]
[--print-armap]
[--print-file-name]
[--numeric-sort]
[--no-sort]
[--reverse-sort]
[--size-sort]
[--undefined-only]
[--portability]
[-f {bsd,sysv,posix}]
[--format={bsd,sysv,posix}]
[--demangle]
[--no-demangle]
[--dynamic]
[--defined-only]
[--line-numbers]
[--version]
[--help]
```

Example:

```
nios-elf-nm hello_world.out > hello_world.nm
```

Creates a file `hello_world.nm` that contains a list of all symbols in the program.

```
hello_world.out:
000406b0 t CWPOverflowTrapHandler
000405fc t CWPUnderflowTrapHandler
000402d6 T PrivatePrintf
00040244 T RAMLimit
00040ae8 A __bss_start
000408ca T __divsi3
000408fc T __modsi3
00040796 T __mulhi3
00040796 T __mulsi3
00000001 a __nios32__
.
.
.
```

Help: For more details on using the GNU linker refer to the on-line documentation by choosing **Programs** > **Cygwin** > **Cygwin Documentation** > **Using binutils** > **nm** (Windows Start Menu).

nios-elf-objcopy

Description: Utility that converts executable binary files (.out) to S-records, which are suitable for ROM images and for download images to embedded systems.

If you use nios-build to generate executable code nios-elf-objcopy is invoked automatically.

Usage: nios-elf-objcopy <switches> in-file [out-file]

Options:

```
-I <bfdname>               :Assume input file is in
                           :format <bfdname>
-O <bfdname>               :Create an output file in
                           :format <bfdname>
-F <bfdname>               :Set both input and output
                           :format to <bfdname>
--debugging                :Convert debugging
                           :information, if possible
-p                         :Copy modified/access
                           :timestamps to the output
-j <name>                  :Only copy section <name>
                           :into the output
-R <name>                  :Remove section <name> from
                           :the output
-S                         :Remove all symbol and
                           :relocation  information
-g                         :Remove all debugging symbols
--strip-unneeded           :Remove all symbols not needed
                           :by relocations
-N <name>                  :Do not copy symbol <name>
-K <name>                  :Only copy symbol <name>
-L <name>                  :Force symbol <name> to be
                           :marked as a local
-W <name>                  :Force symbol <name> to be
                           :marked as a weak
--weaken                   :Force all global symbols to
                           :be marked as weak
-x                         :Remove all non-global symbols
-X s                       :Remove any compiler-generated
                           :symbols
-i <number>                :Only copy one out of every
                           :<number> bytes
-b <num>                   :Select byte <num> in every
                           :interleaved block
--gap-fill <val>           :Fill gaps between sections
                           :with <val>
--pad-to <addr>            :Pad the last section up to
                           :address <addr>
--set-start <addr>         :Set the start address to
                           :<addr>
--change-start <incr>      :Add <incr> to the start
                           :address
--change-addresses <incr>  :Add <incr> to LMA, VMA and
                           :start addresses
```

3

Utilities

Options (cont'd):

```
--change-section-address <name>{=|+|-}<val>
                                :Change LMA and VMA of
                                :section <name> by <val>
--change-section-lma <name>{=|+|-}<val>
                                :Change the LMA of section
                                :<name> by <val>
--change-section-vma <name>{=|+|-}<val>
                                :Change the VMA of section
                                :<name> by <val>
--[no-]change-warnings          :Warn if a named section
                                :does not exist
--set-section-flags <name>=<flags>
                                :Set section <name>'s
                                :properties to <flags>
--add-section <name>=<file>     :Add section <name> found
                                :in <file> to output
--change-leading-char           :Force output format's
                                :leading character style
--remove-leading-char           :Remove leading character
                                :from global symbols
--redefine-sym <old>=<new>      :Redefine symbol name <old>
                                :to <new>
-v --verbose                    :List all object files
                                :modified
-V --version                    :Display this program's
                                :version number
-h --help                       :Display this output
```

Help: For more details on using the GNU linker refer to the on-line documentation by choosing **Programs > Cygwin > Cygwin Documentation > Using binutils > objcopy** (Windows Start Menu).

nios-elf-objdump

Description: Displays information about one or more object files. The options control what particular information to display. This can be very useful if a user is not sure where their routines are being located, or are wondering what kind of code the compiler is producing.

Usage: `nios-elf-objdump <switches> file(s)`

Options:

```
At least one of the following switches must be given:

-a  --archive-headers    :Display archive header
                         :information
-f  --file-headers       :Display the contents of the
                         :overall file header
-p  --private-headers    :Display object format specific
                         :file header contents
-h  --[section-]headers  :Display the contents of the
                         :section headers
-x  --all-headers        :Display the contents of all
                         :headers
-d  --disassemble        :Display assembler contents of
                         :executable sections
-D  --disassemble-all    :Display assembler contents of
                         :all sections
-S  --source             :Intermix source code with
                         :disassembly
-s  --full-contents      :Display the full contents of
                         :all sections requested
-g  --debugging          :Display debug information in
                         :object file
-G  --stabs              :Display the STABS contents of
                         :an ELF format file
-t  --syms               :Display the contents of the
                         :symbol table(s)
-T  --dynamic-syms       :Display the contents of the
                         :dynamic symbol table
-r  --reloc              :Display the relocation entries
                         :inthe file
-R  --dynamic-reloc      :Display the dynamic relocation
                         :entries in the file
-V  --version            :Display this program's version
                         :number
-i  --info               :List object formats and
                         :architectures supported
-H  --help               :Display this information

The following switches are optional:
-b  --target <bfdname>   :Specify the target object
                         :format as <bfdname>
-m  --architecture <machine>
                         :Specify the target
                         :architecture as <machine>
-j  --section <name>     :Only display information for
                         :section <name>
```

3

Utilities

Options (cont'd):

```
-M  --disassembler-options <o>
                             :Pass text <o> on to the
                             :disassembler section
-EB --endian=big            :Assume big endian format when
                             :disassembling
-EL --endian=little          :Assume little endian format
                             :when disassembling
--file-start-context         :Include context from start of
                             :file (with -S)
-l  --line-numbers           :Include line numbers and
                             :filenames in output
-C  --demangle               :Decode mangled/processed
                             :symbol names
-w  --wide                   :Format output for more than
                             :80 columns
-z  --disassemble-zeroes :Do not skip blocks of zeroes
                             :when disassembling
--start-address <addr>       :Only process data whose
                             :address is >= <addr>
--stop-address <addr>        :Only process data whose
                             :address is <= <addr>
--prefix-addresses           :Print complete address
                             :alongside disassembly
--[no-]show-raw-insn         :Display hex alongside symbolic
                             :disassembly
--adjust-vma <offset>        :Add <offset> to all displayed
                             :section addresses

nios-elf-objdump: supported targets:
  elf32-nios
  elf32-little
  elf32-big
  srec
  symbolsrec
  tekhex
  binary
  ihex
```

Example:

```
nios-elf-objdump -D hello_world.out > hello_world.objdump
```

Disassembles the object file `hello_world.out` and creates a disassembly output file `hello_world.objdump` as shown below:

```
hello_world.out:     file format elf32-nios

Disassembly of section .text:

00040100 <nr_jumptostart>:
   40100:06 98       pfx %hi(0xc0)
   40102:40 35       movi %g0,0xa
   40104:00 98       pfx %hi(0x0)
   40106:40 6c       movhi %g0,0x2
   40108:c0 7f       jmp %g0
   4010a:00 30       nop
   4010c:4e 69       ext16d %sp,%o2
   4010e:6f 73       *unknown*
```

Example (cont'd):

```
00040110 <main>:
   40110:17 78        save %sp,0x17
   40112:4a 98        pfx %hi(0x940)
   40114:88 35        movi %o0,0xc
   40116:00 98        pfx %hi(0x0)
   40118:88 6c        movhi %o0,0x4
   4011a:04 98        pfx %hi(0x80)
   4011c:a1 36        movi %g1,0x15
   4011e:00 98        pfx %hi(0x0)
   40120:41 6c        movhi %g1,0x2
   40122:e1 7f        call %g1
   40124:00 30        nop
   40126:df 7f        ret
   40128:a0 7d        restore
   .
   .
   .
```

Help: For more details on using the GNU linker refer to the on-line documentation by choosing **Programs** > **Cygwin** >**Cygwin Documentation** > **Using binutils** > **objdump** (Windows Start Menu).

nios-elf-size

Description: This tool takes any number of ".out", ".o", or ".a" files, and produces a report of the sizes for code (text), data (data), and uninitialized storage (bss).

Usage: `nios-elf-size [options] [file...]`

Options:

```
[-ABdoxV]
[--format=berkeley|sysv]        :default is --
format=berkeley
[--radix=8|10|16]
[--target=bfdname]
[--version]
[--help]
```

Help: For more details on using the GNU linker refer to the on-line documentation by choosing **Programs > Cygwin > Cygwin Documentation > Using binutils > size** (Windows Start Menu).

nios-run

Description: Download code to Nios development board and perform terminal I/O.

Usage: `nios-run [option(s)] [filename]`

Options:

Example: `nios-run -p com2 hello_world.srec`

Downloads the executable file hello_world.srec to the development board via COM2.

srec2flash

Description: The GERMS monitor looks for code in flash memory at location 0x140000. If found, the code is executed.

The srec2flash utility takes code targeted for location 0x40100 (SRAM) and prepends a routine to copy itself from 0x140100 (FLASH) to 0x40100 (SRAM).

It also prepares the file to be written to Flash by prepending the necessary GERMS monitor commands to write the file into flash.

Usage: `srec2flash [options] <srec file> [filename]`

Example: `Srec2flash hello_world.srec`

Generates the file `hello_world.flash` (partial listing below):

```
# This file generated by srec2flash, part of
# the Nios SDK. This file contains a short
# program to run out of flash memory which
# copies the main program down to RAM, and
# executes it there.
#
# Original file: hello_world.srec
#
# Loader program
r0
#
# Erase flash sector 140000
#
# This address is checked by germsMon at startup
#
e140000
#
S21914000000980035009840 6DC07F00304E696F73089810349044
S2191400156E1134116F08981234926C005A50048074015A500455
S21914002A8174011E0140415E92043012E27EF387003021981009
S21914003F340098106CB2993135115E08981234926C3224D27FA0
S206140054003061
#
# Main program
#
r40100-140100
S013000068656C6C6F5F776F726C642E7372656376
S21904010006984035009840 6CC07F00304E696F7317784A988889
S21904011535009888 6C0498A1360098416CE17F0030DF7FA07D48
S21904012A17781298D95F1398DA5F1498DB5F1598DC5F1698DD09
S21904013F5F083316984937009849 6CCB3302980B050B986135AC
.
.
.
```

To burn FLASH on the development board use the nios-run utility as follows:

```
nios-run -x hello_world.flash
```

3 Utilities

Notes:

Appendix A: Command Summary

GERMS Monitor Syntax	Monitor	Description
G<base address>	G40000	GO - Execute a CALL instruction to the specified address.
E<base address>	E180000	Erase flash memory. If the address is within the range of the “flash” ROM, the sector containing that address will be erased.
R<from address>-<to address>	R0-180000	Offset the next download. The next S-record or I-Hex record downloaded will be stored offset by the range specified.
M<address>	M50000	Display memory starting from the address.
M<address>-<address>	M40000-40100	Display a range of memory. Pressing <CR> again will show the same number of bytes, starting where the last M command ended.
M<address>:<value> <value>…	M50000:1 2 3 4	Write successive 16-bit words to memory, until the end of line.
M<address>-<address>:<value>	M50000-50100:AA55	Fill a range of memory with a 16-bit word.
<CR>	<CR>	Display the next 64 bytes of memory.
S<S-record data>	S21840000 . . .	Write S-record to next memory location
:<I-hex record data>	:80000004 . . .	Write I-hex record to next memory location.
<ESC>	<ESC>	Restart the monitor.

Nios-Build

Usage: nios-build [options] files.[sco]

Example: nios-build hello.c

Command Line Options	Description
-b <base address>	Override the standard base address of code.
-m16	Generate code for Nios 16
-m32	Generate code for Nios 32 (default)
-as <quoted string>	Pass quoted string as command line options to assembler
-cc <quoted string>	Pass quoted string as command line options to compiler
-ld <quoted string>	Pass quoted string as command line options to linker

Nios-Run

Usage: `nios-run [option(s)] [filename]`

Example: `nios-run -p com2 hello_world.srec`

Command Line Options	Description
`-b <baud-rate>`	`sets the serial port baud rate (default = 115200)`
`-d`	`provides additional debugging information during download`
`-o <seconds>`	`quit after <seconds> seconds in terminal mode`
`-p <port-name>`	`specifies serial port (default = COM1:)`
`-s <millisecs>`	`specifies a per-character delay (useful for reluctant flash)`
`-t`	`enters terminal mode without downloading code`
`-x`	`exit immediately after downloading code`
`-z`	`shows timestamp for each line, useful for benchmarking`

Appendix B: Assembly Language Macros

The file nios_macros.s, located in the .../inc/ directory, provides a number of assembly language macros useful for low level programming and debugging. For details on assembly language programming, refer to the *Nios Embedded Processor Programmer's Reference Manual*.

Macro	Description
MOVIP %reg,value	MOVIP acts similarly to the Nios instruction MOVI, but allows any size constant. It automatically uses a combination of BGEN, MOVI, MOVHI, and PFX to load the value into the register. MOVIP will use as few instructions as possible (of those above) to load the value into the register. MOVIP can only be used with defined constants; it will generate an error if the constant is not defined at assembly time.
MOVIA %reg,value	Load a native-sized value into the register. The native word size is 16 or 32 bits; 16-bit or 32-bit Nios CPU, respectively. The value need not be defined at assembly time; the linker will fill in the value later.
ADDIP %reg,value	ADDIP acts similarly to ADDI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
SUBIP %reg,value	SUBIP acts similarly to SUBI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
CMPIP %reg,value	CMPIP acts similarly to CMPI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
ANDIP %reg,value	ANDIP acts similarly to ANDI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
ANDNIP %reg,value	ANDNIP acts similarly to ANDNI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
ORIP %reg,value	ORIP acts similarly to ORI, but will work for any 16-bit constant. It will not work for constants greater than 16 bits.
_BR address	_BR acts similarly to BR, but uses %g7 to load the target address. The target address is therefore not limited to the short branch range.
_BSR address	_BSR acts similarly to BSR, but uses %g7 to load the target address. The target address is therefore not limited to the short branch range.
nm_print string	Prints the quoted string to the default UART. Uses %o0 and %g registers.

Macro	Description
nm_println string	Exactly like nm_print, but prints the string followed by a carriage return and line feed.
nm_d_txchar char	This macro expands out to a large block of code that transmits a character to the default UART without altering any registers, or requiring the CWP to move. It does use stack space. Because this macro does not affect any registers or the CWP, it can be very useful for debugging interrupt handlers and low-level services, such as task switchers.
nm_d_txreg char1,char2,%reg	This macro expands out to a rather large block of code which transmits the two characters, followed by a the hexadecimal value of the register. It will print erroneous values for the stack pointer register. Because this macro does not affect any registers or the CWP, it can be very useful for debugging interrupt handlers and low-level services, such as task switchers.