My First Nios II Software Tutorial

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

TU-01003-1.1

Contents

About this Tutorial

This tutorial provides comprehensive information that will help you understand how to create an Altera® FPGA design and run it on your development board.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Information Type	Contact *(1)*
Technical support	www.altera.com/mysupport/
Technical training	www.altera.com/training/ custrain@altera.com
Product literature	www.altera.com/literature/
Altera literature services	literature@altera.com
FTP site	ftp.altera.com

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

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.

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

1. My First Nios II Software Design

May 2007

Introduction

The Nios® II processor core is a soft-core central processing unit (CPU) that you program (along with other hardware components that comprise the Nios II system) onto an Altera® field programmable gate array (FPGA). This tutorial introduces you to the basic software development flow for the Nios II processor. You will use a simple pre-generated Nios II standard hardware system and create a software program to run on it.

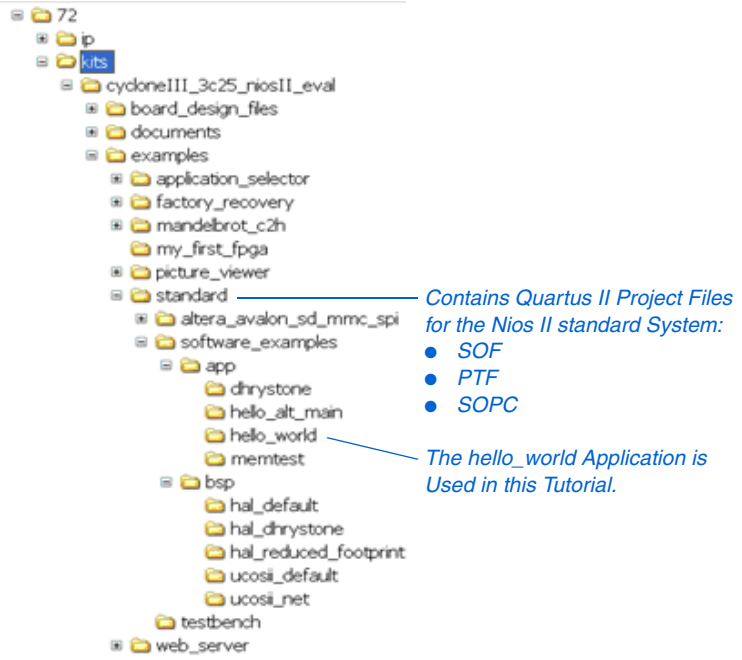
The example Nios II standard hardware system provides the following necessary components:

- Nios II processor core
- Off-chip SSRAM memory interface to store and run the software
- Universal serial bus (USB) serial link for communication between PC host and target hardware
- LED peripheral I/O (PIO)

Software and Hardware Requirements

This section assumes that you have already installed the Quartus® II design software, the Nios II Embedded Design Suite, and the Nios II Embedded Evaluation Kit CD-ROM software. Figure 1–1 shows the default installation directories.

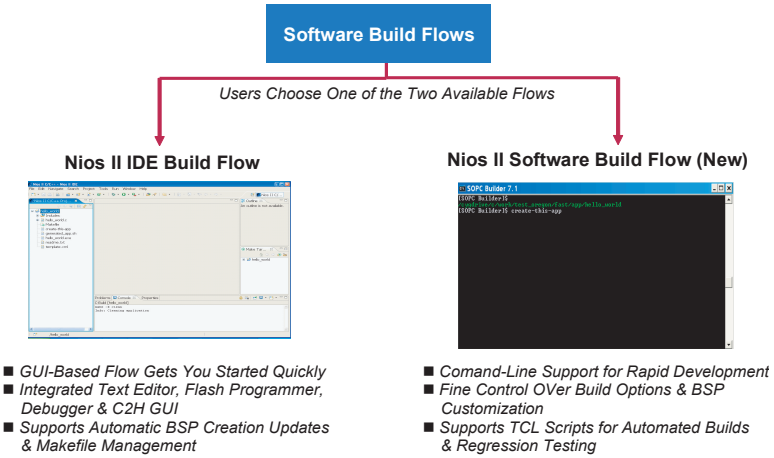
Figure 1–1. Default Nios II Embedded Evaluation Kit Installation Directory



Nios II Build Flows

The Nios II Embedded Design Suite provides two build flows: the Nios II Integrated Development Environment (IDE) build flow and the Nios II software build flow. See Figure 1–2.

Figure 1–2. Nios II Build Flows



The Nios II IDE build flow is an easy-to-use graphical user interface (GUI) that automates build and makefile management. The Nios II IDE integrates a text editor, debugger, the Nios II flash programmer, the Quartus II Programmer, and the Nios II C2H compiler GUI. The included example software application templates make it easy for new software programmers to get started quickly.

The Nios II Software build flow provides full control over build options using command-line tools. This flow allows you to create complex, custom board support packages that may include software (graphics libraries, stacks, file systems, etc.), custom device drivers, and custom build options. You can use scripts to automate the build process and to perform regression testing.

CAUTION

The two software build flows are not interchangeable: once you have begun a software project in a particular flow, you cannot switch to the other flow unless you re-create the project.

This tutorial describes how to use both flows. If you want to use the Nios II IDE build flow, go to the next section. To use the Nios II software build flow, go to "Nios II Software Build Flow" on page 1–17.

Nios II IDE Build Flow

In this section you will use the Nios II IDE to compile a simple C language software program to run on the Nios II standard system configured onto the FPGA on your development board. You will create a new software project, build it, and run it on the target hardware. You will also edit the project, re-build it, and set up a debug session.

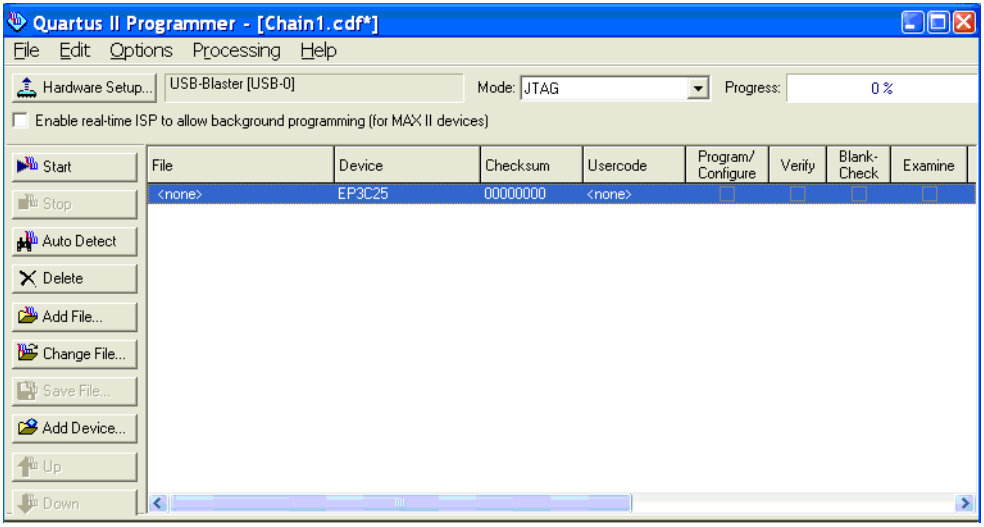
For a complete tutorial on using the Nios II IDE to develop programs, see the software development tutorial, which is available in the IDE help.

Download Hardware Design to the Target FPGA

The software that you build will be executed by a Nios II processor-based system in an FPGA. Therefore, the first step is to configure the device on your target hardware with the pre-generated Nios II standard hardware system. Download the FPGA configuration file, the SRAM Object File (.**sof**), by performing the following steps:

1. Connect the board to the host computer via the USB download cable.
2. Apply power to the board.
3. Start the Nios II IDE. On Windows computers, choose **All Programs > Altera > Nios II EDS** <*version*> **> Nios II IDE** <*version*> in the Windows Start menu.
4. If you the welcome page appears, click **Workbench**.
5. Choose **Tools > Quartus II Programmer**.
6. Click **Auto Detect**. The device EP3C25 should be detected.
7. Click the top row to highlight it. See Figure 1–3.

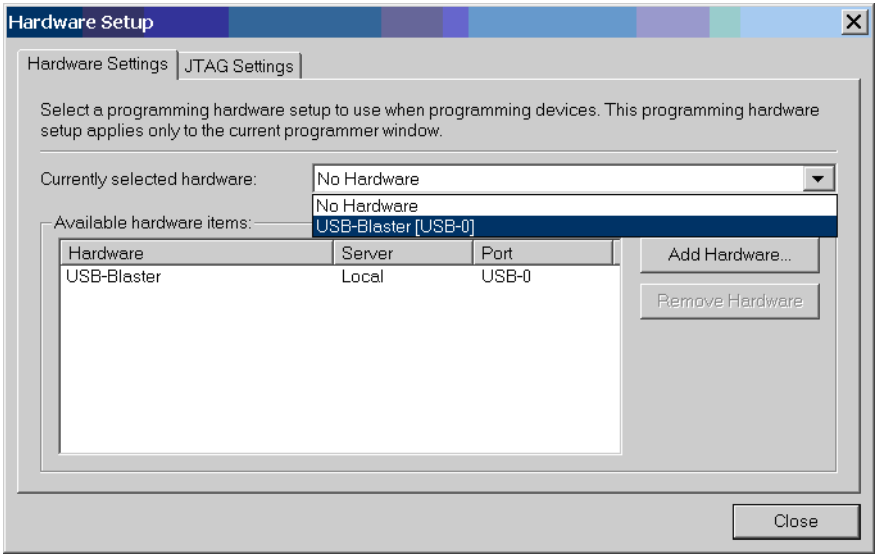
Figure 1–3. Quartus II Programmer



8. Click **Change File**.
9. Browse to the *<installation directory>***altera****72****kits****cycloneIII_3c25_niosII_eval****examples****standard** directory
10. Select the programming file **cycloneIII_embedded_evaluation_kit_standard.sof**.
11. Click **OK**.
12. Click **Hardware Setup** in the top, left corner of the Quartus II Programmer window. The **Hardware Setup** dialog box appears.
13. Select **USB-Blaster** from the **Currently selected hardware** drop-down list box. See Figure 1–4.

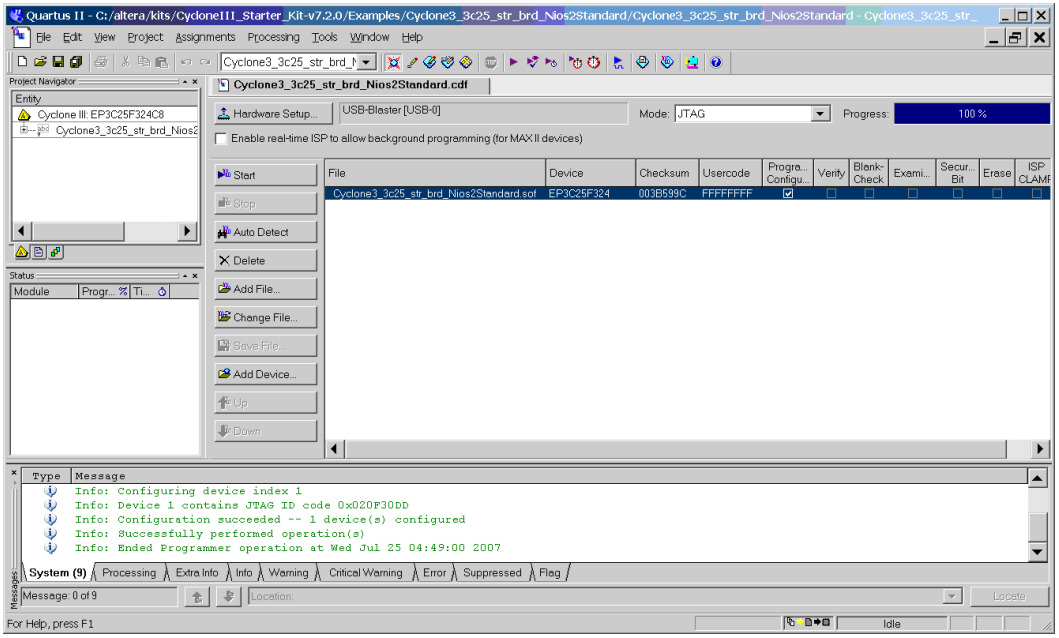
If the appropriate download cable does not appear in the list, you must first install a driver for the cable. Refer to Quartus II Help for information on how to install the driver.

Figure 1–4. Hardware Setup Window



14. Click **Close**.
15. Turn on the **Program/Configure** option for **cycloneIII_embedded_evaluation_kit_standard.sof** (see Figure 1–5).
16. Click **Start**.

Figure 1–5. Quartus II Programmer



The **Progress** meter sweeps to 100% as the Quartus II software configures the FPGA. When configuration is complete, the FPGA is configured with the Nios II system but it does not yet have a C program in memory to execute.

Create the hello_world Example Project

In this section you will create a new Nios II C/C++ application project from an installed example. To begin, perform the following steps in the Nios II IDE:

1. Open a new workspace in the standard hardware project so that the software resides under its own hardware project:

    a. Choose **File > Switch Workspace**.

    b. Click **Browse**.

    c. Navigate to the *<installation directory>***\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard** directory.

d. Click **Make New Folder**.

e. Type `eclipse_workspace`.

f. Click **OK** to exit the workspace launcher.

g. Click **OK**. The Nios II IDE should re-launch in the new workspace.

2. Choose **File > New**.

3. Choose **Nios II C/C++ Application** to open the **New Project** wizard.

4. To associate the software project with the hardware system, perfom the following steps:

Every Nios II software project needs a system description of the corresponding Nios II hardware system. For the Nios II IDE, this system description is contained in a PTF file.

a. Click **Browse** under **Select Target Hardware**. The **Select Target Hardware** dialog box opens.

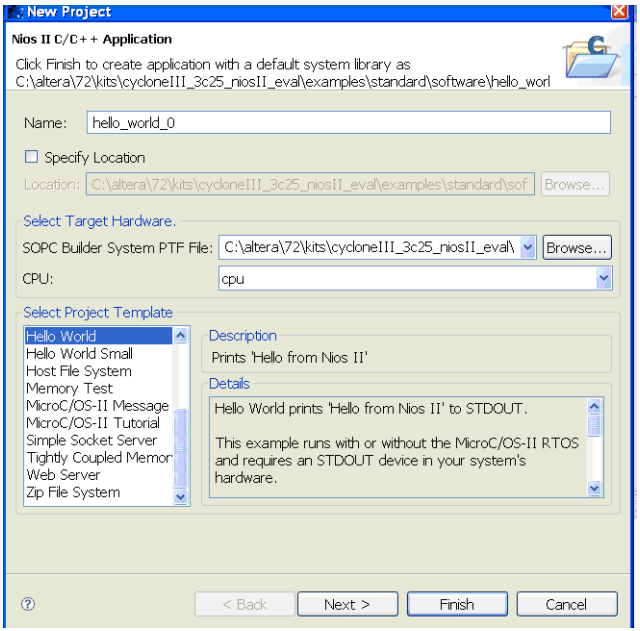
b. Browse to the <*installation directory*>**\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard** directory.

c. Select the file **cycloneIII_embedded_evaluation_kit_standard_sopc.ptf**.

d. Click **Open**. Youare returned to the New Project wizard, which as the **SOPC Builder System** and **CPU** boxes filled in.

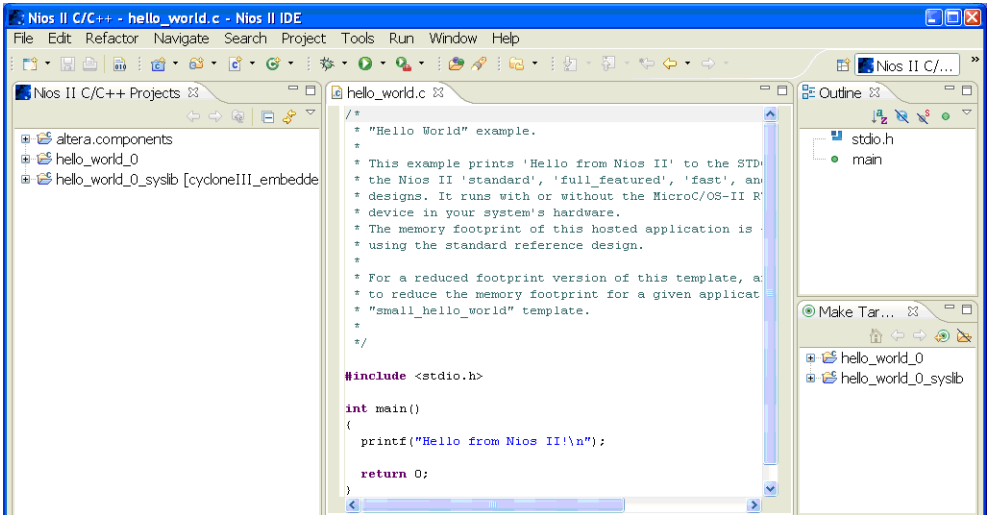
5. Scroll in the **Select Project Template** list to find and select **Hello World**. The **Name** box automatically updates to **hello_world_0**. See Figure 1–6.

Figure 1–6. Nios II IDE New Project Wizard



6. Click **Finish**. The Nios II IDE creates the new project, **hello_world_0**, and returns to the Nios II C/C++ perspective. See Figure 1–7.

Figure 1–7. Nios II IDE C++ Project Perspective for hello_world_0



Whenever you create a new project, the Nios II IDE creates two new projects in the **Nios II C/C++ Projects** tab:

- **hello_world_0** is the C/C++ application project. This project contains all source and header files for your application.
- **hello_world_0_syslib** is a system library that encapsulates the details of the Nios II system hardware.

Build and Run the Program

In this section you will build and run the program to execute the compiled code.

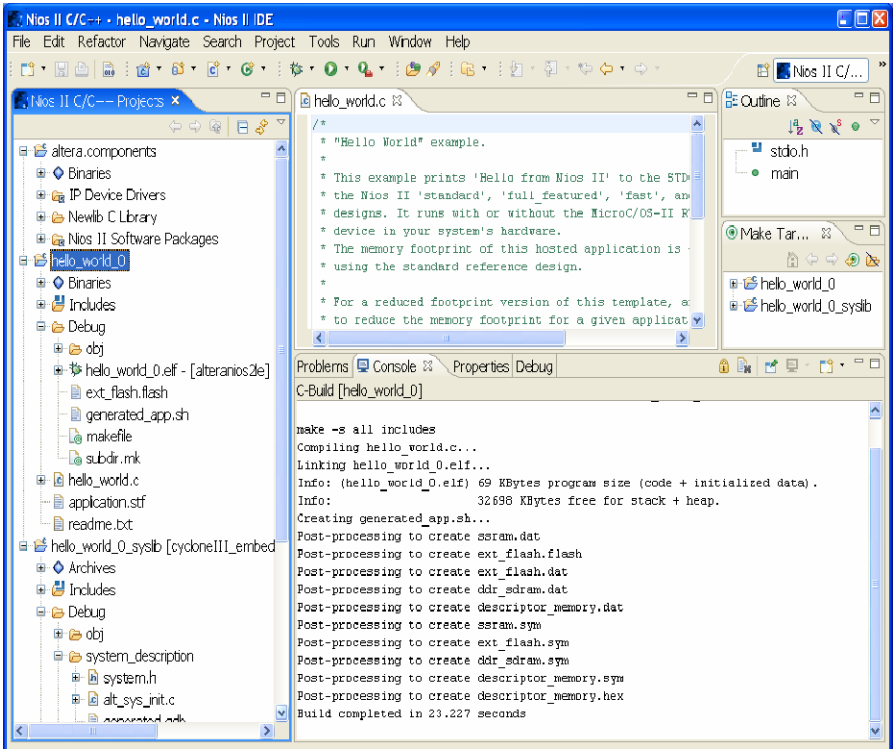
To build the program, right-click the **hello_world_0** project in the **Nios II C/C++ Projects** tab and choose **Build Project**. The **Build Project** dialog box appears, and the IDE begins compiling the project. When compilation completes, the message "Build completed" appears in the **Console** tab. Completion time varies, depending on your system.

When you build the system library for the first time, the Nios II IDE automatically generates files useful for software development, including:

- IP device drivers, including SOPC component device drivers for the Nios II hardware system
- NewLib C library, which is a richly featured C library for the Nios II processor
- Nios II software packages
  - Nios II hardware abstraction layer (HAL)
  - NicheStack TCP/IP network stack, Nios II edition
  - Nios II host file system
  - Nios II read only zip file system
  - Micrium's Micro-C OS-II realtime operating system (RTOS)
- **system.h**, which is a header file that encapsulates your hardware system
- **alt_sys_init.c**, which is an initialization file that initializes the devices in the system
- **Hello_world_0.elf**, which is an executable and linked format file for the application located in the **hello_world_0** folder under **Debug**.

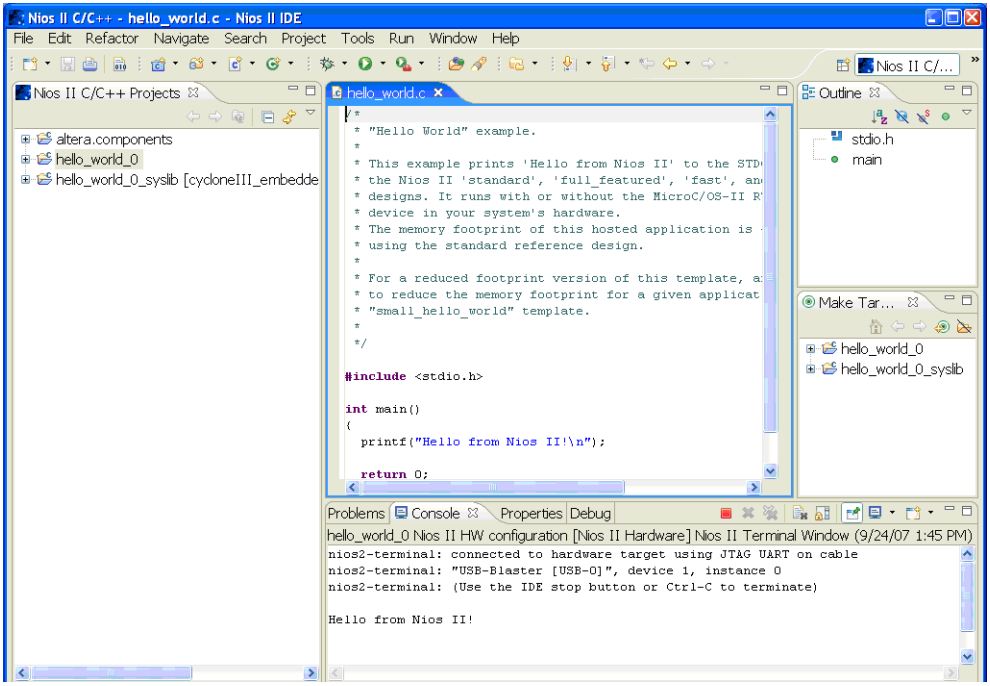
Figure 1–8 shows the Nios II IDE when the build has completed.

Figure 1–8. Nios II IDE hello_world_0 Build Completed



To download the program into the FPGA on the development board, right-click the **hello_world_0** project and choose **Run As > Nios II Hardware**. The IDE downloads the program to the FPGA on the target board and starts execution. When the target hardware begins executing the program, the message "Hello from Nios II!" displays in the Nios II IDE **Console** tab. See Figure 1–9.

Figure 1–9. Hello_World_0 Program Output



Now that you have created, compiled, and run your first software program, you can perform some additional operations, such as configuring the system properties, editing and re-building the application, and debugging the source.

Configure System Properties

By default the project runs program code out of the largest memory device it finds in the target hardware, which, in this case, is the DDR memory. Because this system has SSRAM memory, which has lower latency and faster access compared to DDR SDRAM, you will configure the system properties to execute code from SSRAM memory.

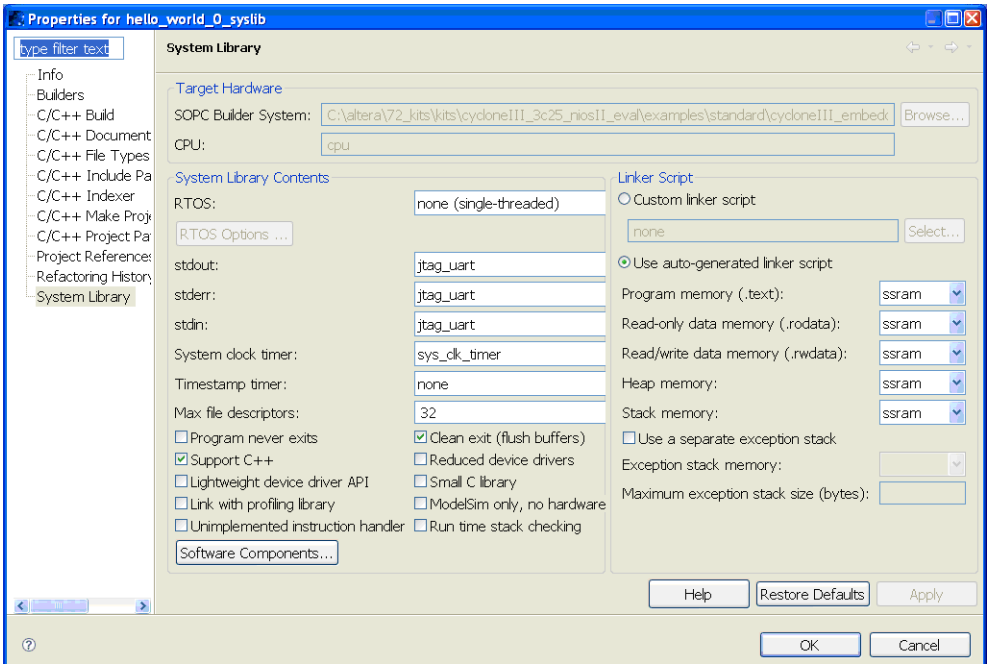
Perform the following steps in the Nios II IDE:

1. In the **Nios II C/C++ Projects** tab, right-click **hello_world_0** and choose **System Library Properties**. The **Properties for hello_world_0_syslib** dialog box opens.

2. Click the **System Library** page. The **System Library** page contains settings for how the program interacts with the underlying hardware. In this example, the settings reflect names that correspond to the targeted Nios II hardware.

3. Under **Linker Script**, change the **Program memory (.text)**, **Read-only data (.rodata)**, **Read/write data (.rwdata)**, **Heap memory**, and **Stack memory** options from **ddr_sdram** (DDR) to **ssram** (SSRAM). See Figure 1–10. This setting determines which memory stores the compiled executable program when the **hello_world_0** program runs.

Figure 1–10. Configuring System Library Properties



4. Click **Apply**.

5. Click **OK** to close the **Properties for hello_world_0_syslib** dialog box and return to the IDE workbench.

You will re-build and download the program to the FPGA in the next section. The system properties will take effect when the program is re-downloaded.

Edit and Re-Run the Program

You can modify the **hello_world_0.c** program file in the IDE, build it , and re-run the program to observe your changes executing on the target board. In this section you will add code that makes LED 1 blink.

For more information how LED1 blinks, refer to "Why the LED Blinks" on page 1–25. Perform the following steps:

1. In the **hello_world_0.c** file, add the text shown in blue in the example below.

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
int main()
{
  printf("Hello from Nios II!\n");
  int count = 0;
  int delay;
  while(1)
  {
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, count & 0x01);
    delay = 0;
    while(delay < 2000000)
    {
        delay++;
    }
    count++;

  }
  return 0;
}
```

2. Save the file.
3. Recompile the file by right-clicking **hello_world_0** in the **Nios II C/C++ Projects** tab and choosing **Run As > Nios II Hardware**.

   You do not need to build the project manually; the Nios II IDE automatically re-builds the program before downloading it to the FPGA.

4. Orient your development board so that you can observe LED 1 blinking.

For more information on running and debugging programs on target hardware, go to the software development tutorial, which is available in the Nios II IDE help.

Debugging the Application

Before you can debug a project in the Nios II IDE, you must create a debug configuration that specifies how to run the software. To set up a debug configuration, perform the following steps:

1. In the **Nios II C/C++ Projects** tab, right-click **hello_world_0** and choose **Debug As > Nios II Hardware**.

   The debug configuration manager shows the message "Specify an SOPC Builder system PTF file."

2. Point the debugger to the system PTF file, which contains information about the target system. Click **Browse**.

3. Go to the *<installation directory>***\altera\72\kits\cycloneIII_3c25_niosII_eval\examples\standard** directory.

4. Choose **cycloneIII_embedded_evaluation_kit_standard_sopc.ptf**.

5. Click **Open**.

6. Click **Apply**.

7. Click **Debug**.

8. If the **Confirm Perspective Switch** dialog box appears, click **Yes**.

   After a moment, the `main()` function appears in the editor. A blue arrow next to the first line of code indicates that execution stopped at that line.

9. Choose **Run > Resume** to resume execution.

When debugging a project in the Nios II IDE, you can pause, stop, or single step the program, set breakpoints, examine variables, and perform many other common debugging tasks.

For more information about debugging software projects in the Nios II IDE, refer to "Nios II Integrated Development Environment" in the *Nios II Software Developer's Handbook* or Nios II IDE help .

Nios II Software Build Flow

The Nios II software build flow is for software programmers who prefer to use the Nios II Command Shell and a few commands. In this section, you use the Nios II software build tools to create an application and a board support package (BSP). A BSP is analogous to a Nios II IDE system library and it contains your hardware system details (in the **system.h** file) as well as device drivers, software packages (if any), operating system (if any), linker files, etc. In this section you will customize the BSP, and build and run the **hello_world** application on the target hardware. You can use your favorite text editor to make changes to the application. To debug the program, you will import the project into the Nios II IDE and use its debugging capabilities.

Create a New Software Project

To create a software project using the Nios II Software Build Flow, you will need to perform the following steps:

1. Open the Nios II command shell by choosing **All Programs > Altera > Nios II EDS** <*version*> **> Nios II Command Shell** in the Windows Start menu.

2. Create a new working directory named **workdir** in a location of your choice and change into it.

3. Type the following commands at the SOPC Builder prompt:

```
mkdir workdir ↵
cd workdir  ↵
```

4. Copy the standard project to the new directory using the following command:

```
cp –R <installation directory>/altera/72/kits/
   cycloneIII_3c25_niosII_eval/examples/standard . ↵
```

5. Ensure the working directory and all subdirectories are writable using the following command:

```
chmod –R +w . ↵
```

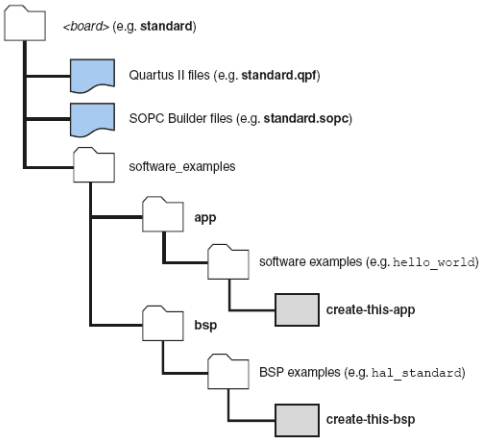
6. Review the directory structure of the standard project that you copied (see Figure 1–11).

   The standard project contains all the Quartus II project files and the SOPC files for the Nios II standard hardware system you will use in this tutorial. Pre-packaged example applications for the standard

project reside in the **standard/software_examples/app** directory. All pre-packaged board support packages reside in the **standard/software_examples/bsp** directory.

You will use the **hello_world** example application for this tutorial. The **hello_world** folder contains a script named **create-this-app**. By default the **create-this-app** script associates the application with a default BSP, which in this case is **hal_default**.

Figure 1–11. Nios II Standard Project Directory Structure



7. Type the following commands at the SOPC Builder prompt to create and build the **hello_world** application with the **create-this-app** script:

```
cd standard/software_examples/app/hello_world ↵
./create-this-app ↵
```

A good way to familiarize yourself with using the Nios II software build flow command set is to open and study the provided **create-this-app** script. You will observe that the script:

- Copies the application source code into the **app** directory.
- Runs a command (**nios2-app-generate-makefile**) to create the application makefile (named **Makefile**).
- Runs **make** to create the executable file (**hello_world.elf**).
- Finds a compatible BSP by looking in **workdir/standard/software_example/bsp**. In this case it selects the **hal_default** BSP.

- Invokes the **create-this-bsp** script, which generates the BSP in the **workdir/standard/software_example/bsp/hal_default** directory.

Run the hello_world Application

To run the **hello_world** application, perform the following steps:

1. Configure the FPGA with the Nios II standard system using the SOF.

   a. Navigate to **workdir/standard** where the SOF for the Nios II standard Quartus II project is located.

   b. Type the following command at the SOPC Builder prompt:

   ```
   nios2-configure-sof ↵
   ```

   The **nios2-configure-sof** script runs the Quartus II Programmer to download the SOF. The board is configured and ready to run the executable.

2. Launch another Nios II command shell. If possible, orient both shell windows so that the are visible simultaneously on your desktop.

3. When the executable is downloaded, the Nios II processor communicates via the Joint Test Action Group (JTAG) UART. In the second command shell, start the Nios II terminal application to connect to the Nios II processor on the board via the JTAG UART port by typing the following command at the SOPC Builder prompt:

   ```
   nios2-terminal ↵
   ```

   The Nios II terminal will capture and display all communication from the Nios II processor.

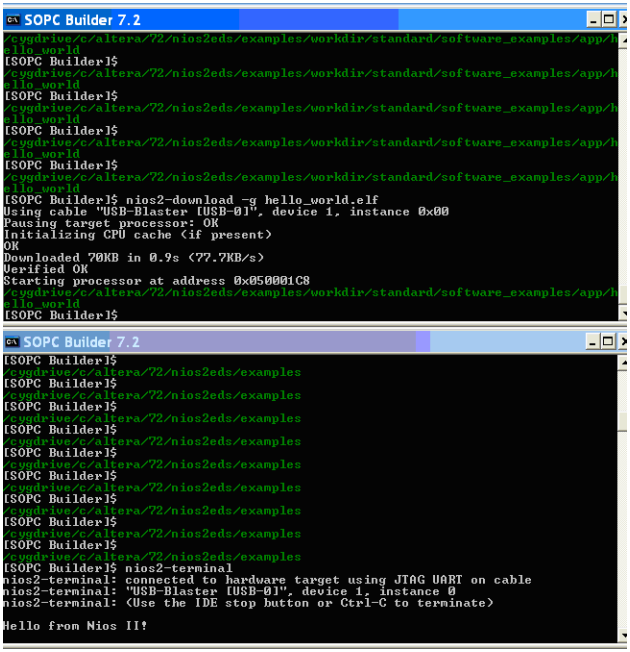
4. Return to the first command shell and navigate to the **workdir/standard/software_examples/app/hello_world** directory.

5. Download and run the **hello_world** application by typing the following command at the SOPC Builder prompt:

   ```
   nios2-download -g hello_world.elf ↵
   ```

   As in the Nios II IDE build example, the message “Hello from Nios II!” appears in the Nios II terminal.

Figure 1–12. Hello World Program Output in the Nios II Terminal



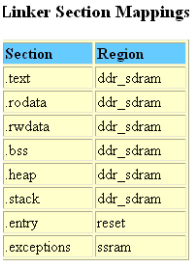
Now that you have successfully run your first software program, you can perform some additional operations, such as configuring your application, editing it, and debugging it.

Configure the Application

As described in the Nios II IDE flow, the project defaults to running program code out of the largest memory device it finds in the target hardware, which, in this case, is the DDR memory. Because this system has SSRAM memory, which has lower latency and faster access compared to DDR SDRAM, you will configure the system properties to execute code from SSRAM memory.

Using Windows Explorer, navigate to the **workdir\standard\software_examples\bsp\hal_default** directory. The **summary.html** file contains all default settings for the BSP. In the Linker Section Mappings table, observe that the program executes using the DDR SDRAM (see Figure 1–13). You will configure the Linker Section Mappings to execute program code using the SSRAM instead.

Figure 1–13. Linker Section Mappings for DDR SRAM



Linker Section Mappings

Section	Region
.text	ddr_sdram
.rodata	ddr_sdram
.rwdata	ddr_sdram
.bss	ddr_sdram
.heap	ddr_sdram
.stack	ddr_sdram
.entry	reset
.exceptions	ssram

You use the **nios2-bsp** command to change the system so that it executes using the SSRAM. Before you type this command, type `nios2-bsp --help` ↵ to learn the arguments that accompany it.

When using the **nios2-bsp** command, you specify the:

- Type of BSP, HAL or µC-OSII (In this case, you use HAL)
- Location of the BSP (in this case it is **hal_default**)
- Location of the SOPC file (in this case it is **cycloneIII_embedded_evaluation_kit_standard_sopc.sopc** located in **workdir\standard**)
- Custom Tcl options (in this case you use **default-sections-mapping** <*region*> to specify SSRAM as the location for the linker regions)

If you type `nios2-bsp` ↵ in the Nios II command shell, the command usage displays.

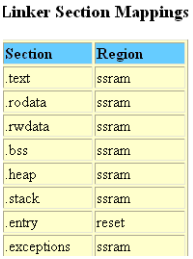
1. In the Nios II command shell, navigate to the **workdir\standard\software_examples\bsp\hal_default** directory

2. Type the following command on a single line:

```
nios2-bsp hal . ../../../cycloneIII_embedded_evaluation_kit_standard_sopc.sopc
   --default_sections_mapping ssram ↵
```

3. To verify the change in the system properties, refresh or re-open **summary.html** in the **hal_default** directory. You should see that the Linker Section Mappings table is now mapped to SSRAM instead of DDR SDRAM. See Figure 1–14.

Figure 1–14. Linker Section Mappings for SSRAM



Linker Section Mappings

Section	Region
.text	ssram
.rodata	ssram
.rwdata	ssram
.bss	ssram
.heap	ssram
.stack	ssram
.entry	reset
.exceptions	ssram

4. Verify that your application is associated with the new BSP by navigating to the **workdir\standard\software_examples\app\hello_world** directory and typing the commands:

```
make clean_all ↵
make ↵
```

The resulting **hello_world.elf** should reflect the changes that you made to the BSP. You will rebuild and download the program in the next section. The modifications the system properties take effect when you re-download the program.

Edit and Re-Run the Program

In this section you will add code that causes LED 1 on the development board to blink. For information on how LED 1 blinks, go to "Why the LED Blinks" on page 1–25. Perform the following steps to add the code:

1. Using a text editor, add the code shown in blue in the following example to **hello_world.c**:

```
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
int main()
{
  printf("Hello from Nios II!\n");
  int count = 0;
  int delay;
  while(1)
  {
    IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, count & 0x01);
    delay = 0;
    while(delay < 2000000)
    {
        delay++;
    }
    count++;

  }
  return 0;
}
```

2. In one command shell, save and rebuild the edited **hello_world_0** project using the command `make` ↵.

3. Download the program to the FPGA on the development board by typing the following command at the SOPC Builder prompt:

   `nios2-download -g hello_world_0.elf` ↵

4. In the second command shell, run the Nios II terminal by typing `nios2-terminal` ↵.

5. When the target hardware executes the program, the message "Hello from Nios II!" displays in the Nios II terminal.

6. Orient the development board so that you can observe the LED 1 blinking.

Debugging the Application

You use the Nios II IDE to debug a Nios II software build flow program. First, you import the program into the Nios II IDE. Next, the IDE uses the project's makefiles to build the project. To import the **hello_world** application, perform the following steps:

1. Launch the Nios II IDE by choosing **All Programs > Altera > Nios II EDS** <*version*> **> Nios II IDE** <*version*> in the Windows Start menu.
2. Choose **File > Import.**
3. In the **Import** dialog box, expand the **Altera Nios II** folder, and select **Existing Nios II Software build tools or folder into workspace**.
4. Click **Next**.
5. Click **Browse**.
6. Navigate to the **workdir/standard/software_examples/app/hello_world** directory. Click **Open**.
7. Click **OK**. The wizard fills the project name and path. The project name is the directory name.
8. Click **Finish**. The wizard imports the project, creating a new C/C++ application project in the workspace.
9. Repeat steps 2 through 6, but instead import the **hal_default** folder from **workdir/standard/software_examples/bsp/hal_default** into the Nios II IDE.

Before you can debug a project in the Nios II IDE, you must set up a debug session. To set up a debug session, perform the steps given in "Debugging the Application" on page 1–16.

When debugging a project in the Nios II IDE, you can pause, stop, or single step the program, set breakpoints, examine variables, and perform many other common debugging tasks.

For more information on creating and configuring projects using the Nios II software build flow, refer to the "Nios II Software Build Tools" chapter in the *Nios II Software Developer's Handbook*.

Why the LED Blinks

The Nios II system description header file, **system.h**, contains the software definitions, names, locations, base addresses, and settings for all of the components in the Nios II hardware system. The **system.h** file is located in the **hello_world_0_syslib\Debug\system_description** directory.

If you look at the **system.h** file for the Nios II project example used in this tutorial, you will notice the **led_pio** function. This function controls the LED. The Nios II microprocessor controls the PIO ports (and thereby the LED) by reading and writing to the register map. For the PIO, there are four registers: `data`, `direction`, `interruptmask`, and `edgecapture`. To turn the LED on and off, the application needs to writes to the PIO data register.

The PIO core has an associated software file, **altera_avalon_pio_regs.h**. This file defines the core's register map, providing symbolic constants to access the low-level hardware. The **altera_avalon_pio_regs.h** file is located in *<installation directory>***\altera\72\ip\sopc_builder_ip\altera_avalon_pio**.

When you include the **altera_avalon_pio_regs.h** file, several useful functions that manipulate the PIO core registers are available to your program. In particular, the `IOWR_ALTERA_AVALON_PIO_DATA(base, data)` function can be used to write to the PIO data register and thereby turn the LED on and off.

The PIO is just one of many SOPC peripherals that you can use in a system. To learn about the PIO core and other embedded peripheral cores, refer to *Quartus II Version <version> Handbook Volume 5: Embedded Peripherals*.

When developing your own designs, you can use the software functions and resources that are provided with the Nios II HAL. Refer to the *Nios II Software Developer's Handbook* for extensive documentation on developing your own Nios II processor-based software applications.

Next Steps

The following documents provide next steps to further your understanding of the Nios II processor:

- *Developing Software for Nios II*—These short, online software tutorials walk you through the basics of developing software for the Nios II processor. You can access these tutorials from the Training link on the Embedded Processing web page at www.altera.com/embedded.
- *Nios II Software Developer's Handbook*—This handbook provides a complete reference on developing software for the Nios II processor.
- *Software Development Tutorial*—This tutorial teaches how to use the Nios II IDE to develop, run, and debug new Nios II C/C++ application projects. This tutorial is available in the Nios II IDE help.
- *Nios II IDE Help*—The Nios II IDE help provides complete reference on features of the IDE. To open the help, click **Help > Help Contents** and click the **Nios II IDE Help** book in the **Contents** pane.
- *Nios II Processor Reference Handbook*—This handbook provides a complete reference for the Nios II processor hardware.
- *Quartus II <version> Handbook Volume 5: Embedded Peripherals*—This olume contains details on the peripherals provided with the Nios II Embedded Design Suite.
- *Quartus II <version> Handbook Volume 4: SOPC Builder*—This volume provides a complete reference on using SOPC Builder, including building memory subsystems and creating custom components.

For a complete list of all documents available for the Nios II processor, visit the Nios II literature page at www.altera.com/nios2.