

1 Tutorial 1: Hello World on the Nios II platform, Cyclone III FPGA



Overview

This tutorial will demonstrate how to generate hardware and related software interfaces in the form of RTL (Register Transfer Logic) HDL descriptions and software system libraries appropriate for use with the **Altera Quartus II**, **SOPC Builder**, and **Nios II IDE** software tools.

The sample project will implement a trivial **Hello World** application. This sample project includes a software process (running on the Nios II processor) and a hardware process (running in the FPGA) that communicate via a single stream over the **Avalon** bus. Onchip memory inside the FPGA is sufficient for accommodating the **Hello World** executable file.

The purpose of this tutorial is to take you through the entire process of generating hardware and software interfaces and importing the relevant files to the Altera environment. The tutorial will also describe how to create the platform and downloadable FPGA bitmap using the **Altera** tools. Finally, you will learn how to build and run an Impulse C software application on the **Nios II** processor using the **Nios II IDE**.

The hardware platform used in this tutorial is **Altera Cyclone III Evaluation Kit**, featuring **Altera Cyclone III EP3C25 FPGA**, and a touch-screen LCD display.

This tutorial will require approximately 90 minutes to complete, including software run times.

Steps

- [Loading the Hello World Application](#)
- [Compiling the Application for Simulation](#)
- [Building the Application for the Target Platform](#)
- [Exporting Files from CoDeveloper](#)
- [Creating a Quartus Project](#)
- [Creating the New Platform](#)
- [Configuring the New Platform](#)
- [Generating the System](#)
- [Generating the FPGA Bitmap](#)
- [Running the Application on the Platform](#)

*Note: This tutorial assumes you have purchased or are evaluating the **CoDeveloper Platform Support Package** for **Altera Nios II**, and that you have installed and have valid licenses for the **Altera Quartus II**, **SOPC Builder**, and **Nios II IDE** products.*

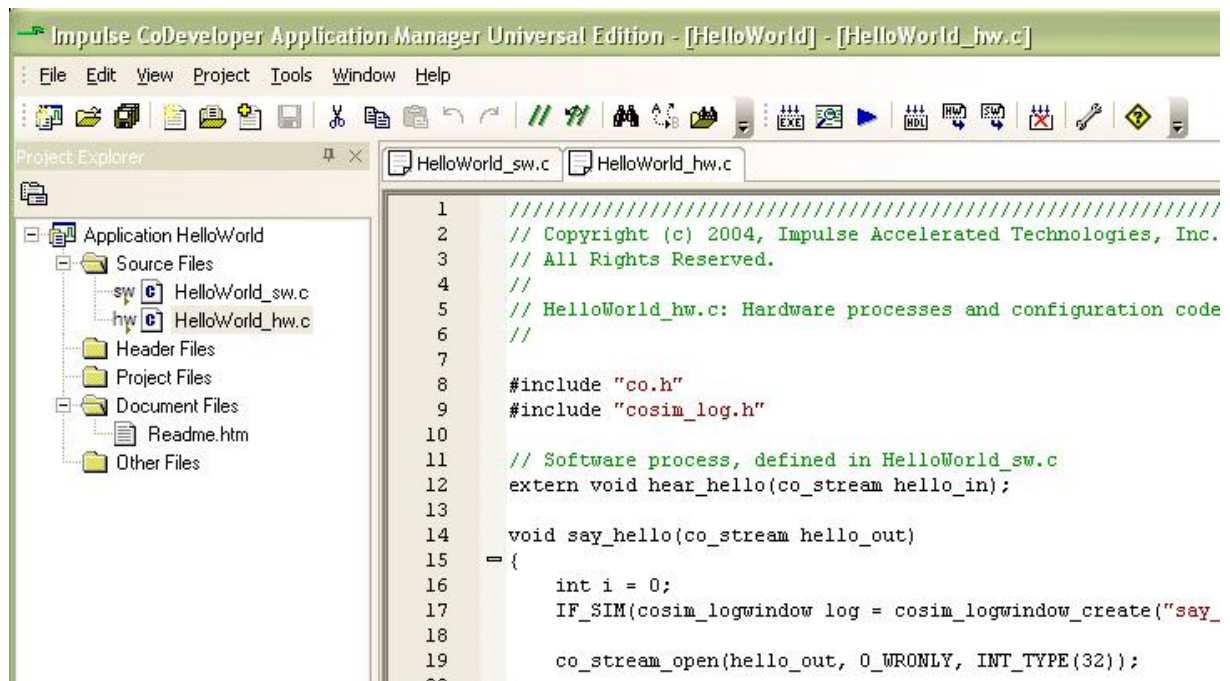
1.1 Loading the Hello World Application

Hello World Tutorial for Nios II, Step 1

To begin, start the CoDeveloper Application Manager by selecting Application Manager from the Start -> Programs -> Impulse Accelerated Technologies -> CoDeveloper program group.

Note: this tutorial assumes that you have already read and understand the basic "Hello World" tutorial presented in the CoDeveloper User's Guide.

Open the Altera Nios II Hello World sample project by selecting Open Project from the File menu, or by clicking the Open Project toolbar button. Navigate to the .\Examples\Embedded\HelloWorld_NIOS directory within your CoDeveloper installation. (You may wish to copy this example to an alternate directory before beginning.) Opening the project will display a window similar to the following:



Files included in the HelloWorld project include:

Source files HelloWorld_sw.c and HelloWorld_hw.c - These source files represent the complete application, including the **main()** function, a software process, and a single hardware process.

Quartus subdirectory - Files in the Quartus subdirectory are used later in this tutorial to simplify the creation of the hardware platform.

Note

Due to limitations in the Altera SOPC Builder software, CoDeveloper projects must not be located in a directory that includes spaces in its name (such as "My Documents").

See Also

Step 2: [Compiling the Application for Simulation](#)

[Tutorial 1: Hello World on the Nios II platform](#)

1.2 Compiling the Application for Simulation

Hello World Tutorial for Nios II, Step 2

Before compiling the HelloWorld application to the target Nios II platform, let's first take a moment to understand its basic operation and the contents of its primary source files.

The specific process that we will be compiling to hardware is represented in HelloWorld_hw.c by the following function:

```
void say_hello(co_stream hello_out)
```

This hardware process simply sends data to an output stream (**hello_out**).

This hardware process is accompanied in the HelloWorld_hw.c file by a configuration function (**config_hello()**) that specifies the connection between the **hello_out** output stream and the corresponding input stream for the software process **hear_hello**.

The process **hear_hello** (which is declared as an **extern** function at the start of HelloWorld_hw.c) is defined in the source file HelloWorld_sw.c. This process will, when compiled, be loaded onto the Nios II processor and will communicate with the FPGA hardware via an automatically-generated hardware/software interface. In this case the software process simply reads data from the stream associated with the hardware process. (In a more typical Impulse C application there are multiple input/output streams associated with a given hardware process.)

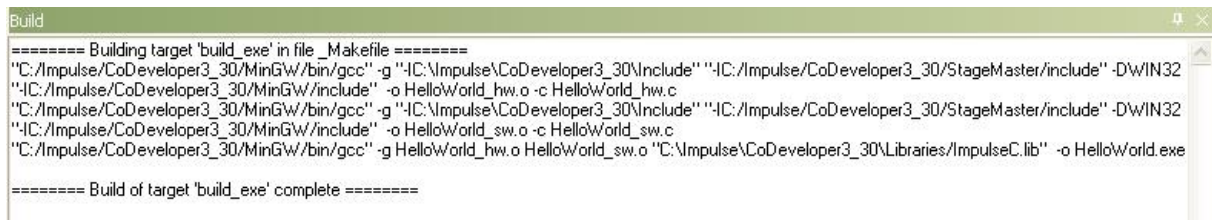
HelloWorld_sw.c also includes a **main()** function for the application. This **main()** function makes reference to the lower-level configuration function **config_hello()** through the Impulse C standard function **co_initialize()**, which has been declared as an **extern** function and may be found in HelloWorld_hw.c.

*Note: the organization of source files shown in this example is not required, but is recommended in order to simplify the compilation process. It is a good idea to keep the **main()** function and software processes of your application in one file or set of files (as shown), and to place hardware processes, configuration function, and **co_initialize()** in one or more different files. Doing so will avoid potential problems with cross-compiler incompatibilities between hardware and software processes as well as simplify the compilation and linking of complete applications for desktop simulation.*

Simulating the Hello World Application

To compile and simulate the application for the purpose of functional verification:

1. Select Project -> Build Simulation Executable (or click the Build Simulation Executable button) to build the HelloWorld.exe executable. A command window will open, displaying the compile and link messages as shown below:

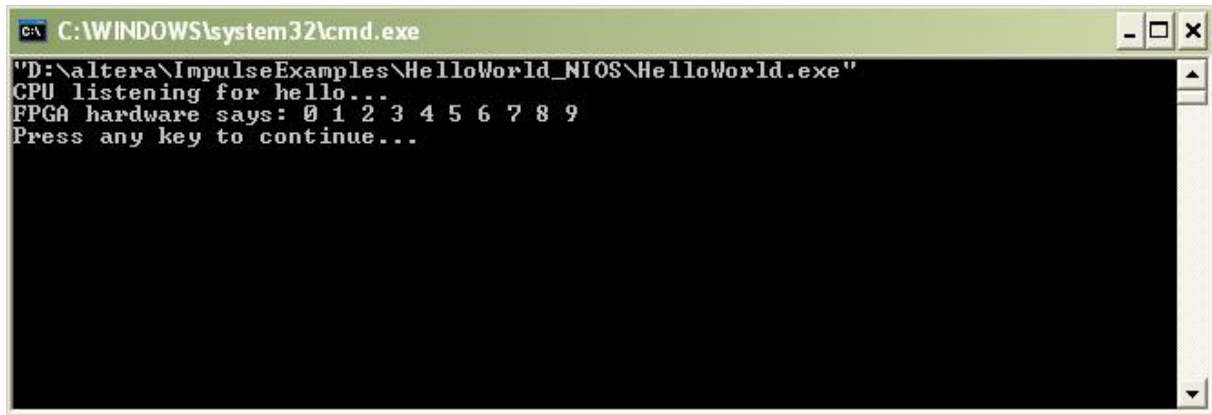


```

Build
===== Building target 'build_exe' in file _Makefile =====
"C:\Impulse\CoDeveloper3_30\MinGW\bin/gcc" -g "C:\Impulse\CoDeveloper3_30\Include" "C:\Impulse\CoDeveloper3_30\StageMaster\include" -DWIN32
"C:\Impulse\CoDeveloper3_30\MinGW\bin/gcc" -o HelloWorld_hw.o -c HelloWorld_hw.c
"C:\Impulse\CoDeveloper3_30\MinGW\bin/gcc" -g "C:\Impulse\CoDeveloper3_30\Include" "C:\Impulse\CoDeveloper3_30\StageMaster\include" -DWIN32
"C:\Impulse\CoDeveloper3_30\MinGW\bin/gcc" -o HelloWorld_sw.o -c HelloWorld_sw.c
"C:\Impulse\CoDeveloper3_30\MinGW\bin/gcc" -g HelloWorld_hw.o HelloWorld_sw.o "C:\Impulse\CoDeveloper3_30\Libraries\ImpulseC.lib" -o HelloWorld.exe
===== Build of target 'build_exe' complete =====

```

2. You now have a Windows executable representing the HelloWorld application implemented as a desktop (console) software application. Run this executable by selecting Project -> Launch Simulation Executable. A command window will open and the simulation executable will run as shown below:



```

C:\WINDOWS\system32\cmd.exe
"D:\altera\ImpulseExamples\HelloWorld_NIOS\HelloWorld.exe"
CPU listening for hello...
FPGA hardware says: 0 1 2 3 4 5 6 7 8 9
Press any key to continue...

```

Verify that the simulation produces the following output:

```

CPU listening for hello...
FPGA hardware says: 0 1 2 3 4 5 6 7 8 9

```

Note that, although the displayed messages indicate that the FPGA hardware has generated the indicated values, in fact the values are being generated by a hardware process which, for simulation purposes, has been compiled as a software process on your host development system (Windows). The next steps will show how to take this same Impulse C application and compile to actual hardware.

See Also

Step 3: [Building the Application for the Target Platform](#)

[Tutorial 1: Hello World on the Nios II platform](#)

1.3 Building the Application for the Target Platform

Hello World Tutorial for Nios II, Step 3

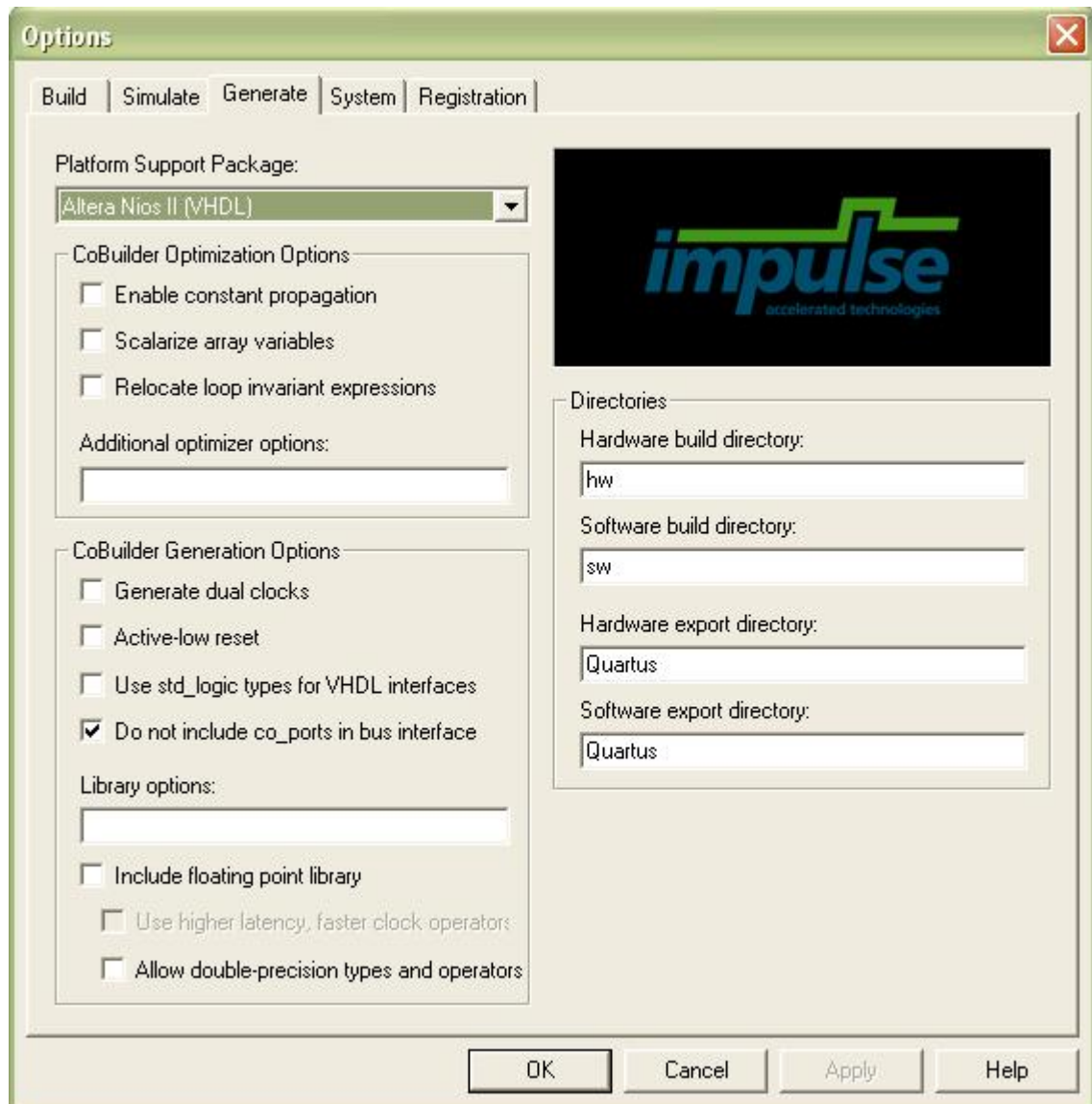
Specifying the Platform Support Package

The next step, prior to compiling and generating the HDL and related output files, is to select a platform

target. Which target you select has a number of implications, including:

- The output file format (e.g., VHDL, Verilog, or other intermediate language)
- The primitive components that will be referenced in the output (e.g., memory cores or buffer types)
- The types of optimizations performed during compilation
- The software library components that will be generated

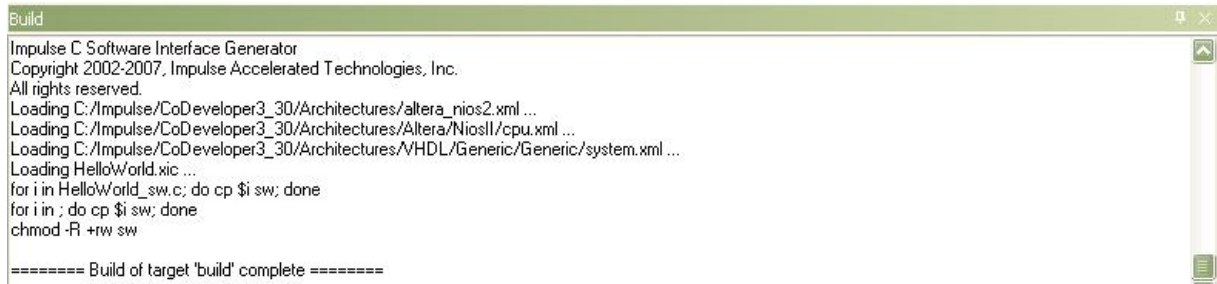
To specify a platform target, open the Generate Options dialog as shown below (Project -> Options, Generate tab):



Specify **Altera Nios II (VHDL)** as shown. Also specify "hw" and "sw" for the hardware and software directories as shown, and specify "Quartus" for the hardware and software export directories. Click OK to save the options and exit the dialog.

Generate HDL for the Hardware Process

To generate hardware in the form of HDL files, and to generate the associated software interfaces and library files, select Generate HDL from the Project menu, or click on the Generate HDL button. A series of processing steps will run (in a command window) as shown below:



```
Build
Impulse C Software Interface Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/Altera/NiosII/cpu.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Generic/Generic/system.xml ...
Loading HelloWorld.xic ...
for i in HelloWorld_sw.c; do cp $i sw; done
for i in ; do cp $i sw; done
chmod -R +rw sw
===== Build of target 'build' complete =====
```

When processing is complete, several files will have been created in the "hw" and "sw" subdirectories of your project directory. Take a moment to review these generated files. They include:

Hardware directory ("hw")

- Generated VHDL source files (HelloWorld_comp.vhd, HelloWorld_top.vhd and subsystem.vhd) representing the hardware process and the generated hardware stream interface.
- A **lib** subdirectory containing required VHDL library elements.
- A **class** subdirectory containing generated files required by the Altera SOPC Builder tools.

Software directory ("sw")

- C source files extracted from the project that are required for compilation to the embedded processor (in this case HelloWorld_sw.c).
- A generated C file (co_init.c) containing the hardware configuration function. This file will also be compiled to the embedded processor.
- A **class** subdirectory containing additional software libraries to be compiled as part of the embedded software application. These libraries implement the software side of the hardware/software interface.

If you are an experienced Altera tools user, you may copy these files manually to your Altera project area and, if needed, modify them to suit your needs. In the next step, however, we will show how to use the hardware and software export features of CoDeveloper to move these files into your Altera project automatically.

See Also

Step 4: [Exporting Files from CoDeveloper](#)

[Tutorial 1: Hello World on the Nios II platform](#)

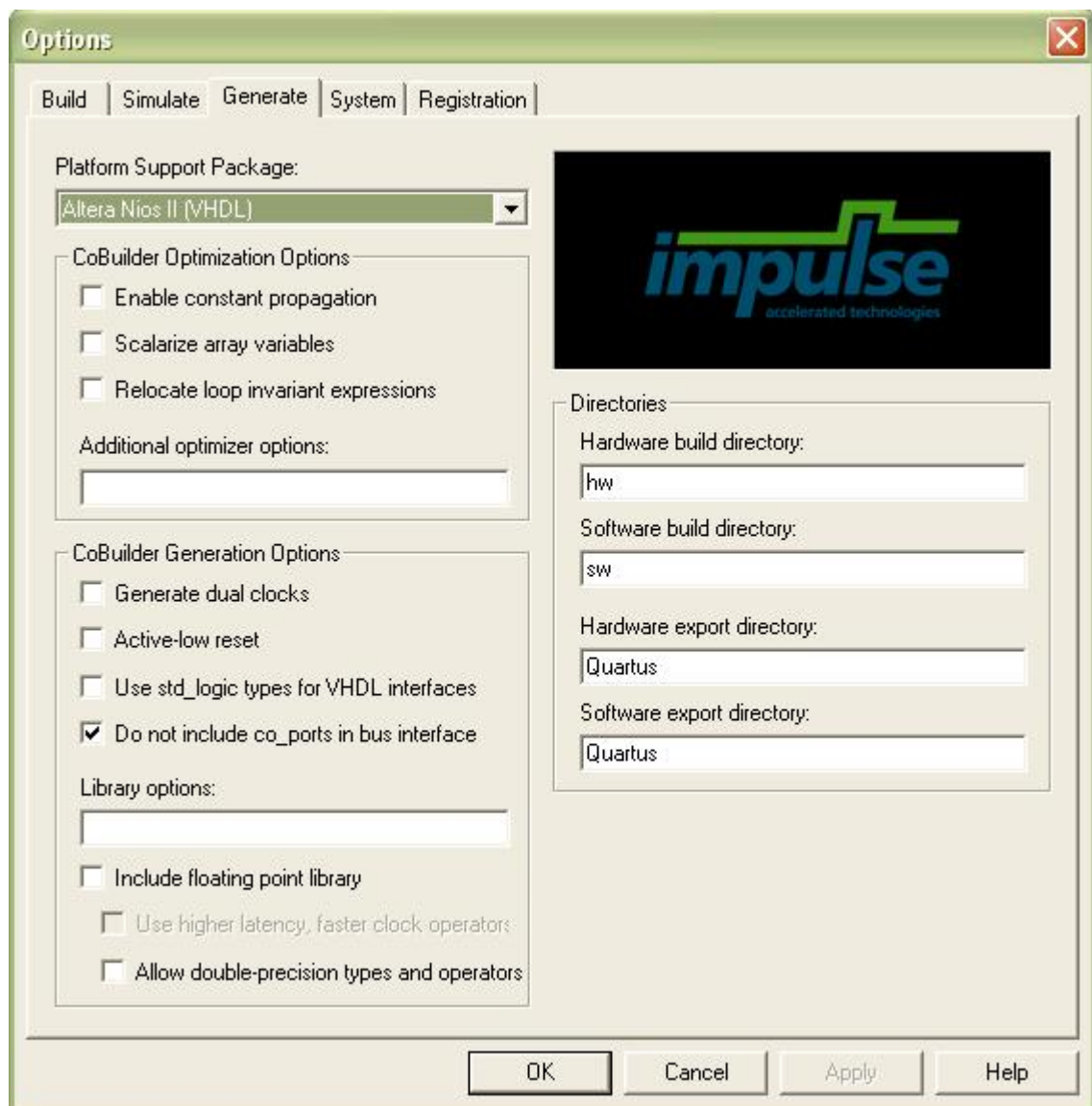
1.4 Exporting Files from CoDeveloper

Hello World Tutorial for Nios II, Step 4

As you saw in the previous step, CoDeveloper creates a number of hardware and software-related output files that must all be used to create a complete hardware/software application on the target platform (in this case an Altera FPGA with an embedded Nios II processor). You can, if you wish, copy these files manually and integrate them into your existing Altera projects. Alternatively, you can use the export features of CoDeveloper to integrate the files into the Altera tools semi-automatically. This section will walk you through the process, using a new Quartus project as an example.

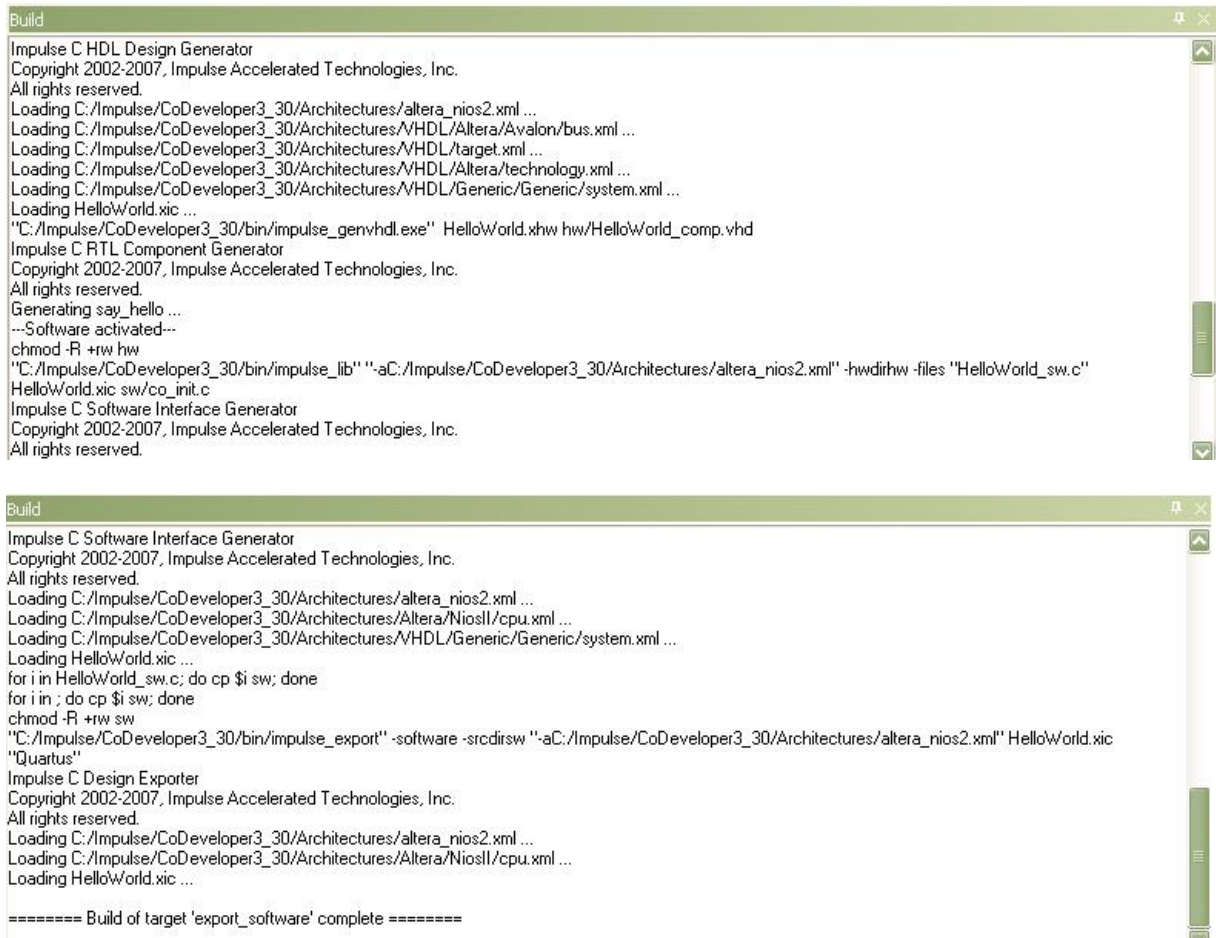
Note: you must have the Altera Quartus II (version 4.0 or later) and SOPC Builder software installed in order to proceed with this and subsequent steps.

Recall that in [Step 3](#) you specified the directory "Quartus" as the export target for hardware and software:



These export directories specify where the generated hardware and software files are to be copied when the Export Software and Export Hardware features of CoDeveloper are invoked. Within these target directories (in this case we have specified both directories as "Quartus"), the specific destination (which may be a subdirectory under "Quartus") for each file is determined from the Platform Support Package architecture library files. It is therefore important that the correct Platform Support Package (in this case, Altera Nios II) is selected prior to starting the export process.

To export the files from the build directories (in this case "hw" and "sw") to the export directories (in this case the "Quartus" directory), select Project -> Export Generated Hardware (HDL) and Project -> Export Generated Software, or select the Export Generated Hardware and Export Generated Software buttons from the toolbar.



```

Build
Impulse C HDL Design Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Altera/Avalon/bus.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Altera/target.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Altera/technology.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Generic/Generic/system.xml ...
Loading HelloWorld.xic ...
"C:/Impulse/CoDeveloper3_30/bin/impulse_genvhdl.exe" HelloWorld.xhw hw/HelloWorld_comp.vhd
Impulse C RTL Component Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Generating say_hello ...
---Software activated---
chmod -R +rw hw
"C:/Impulse/CoDeveloper3_30/bin/impulse_lib" "-aC:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml" -hwdirhw -files "HelloWorld_sw.c"
HelloWorld.xic sw/co_init.c
Impulse C Software Interface Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.

Build
Impulse C Software Interface Generator
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/Altera/NiosII/cpu.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/VHDL/Generic/Generic/system.xml ...
Loading HelloWorld.xic ...
for i in HelloWorld_sw.c; do cp $i sw; done
for i in ; do cp $i sw; done
chmod -R +rw sw
"C:/Impulse/CoDeveloper3_30/bin/impulse_export" -software -srcdirsw "-aC:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml" HelloWorld.xic
"Quartus"
Impulse C Design Exporter
Copyright 2002-2007, Impulse Accelerated Technologies, Inc.
All rights reserved.
Loading C:/Impulse/CoDeveloper3_30/Architectures/altera_nios2.xml ...
Loading C:/Impulse/CoDeveloper3_30/Architectures/Altera/NiosII/cpu.xml ...
Loading HelloWorld.xic ...

===== Build of target 'export_software' complete =====
  
```

Note: you must select BOTH Export Software and Export Hardware before going onto the next step.

You have now exported all necessary files from CoDeveloper to the Quartus project directory.

See Also

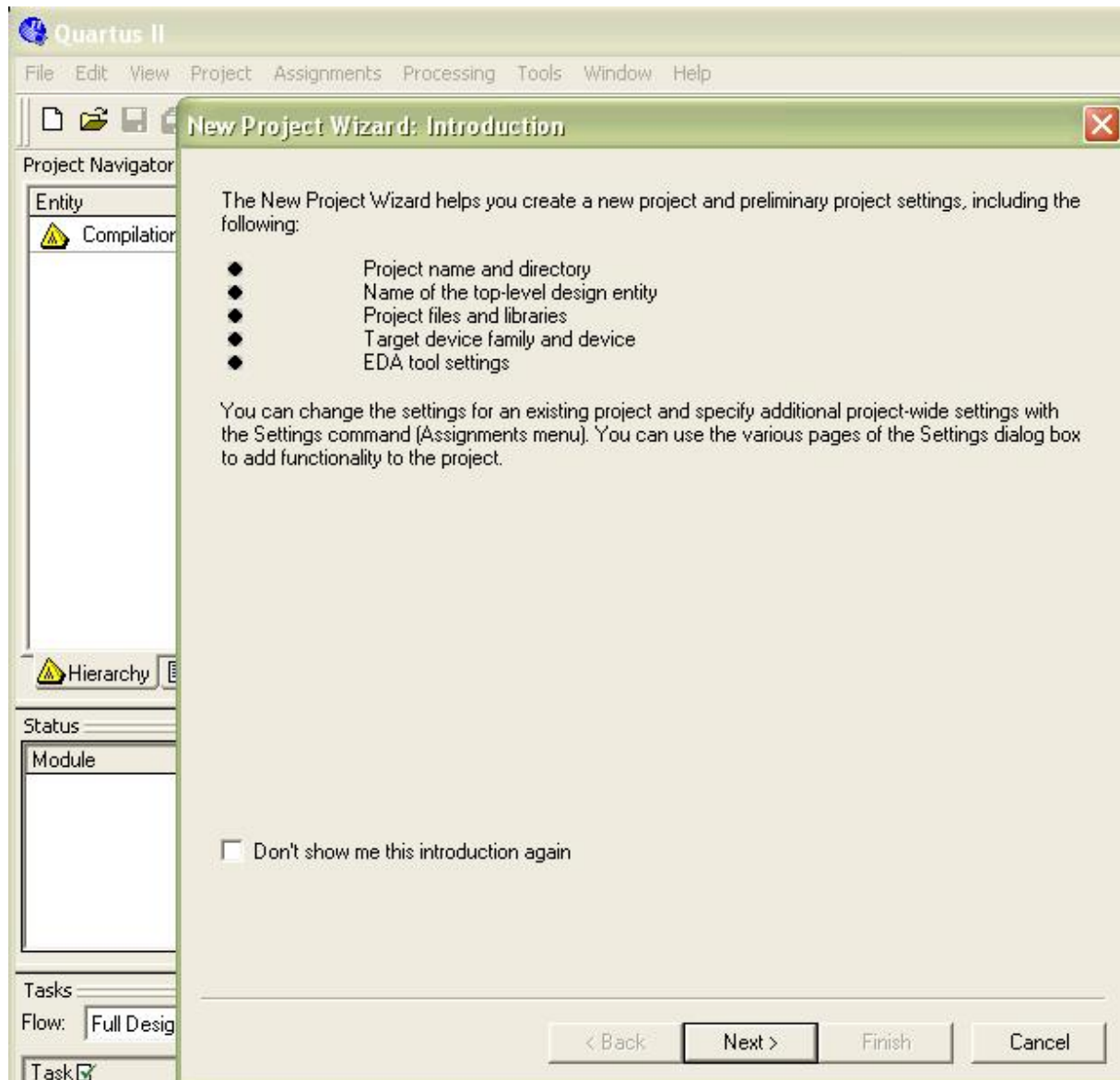
Step 5: [Creating a Quartus Project](#)

[Tutorial 1: Hello World on the Nios II platform](#)

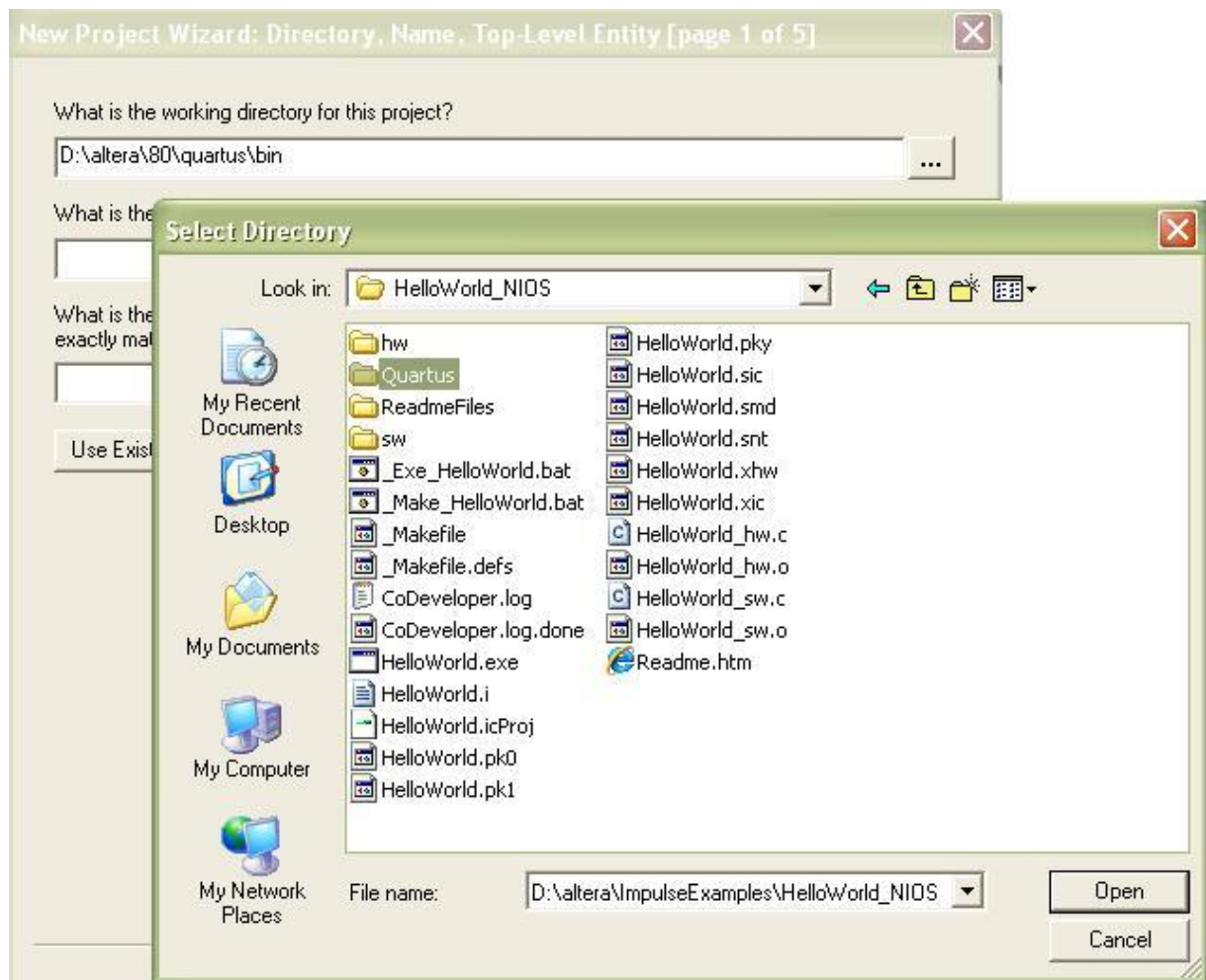
1.5 Creating a Quartus Project

Hello World Tutorial for Nios II, Step 5

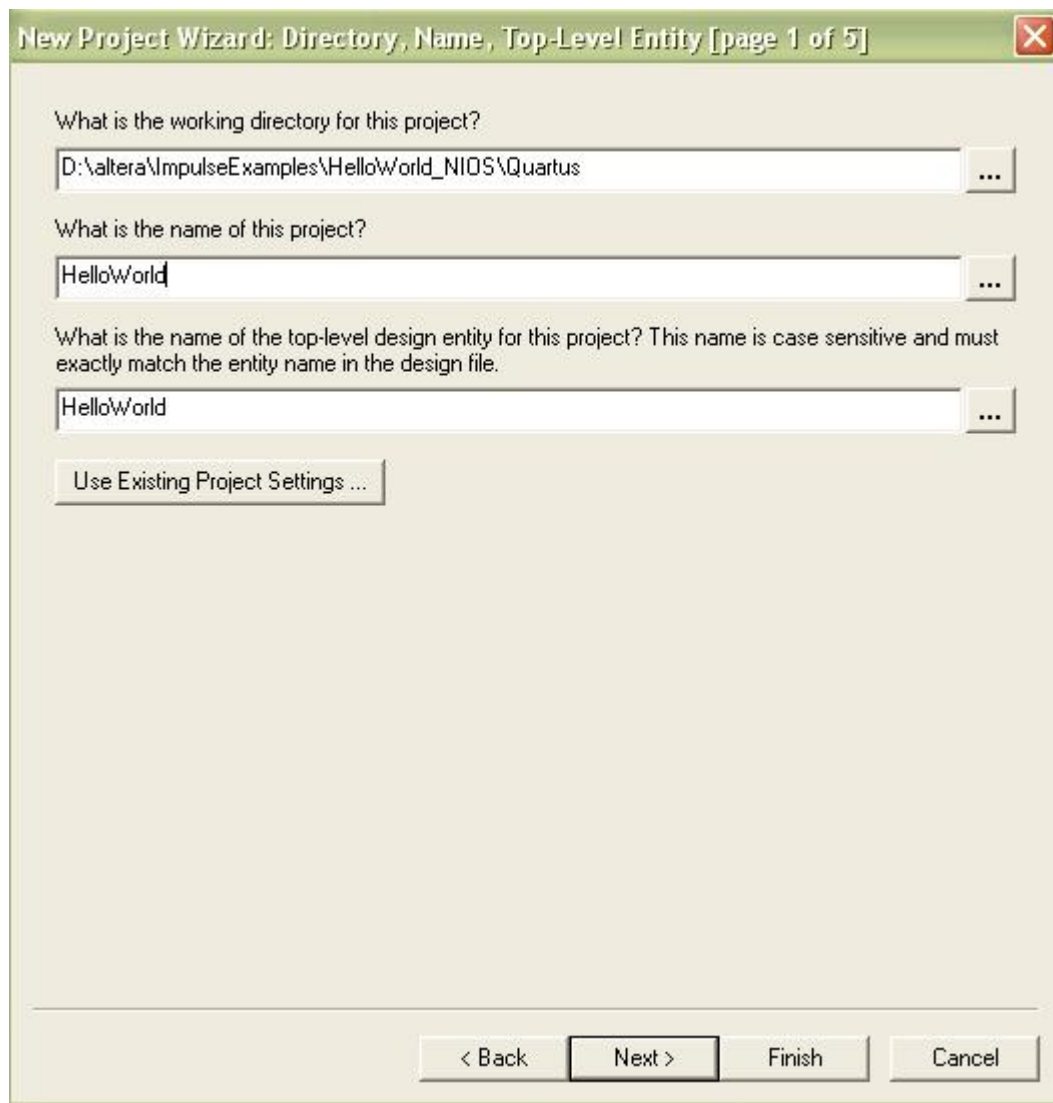
Now we'll move into the Altera tool environment. Begin by launching Altera Quartus II (from the **Windows Start -> All Programs -> altera -> Quartus II 8.0 -> Quartus II 8.0 (32-Bit)**). Open a new project by selecting **File -> New Project Wizard**.



In the field prompting you for the new project's working directory, use the browse button and find the directory (**Quartus**) to which you exported the hardware and software files in the previous step:



Select the **Quartus** directory and click **Open**. On page one of the **New Project Wizard** dialog, enter **HelloWorld** in both the project name and top-level design entity fields as shown:

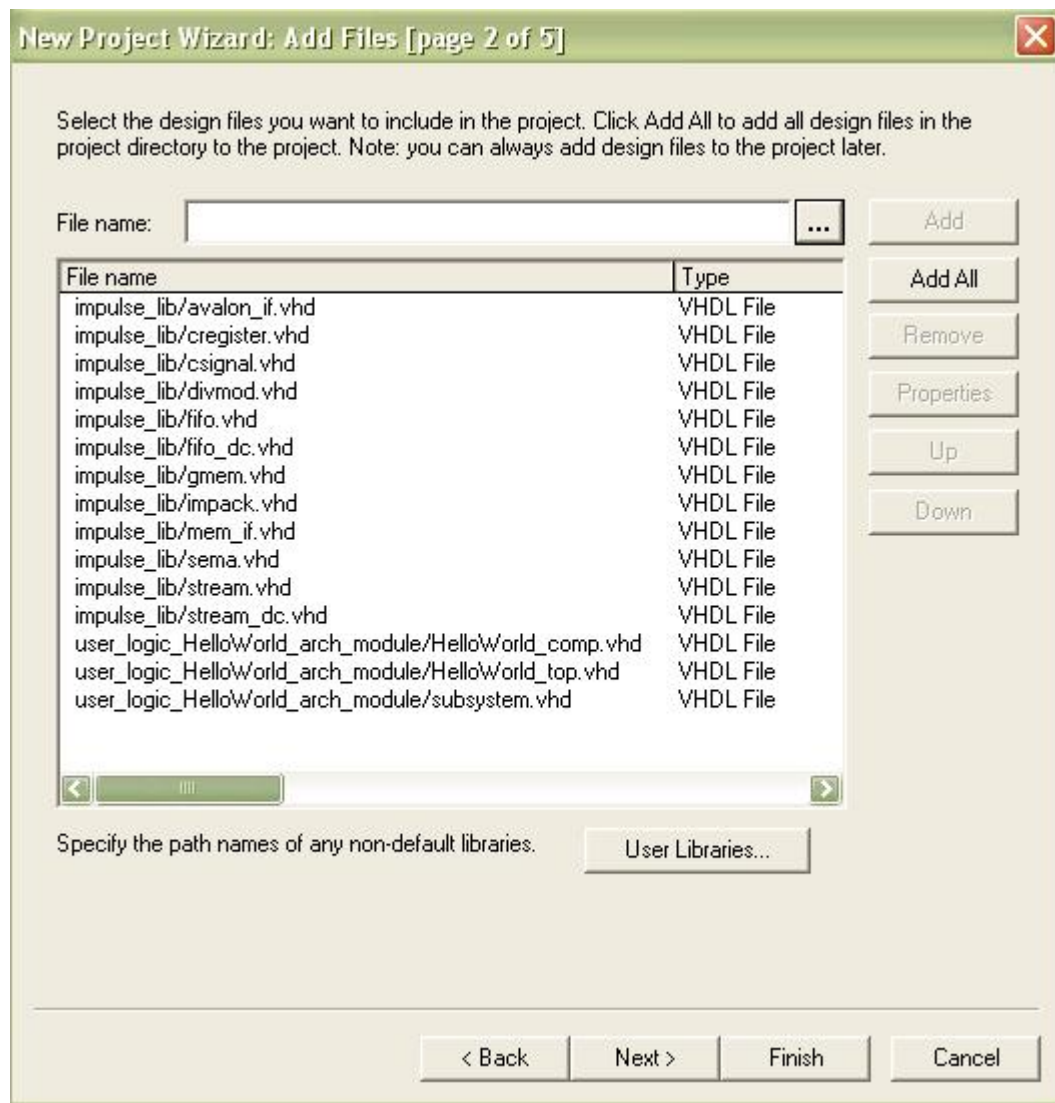


Now click Next.

Now you will import the VHDL files generated by CoDeveloper to your Quartus project. In the Add Files page (page 2), add the files in the following order:

1. Core logic files in the **user_logic_HelloWorld_arch_module** subdirectory: **subsystem.vhd**, **HelloWorld_top.vhd** and **HelloWorld_comp.vhd**
2. All .vhd files in the **impulse_lib** subdirectory

The files should be listed in the opposite order from which they were added (e.g. the **impulse_lib** files should be at the top of the list):



Click **Next** to proceed.

In the **Family and Device Settings** page (page 3) select the device you will be targeting. For this example we will choose **Cyclone III, EP3C25F324C6** device, with 324 pins and speed grade 6. This is the FPGA used in the **Cyclone III FPGA Evaluation Kit**.

New Project Wizard: Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.

Device family
Family: Cyclone III
Devices: All

Target device
☐ Auto device selected by the Filter
☒ Specific device selected in 'Available devices' list

Show in 'Available device' list
Package: Any
Pin count: 324
Speed grade: 6
☒ Show advanced devices
☐ HardCopy compatible only

Available devices:

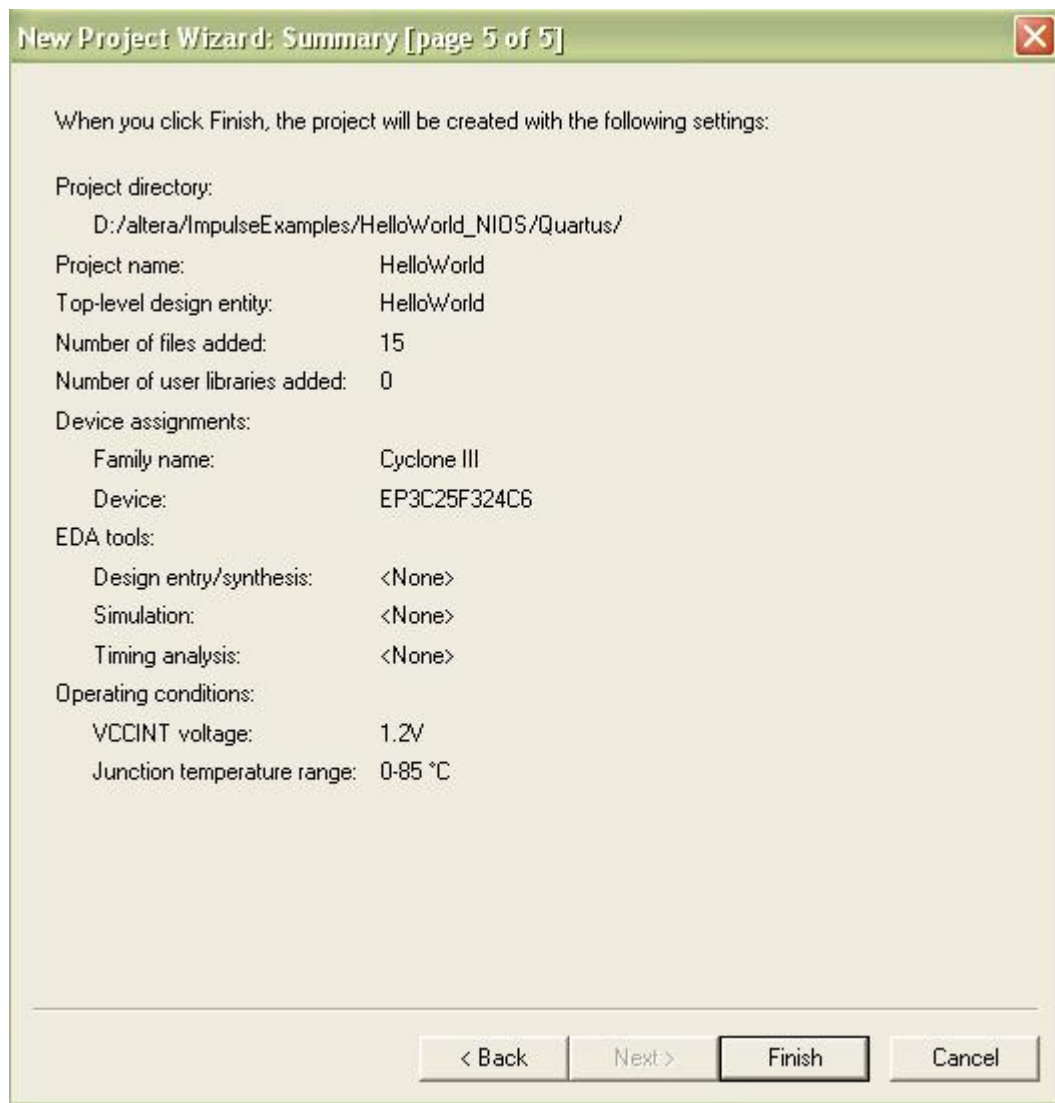
Name	Core v...	LEs	User I/...	Memor...	Embed...	PLL
EP3C25F324C6	1.2V	24624	216	608256	132	4
EP3C40F324C6	1.2V	39600	196	1161216	252	4

Companion device
HardCopy:
☒ Limit DSP & RAM to HardCopy device resource

< Back Next > Finish Cancel

Click **Next** again. Skip the **EDA Tool Settings** page (page 4) by again clicking **Next**.

You will see a **Summary** page listing the options you have chosen as shown below:



Click **Finish** to exit the Wizard and return to Quartus.

The next steps will demonstrate how to create and configure a hardware system with a Nios II processor and the necessary I/O interfaces for our sample application.

See Also

Step 6: [Creating the New Platform](#)

[Tutorial 1: Hello World on the Nios II platform](#)

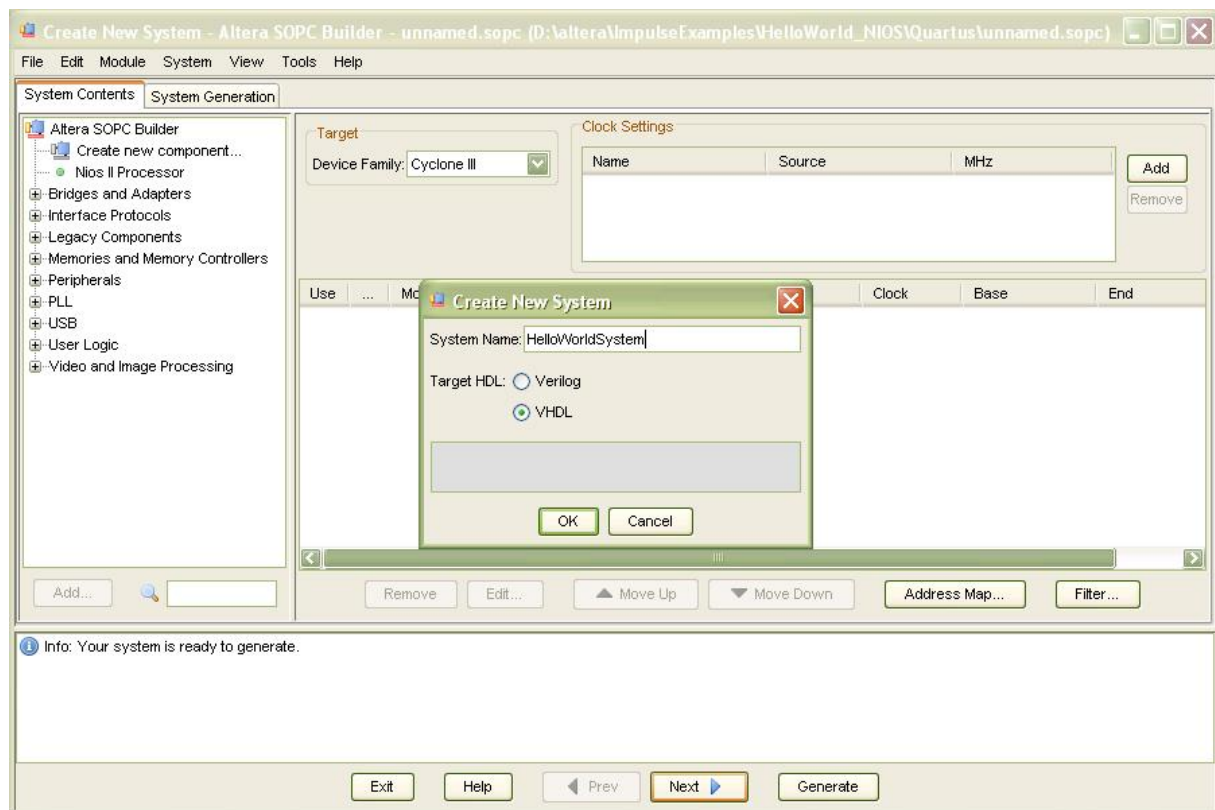
1.6 Creating the New Platform

Hello World Tutorial for Nios II, Step 6

Now that you have created a **Quartus** project using the wizard, you will need to specify additional information about your platform in order to support the requirements of your software/hardware application. These steps include the creation of a hardware system with a Nios II processor and the necessary I/O elements.

You will use **SOPC Builder** to create a hardware system containing an Altera Nios II embedded processor, the FPGA module created for the HelloWorld hardware process by CoBuilder, and several necessary peripherals. To do this, select **Tools -> SOPC Builder** to start SOPC Builder.

A new **SOPC Builder** window will appear. In the **Create New System** dialog that appears, enter **HelloWorldSystem** as the **System Name**, and specify **VHDL** for the **Target HDL Language**:



Click **OK** to continue.

Note: the System Name that you specify in this step must be a valid VHDL identifier: specifically, it must not begin with a numeric character or include spaces or other non-alphanumeric characters other than the underscore character (_).

See Also

Step 7: [Configuring the New Platform](#)

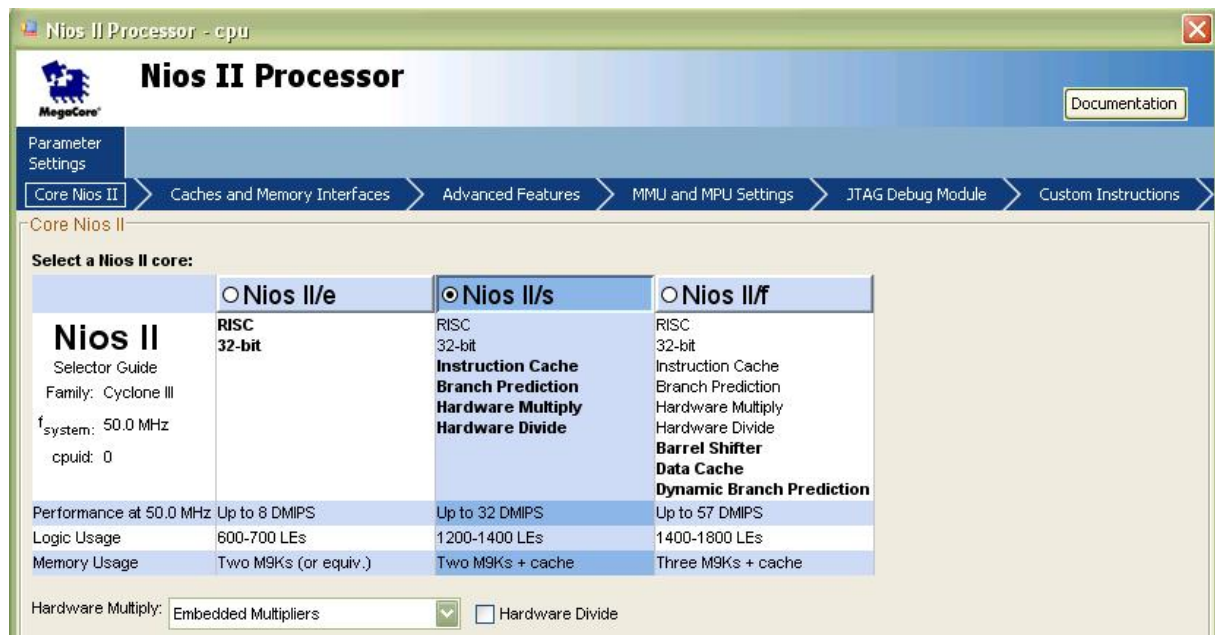
[Tutorial 1: Hello World on the Nios II platform](#)

1.7 Configuring the New Platform

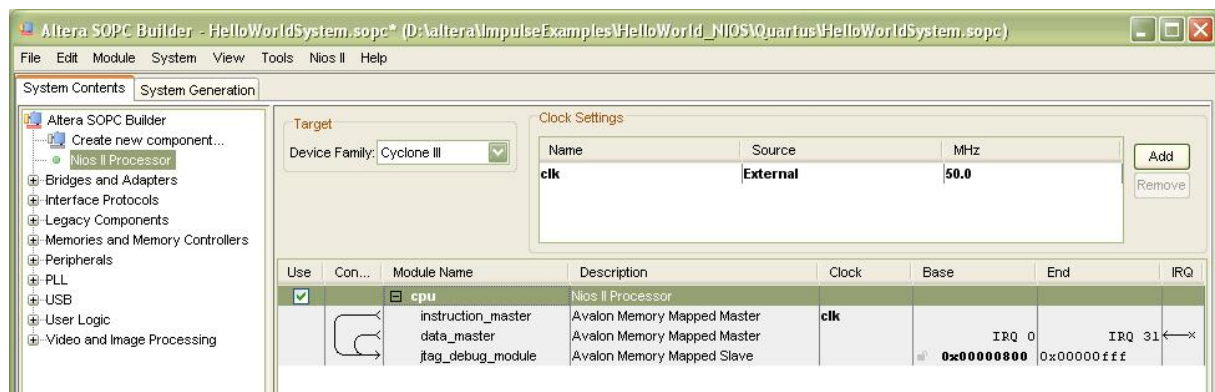
Hello World Tutorial for Nios II, Step 7

The following instructions will lead you through the process of creating your Nios II-based platform using SOPC Builder. This process requires many steps, but will only need to be done once for each new project that you create.

We'll begin by adding the largest component of the **HelloWorld** system, the Nios II processor. From the **System Contents** tab (on the left side of the **SOPC Builder** window), double-click **Nios II Processor** under **Altera SOPC Builder**. The **Nios II Processor - cpu** configuration Wizard will appear. Select the **Nios II/s** core as shown below:



Click **Finish** to add the Nios II CPU to the system and return to **SOPC Builder**. A module called **cpu** appears in the SOPC window.



Next, you must add the necessary peripherals to the new Nios II system. If you are not familiar with how to do this in **SOPC Builder**, you may wish to review the information provided in your **Nios II Development Kit** documents, and in particular the tutorials provided by Altera. Refer to the instructions provided by Altera in the following file:

http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf

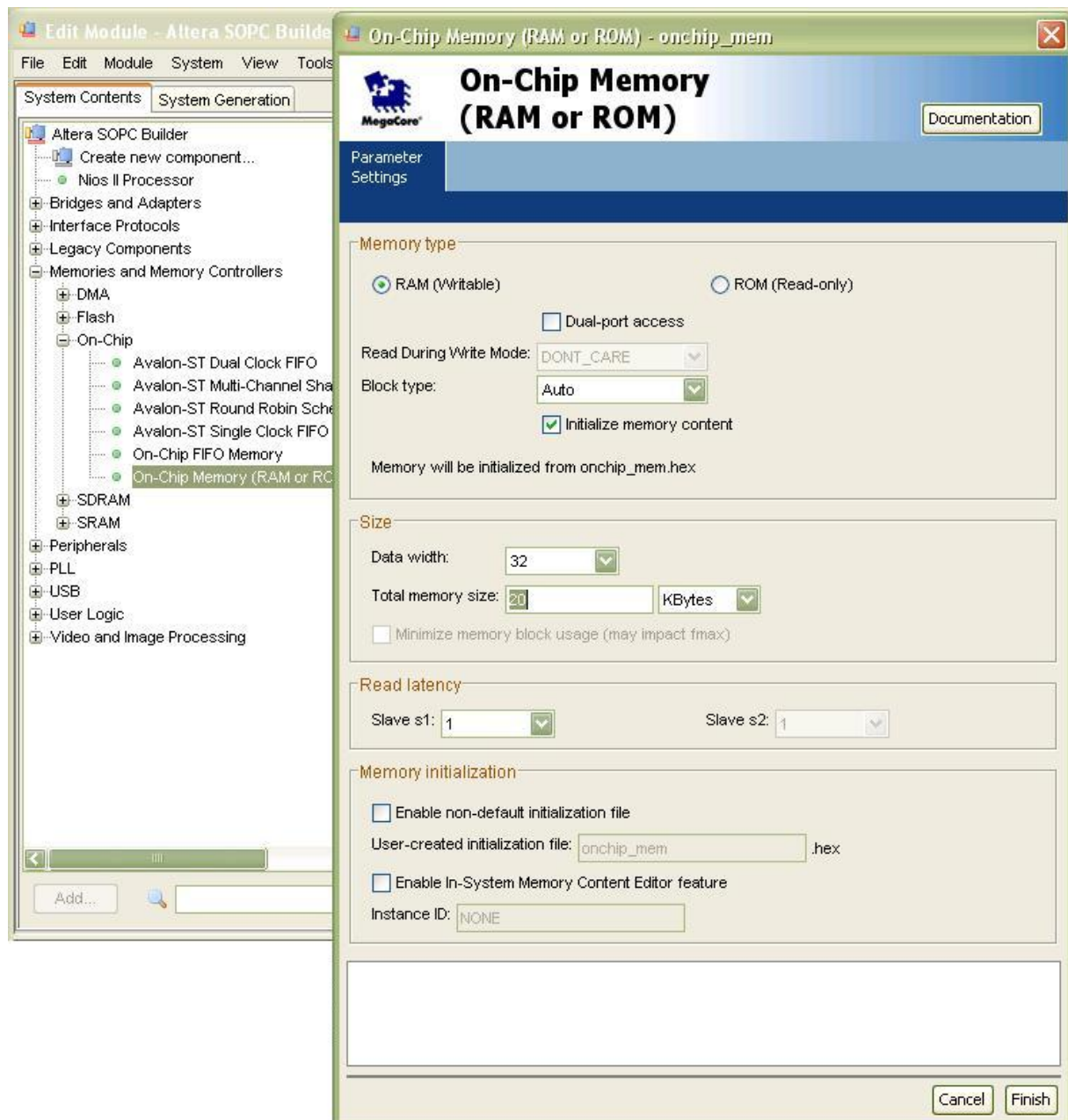
Using the methods described in the Altera documentation (and summarized below), you will need to add the following components:

On-Chip Memory

To add the on-chip memory peripheral, **onchip_mem**, perform the following steps:

Locate **Memories and Memory Controllers -> On-Chip -> On-Chip Memory (RAM or ROM)** and double-click to add. The **On-Chip Memory (RAM or ROM) - onchip_mem** wizard displays.

In the **Size** box, type in **20** as memory size, and make sure the unit is **KBytes**:



Click **Finish**. You are returned to the **Altera SOPC Builder** window.

JTAG UART Interface

The **JTAG UART** is used for communication between the board and the host machine and for debugging software running on the Nios II processor. To add the **JTAG UART** peripheral, **jtag_uart**, perform the following steps:

Locate **Interface Protocols** -> **Serial** -> **JTAG UART**, and double-click to add. The **JTAG UART - jtag_uart** wizard displays.

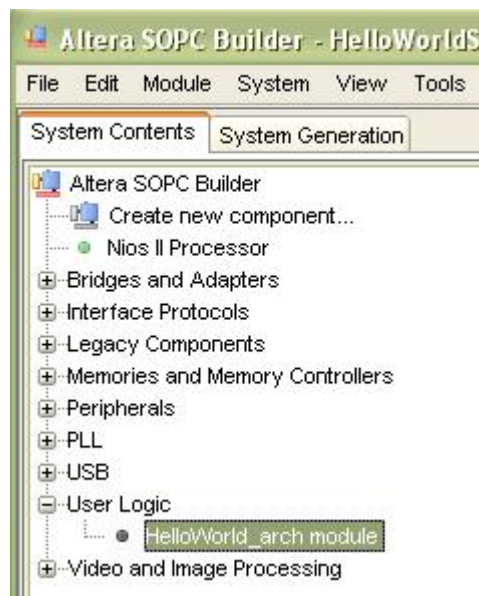
Leave all options at their default settings.



Click **Finish**. You are returned to the **Altera SOPC Builder** window.

Adding the Hardware Process Module "HelloWorld_arch"

Now add the **HelloWorld_arch** module, which implements the HelloWorld hardware process. Locate **User Logic** -> **HelloWorld_arch** module and double-click to add:



You have now completed adding the necessary peripherals. The modules and their associated names should appear as shown below:

Target: Device Family: Cyclone III

Clock Settings:

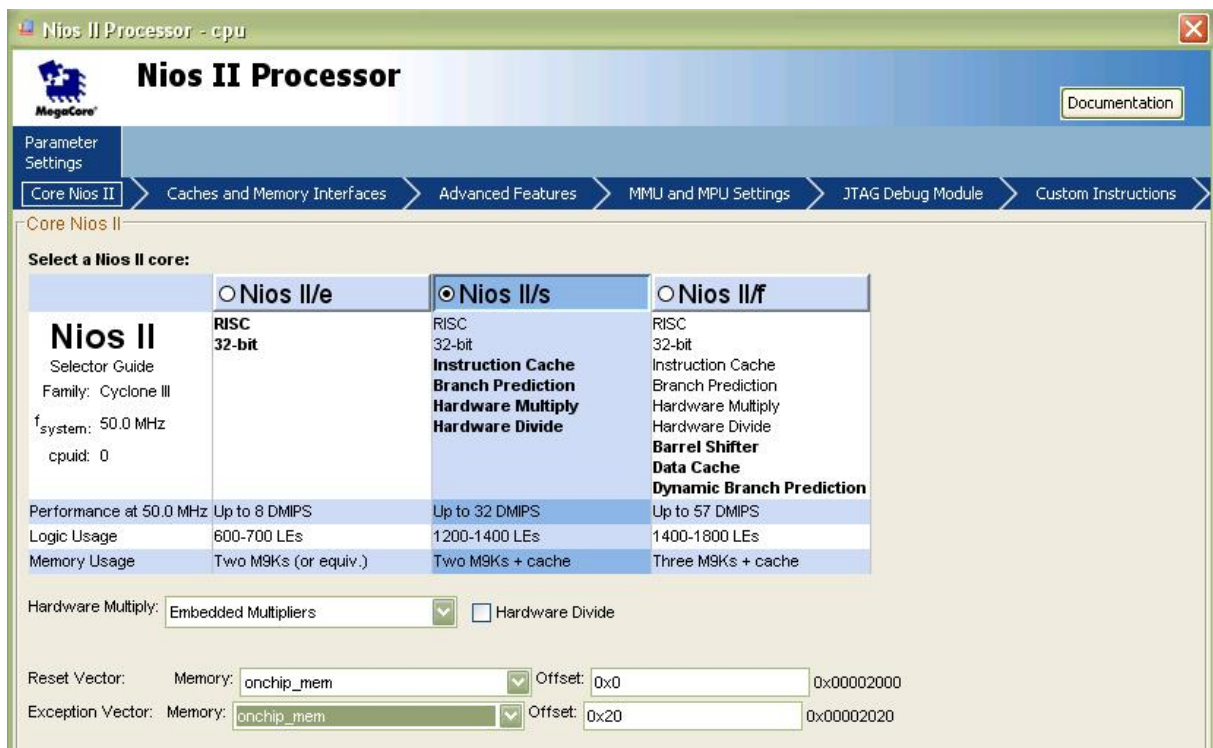
Name	Source	MHz
clk	External	50.0

Use: Con... Module Name Description Clock Base End IRQ

Use	Con...	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		cpu	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	clk			
		data_master	Avalon Memory Mapped Master	clk			
		jtag_debug_module	Avalon Memory Mapped Slave	clk			
		onchip_mem	On-Chip Memory (RAM or ROM)				
		s1	Avalon Memory Mapped Slave	clk	0x00000800	0x00000fff	IRQ 0
		jtag_uart	JTAG UART				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk	0x00002000	0x00006fff	
		user_logic_HelloWorld_arch_module_classic_0	HelloWorld_arch module				
		p_hear_hello_hello_in	Avalon Memory Mapped Slave	clk	0x00000000	0x00000007	
					0x00000010	0x0000001f	

Editing CPU Settings

Now double-click the **cpu** module to edit its settings. Change the Memory for both Reset Vector and Exception Vector to **onchip_mem**, as shown:



Click OK to save the changes.

Assigning Addresses

Next, select the SOPC Builder menu **System -> Auto-Assign Base Addresses**. This will automatically assign addresses to each memory-based module. Address space conflict can be avoided. The system will look like the following after the address assigning is done.

Use	Con...	Module Name	Description	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		cpu	Nios II Processor				
		instruction_master	Avalon Memory Mapped Master	clk			
		data_master	Avalon Memory Mapped Master	clk			
		jtag_debug_module	Avalon Memory Mapped Slave	clk			
<input checked="" type="checkbox"/>		onchip_mem	On-Chip Memory (RAM or ROM)	clk			
		s1	Avalon Memory Mapped Slave	clk	0x00008000	0x0000cfff	
<input checked="" type="checkbox"/>		jtag_uart	JTAG UART	clk			
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk	0x00011010	0x00011017	
<input checked="" type="checkbox"/>		user_logic_HelloWorld_arch_module_classic_0	HelloWorld_arch module	clk			
		p_hear_hello_hello_in	Avalon Memory Mapped Slave	clk	0x00011000	0x0001100f	

Save the system settings by selecting from the menu **File -> Save**.

Your new Nios II platform is ready for system generation.

See Also

Step 8: [Generating the System](#)

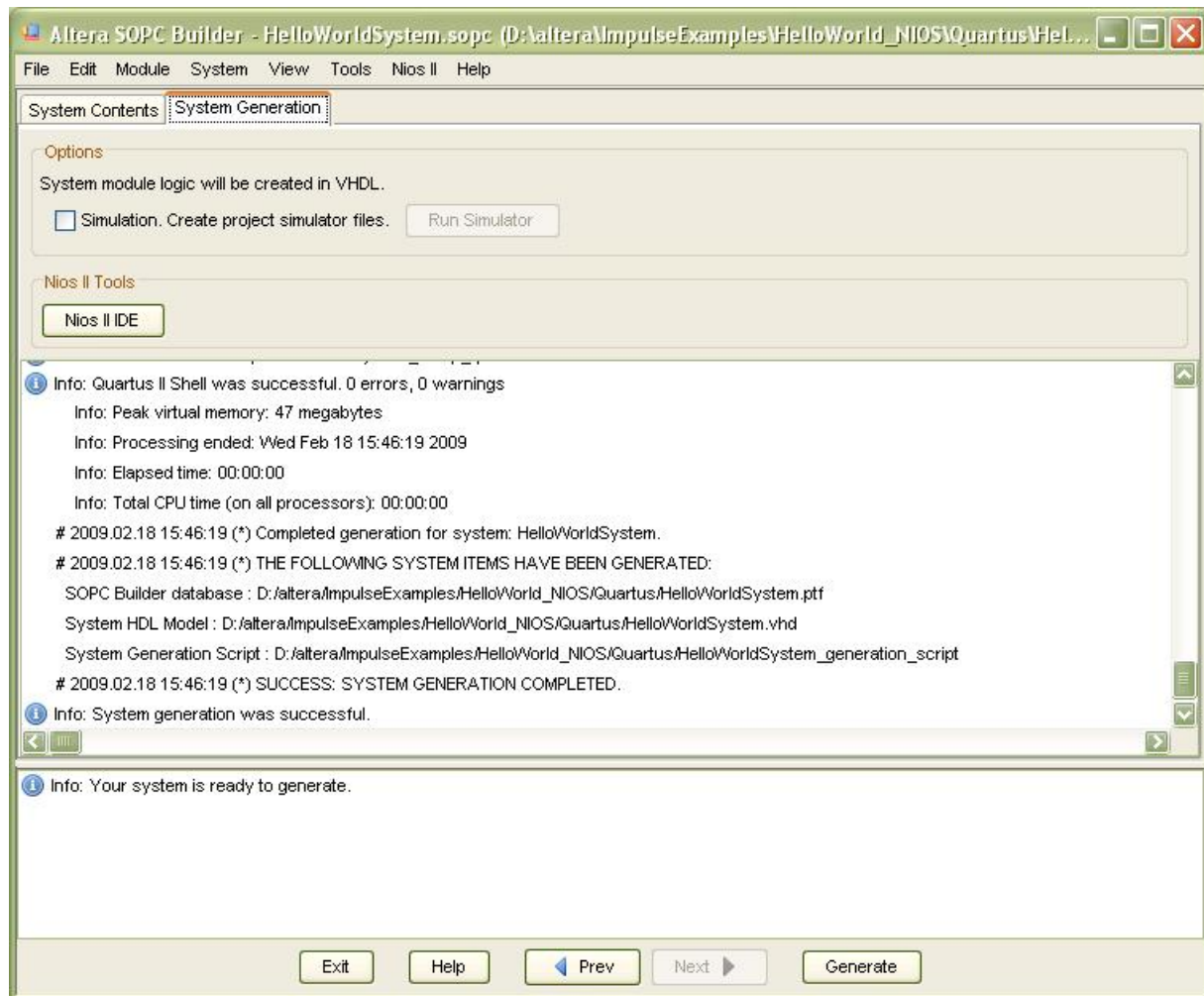
[Tutorial 1: Hello World on the Nios II platform](#)

1.8 Generating the System

Hello World Tutorial for Nios II, Step 8

At this point you have set up and configured your new Nios II-based platform, including the hardware module generated by CoBuilder, and can now start the system generation process within SOPC Builder.

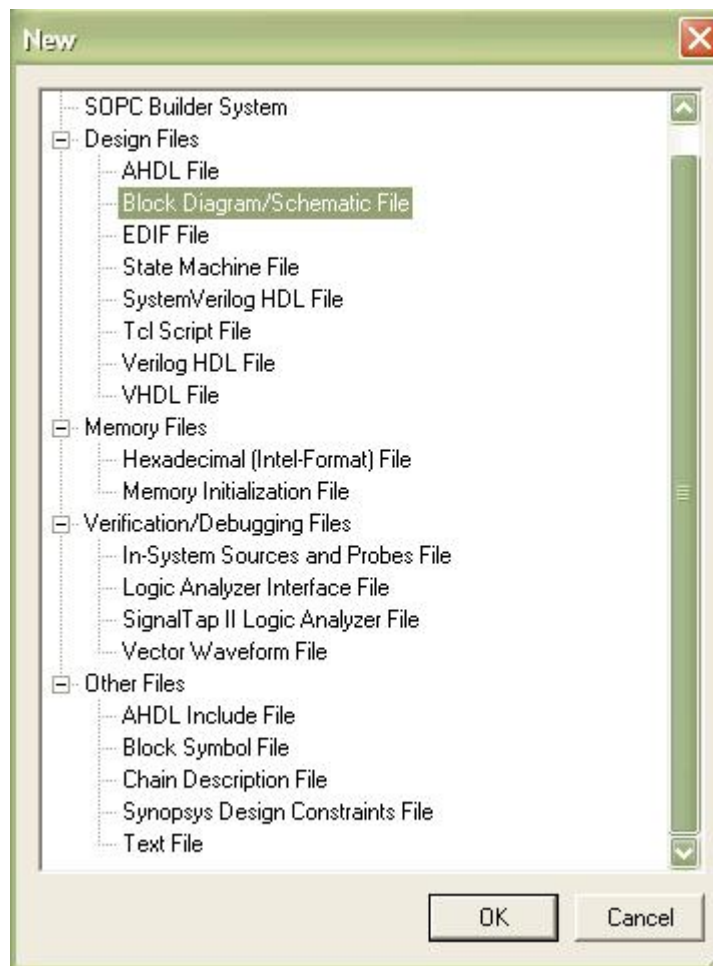
Click **Generate** on the bottom of the **SOPC Builder** window to generate the system. The **SOPC Builder** will automatically switch to the **System Generation** tab and display generation information. Make sure the **Simulation** option is unchecked to save time. This process may take several minutes.



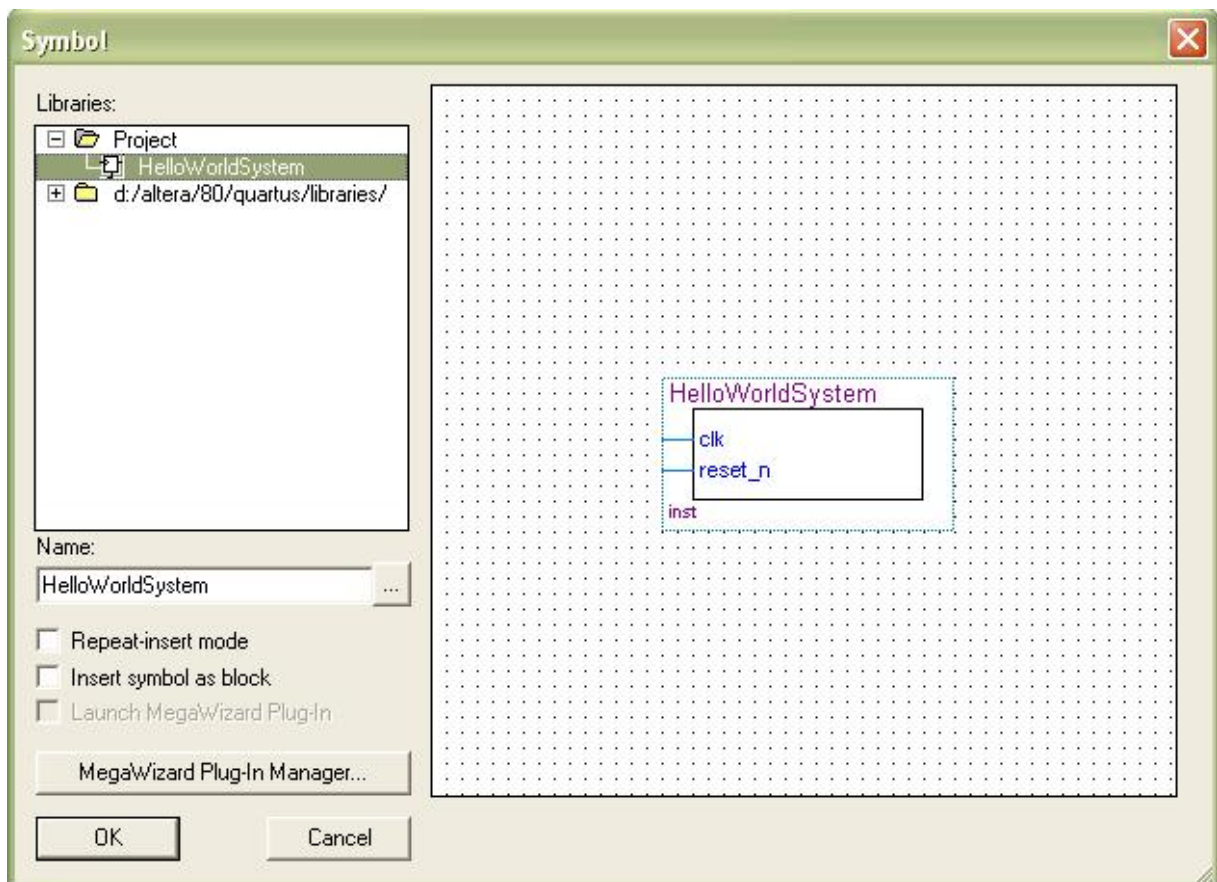
When generation is complete you may exit SOPC Builder and return to Quartus.

Creating a Block Diagram

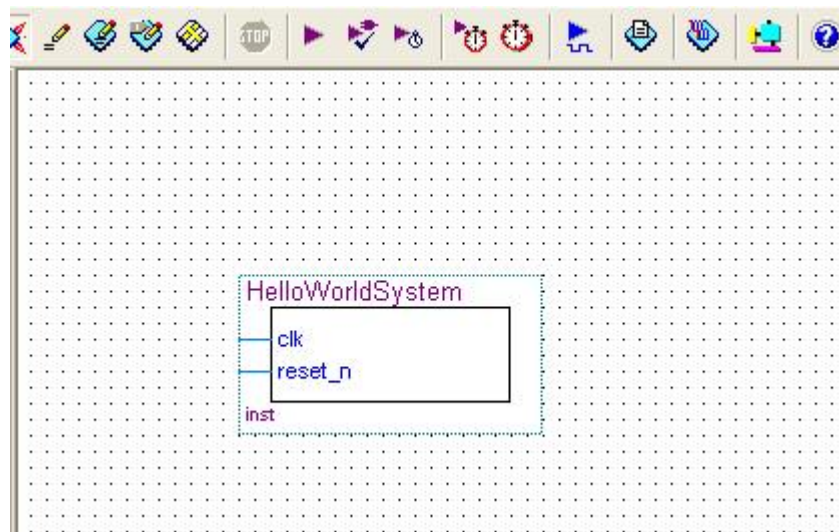
Now you will need to create a block diagram to connect the complete SOPC Builder-generated system (which includes the HelloWorld hardware process module, the Nios II processor, and peripherals) to the pins on the FPGA. Open a **File -> New** dialogue from the Quartus menu, and select **Block Diagram/Schematic File** as shown below:



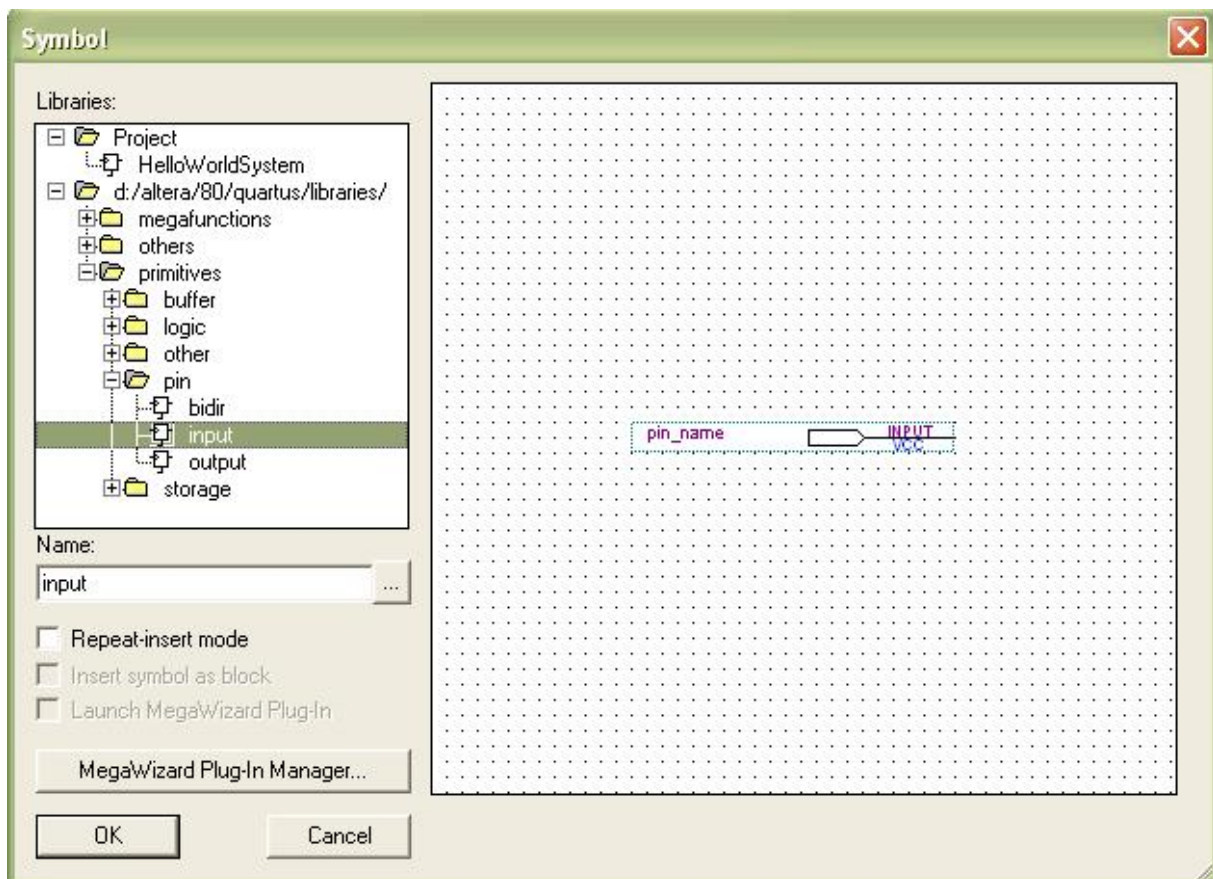
Click OK, and a new DBF file will appear. Now add the block representing the SOPC Builder-generated system. Double-click anywhere in the new block diagram file to bring up the **Symbol** dialog. Open the **Project** folder and select the **HelloWorldSystem** symbol as shown below:



Click OK, and a symbol outline appears attached to the mouse pointer. Drag the symbol to a proper location in the Block Diagram.

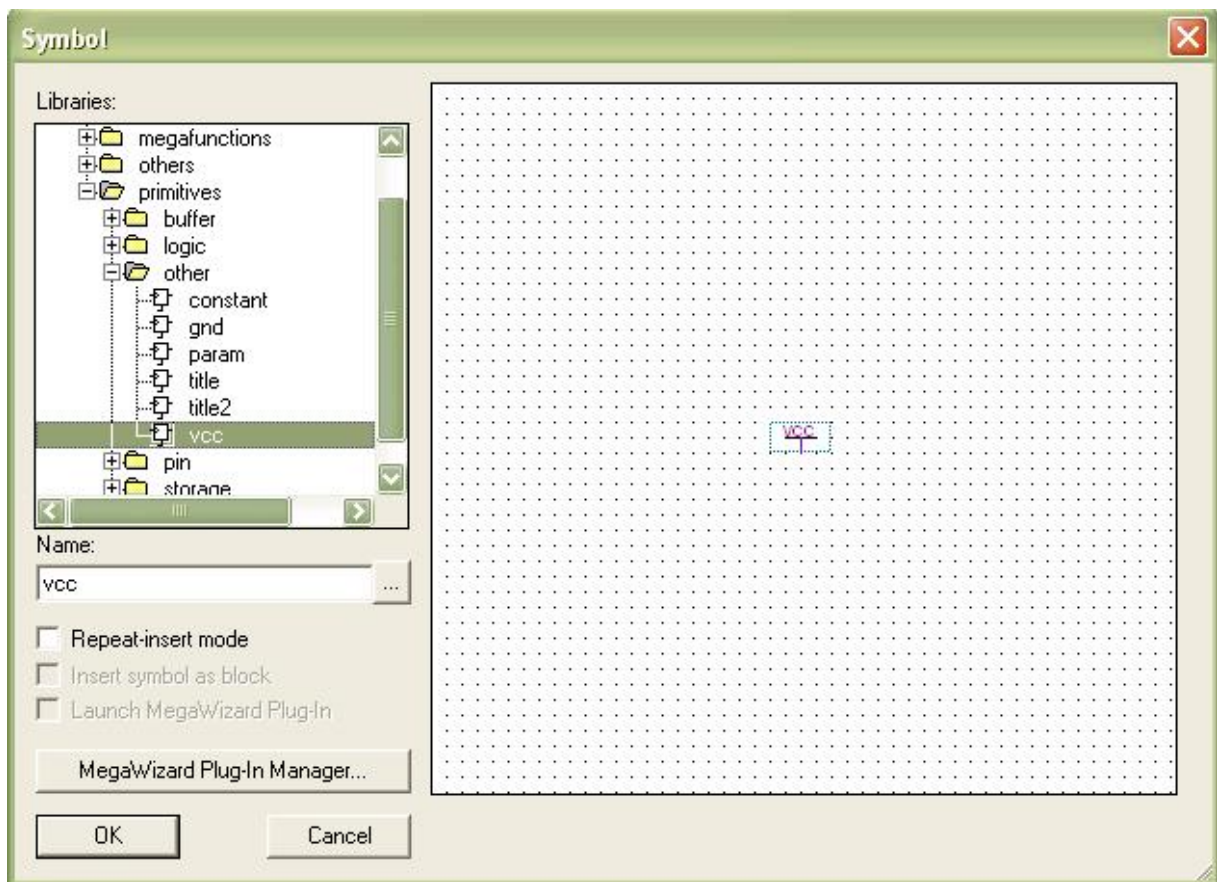


Next, bring up the **Symbol** dialog again, and select **input** port as shown below:

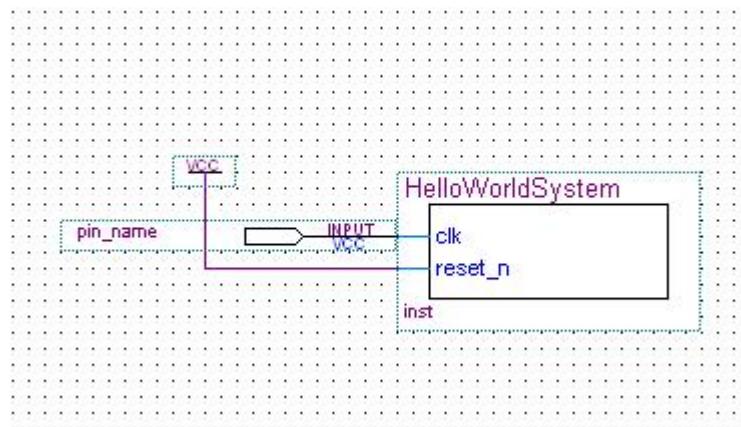


Click OK to add the input port to the block diagram.

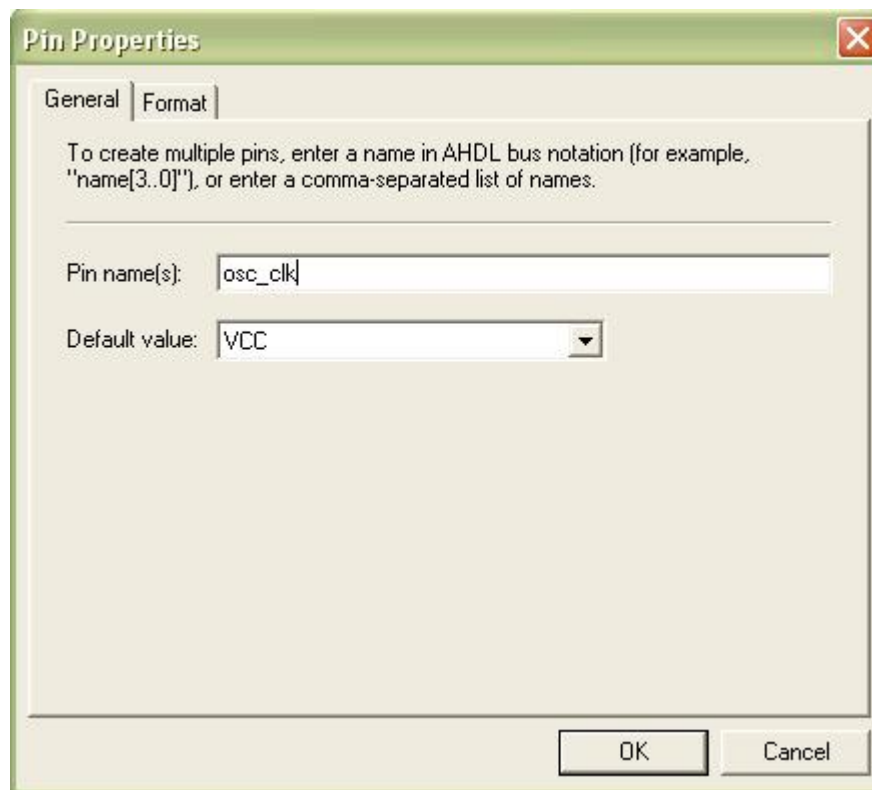
Next, bring up the **Symbol** dialog again, and select **vcc** as shown below:



Click OK to add the **vcc** to the block diagram. Align the **input** port with the **clk** pin, and the **vcc** to the **reset_n** pin on the block diagram as shown:



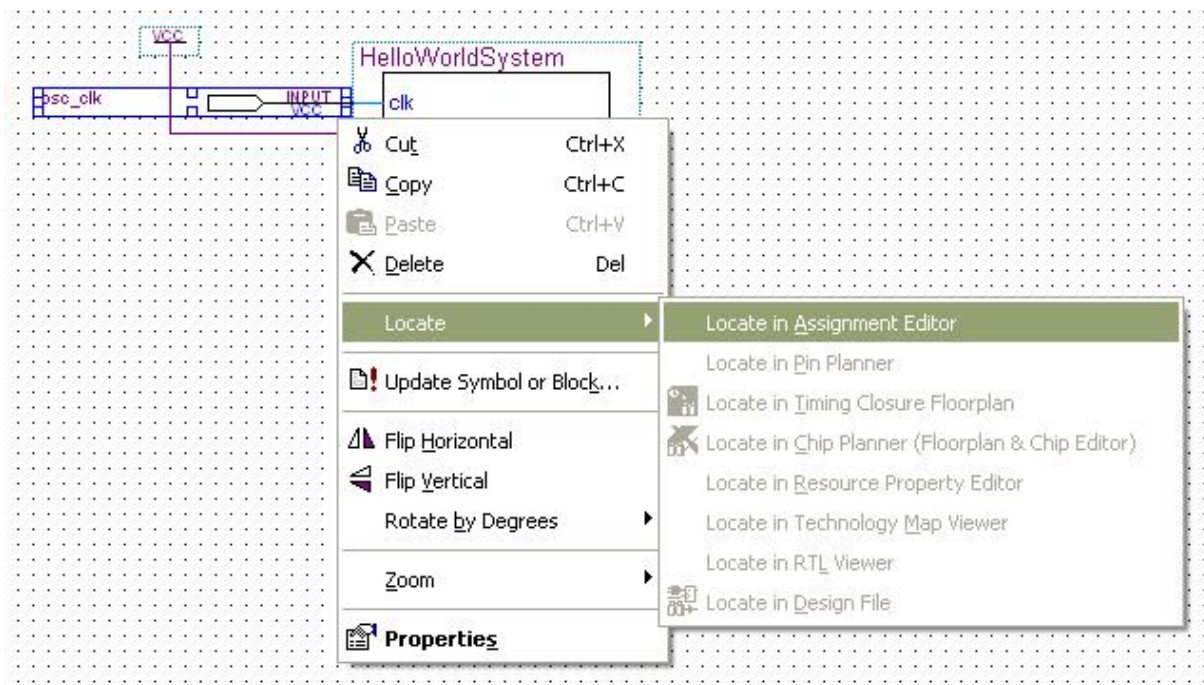
Change the name of the **input** port by double-clicking it, and then type in the name **osc_clk** in the **Pin name** box.



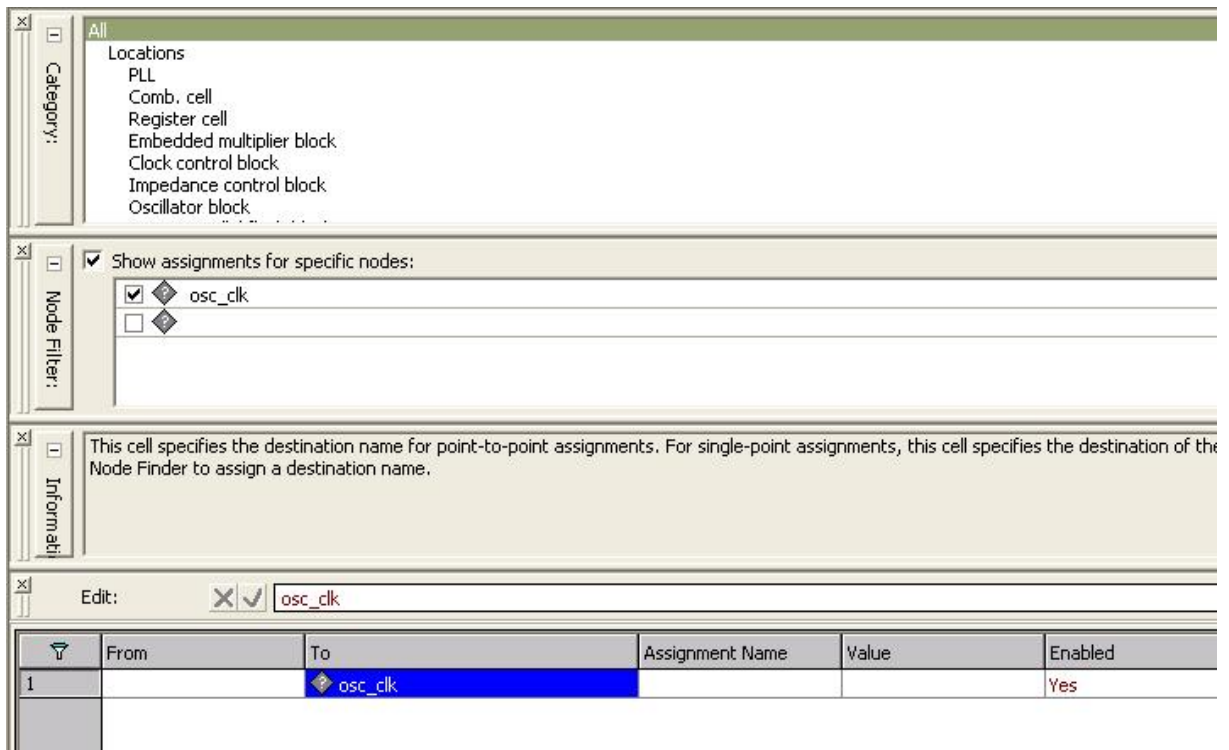
Click OK to accept the change.

Pin Assignment

The next step is to assign pins. Here we only have one pin that needs assignment. Right-click the **osc_clk** pin, and select **Locate in Assignment Editor**.



The Assignment Edit window will appear with the **osc_clk** pin:



In the **Assignment Name** column, choose **Location** from the list. In the **Value** column, type in **V9** as the pin location.

Edit: [X] [✓] v9					
	From	To	Assignment Name	Value	Enabled
1		osc_clk	Location	v9	Yes
2		osc_clk			Yes

Save your project. Save the block diagram file as **HelloWorld.bdf**. This will be your top-level design file for the **HelloWorld** project.

Your project is now ready for bitmap generation and subsequent downloading.

Tip: you may wish to save your Altera project at this point and save a copy for later use with other CoBuilder-generated projects.

See Also

Step 9: [Generating the FPGA Bitmap](#)

[Tutorial 1: Hello World on the Nios II platform](#)

1.9 Generating the FPGA Bitmap

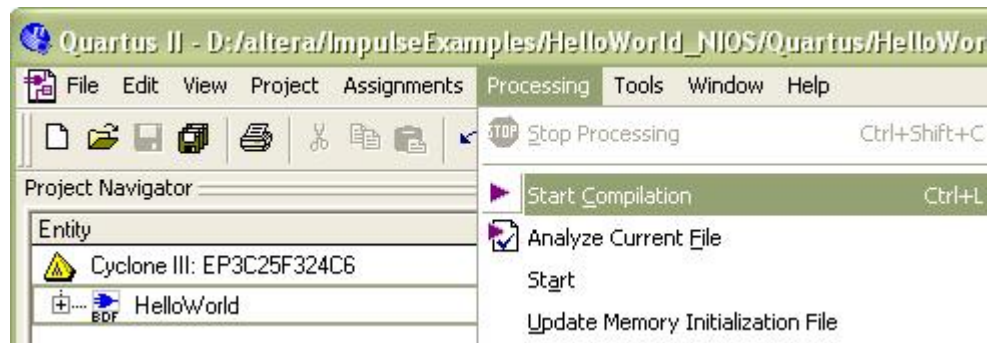
Hello World Tutorial for Nios II, Step 9

At this point, if you have followed the tutorial steps carefully you have successfully:

- Generated hardware and software files from the CoDeveloper environment.
- Created a new Altera Quartus II project and used SOPC Builder to create a new Nios II-based platform.
- Imported your CoDeveloper-generated files to the Altera tools environment.
- Completed a block diagram and assigned pins for the selected FPGA device.

You are now ready to compile your design, generate the bitmap and download the complete application to the target platform. This process is not complicated (at least in terms of your actions at the keyboard) but can be quite time consuming due to the large amount of processing that is required within the Altera tools.

To generate the bitmap, select **Processing -> Start Compilation** as shown below:



Note: this process may require a few minutes to complete, depending on the speed and memory of your development system.

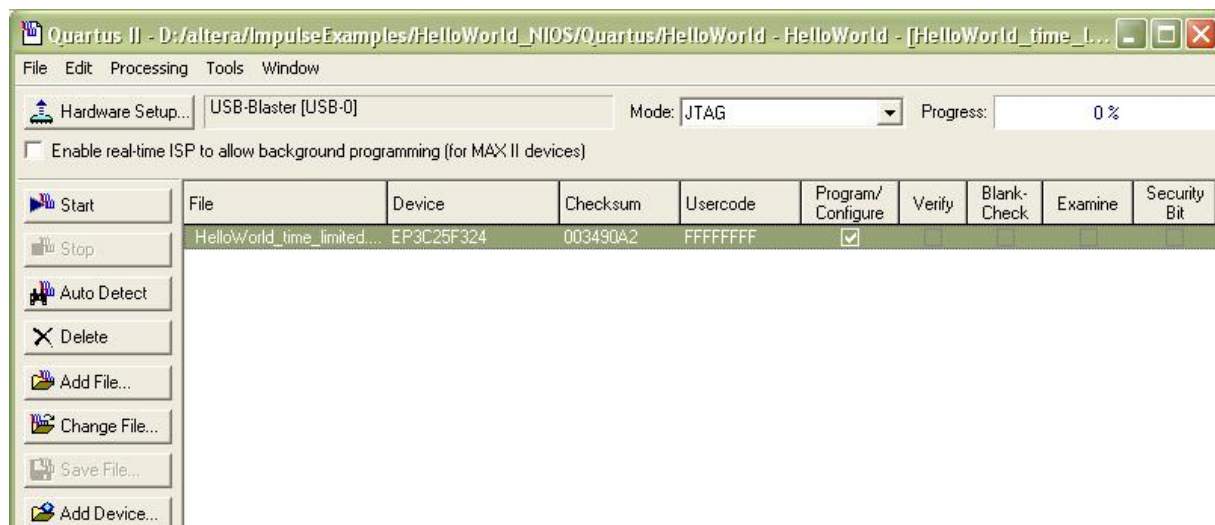
During compilation, Quartus will analyze the generate VHDL source files, synthesize the necessary logic, and create logic that is subsequently placed and routed into the FPGA along with the Nios II processor and interface elements that were previously specified. The result will be a bitmap file (in the appropriate Altera format) ready for downloading to the device.

When the bitstream has been generated, select **Tools -> Programmer** to open the new programming file. Select **File -> Save As** and save the chain description file as **HelloWorld.cdf** (make sure the **Add file to current project** option is selected).

The programming file HelloWorld.sof should be visible in the programming window. If it is not, select **Add File...** and open **HelloWorld.sof**.

Enable Program/Configure for HelloWorld.sof and make sure your programming hardware (e.g., the **USB-Blaster [USB-0]** cable) is configured properly. Click **Start** to begin downloading the **HelloWorld.sof** file to the target device.

*Note: If you don't have the full license for **OpenCore Plus** megafunctions, then a message will pop up. Click **OK** to continue. The bitmap file will be named **HelloWorld_time_limited.sof**. After the downloading is done, a **OpenCore Plus Status** message box will pop up. Don't click the **Cancel** button. Otherwise the downloaded bitmap will be reset.*



Now that the hardware is programmed, you are ready to download and run the software application on the platform.

See Also

Step 10: [Running the Application on the Platform](#)

[Tutorial 1: Hello World on the Nios II platform](#)

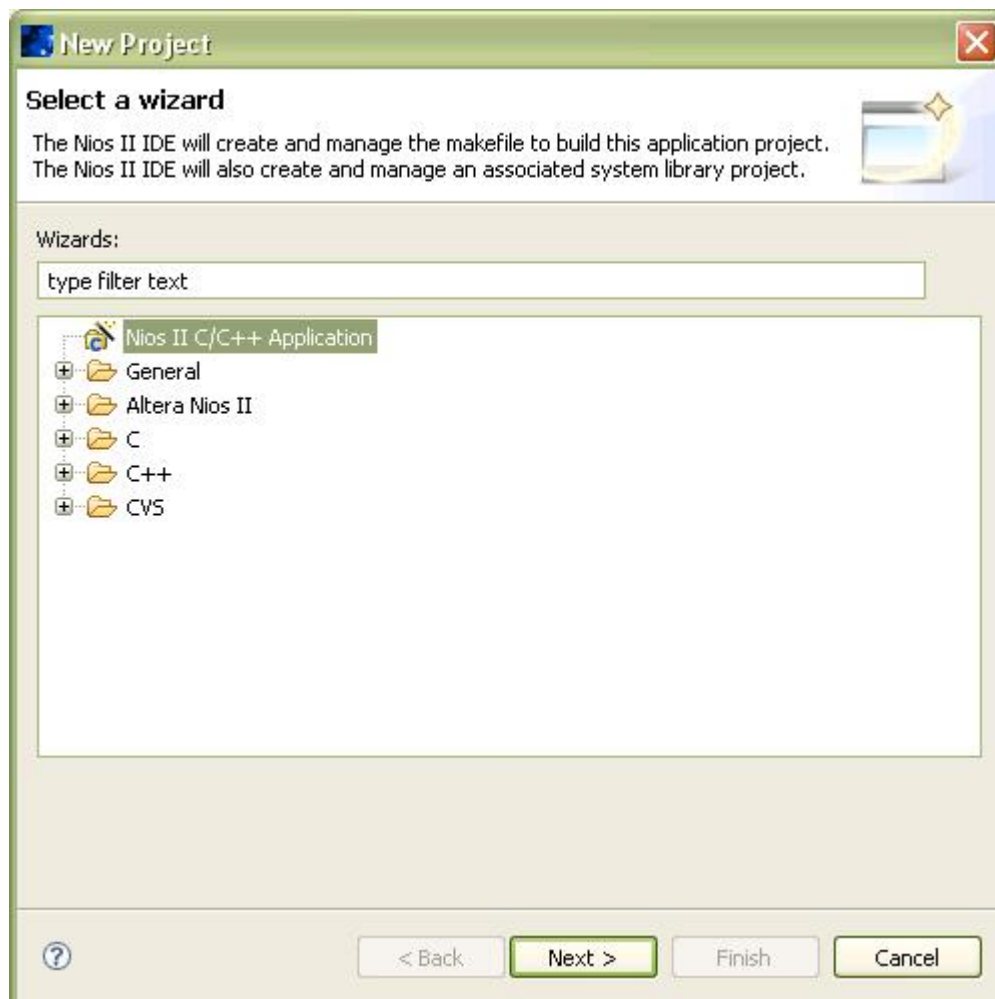
1.10 Running the Application on the Platform

Hello World Tutorial for Nios II, Step 10

In the previous step, you programmed the FPGA device with the design you created in **Quartus** and **SOPC Builder**. Now you will use **Altera Nios II IDE** to compile the software portion of the project and run it on the development board.

Begin by starting the **Nios II IDE** (usually available in the **Windows Start** menu -> **All Programs** -> **altera** -> **Nios II EDS 8.0** -> **Nios II 8.0 IDE**). If the **Workspace Launcher** dialog box appears, click **OK** to use the default workspace.

A **Nios II IDE** window will appear. To create a new project to manage the **HelloWorld** software files, select **File** -> **New** -> **Project...**; the following wizard will appear:



Select **Altera Nios II C/C++ Application** as shown, and click **Next**.

On the next page, select the project path, target hardware, and project template as follows, using the Browse buttons to locate the appropriate **Path** and **SOPC Builder System** options:

Name: HelloWorld

Specify Location: checked

Path: D:\altera\ImpulseExamples\HelloWorld_NIOS\Quartus\software\HelloWorld_arch
(The project path should point to the software files that were exported by CoDeveloper in

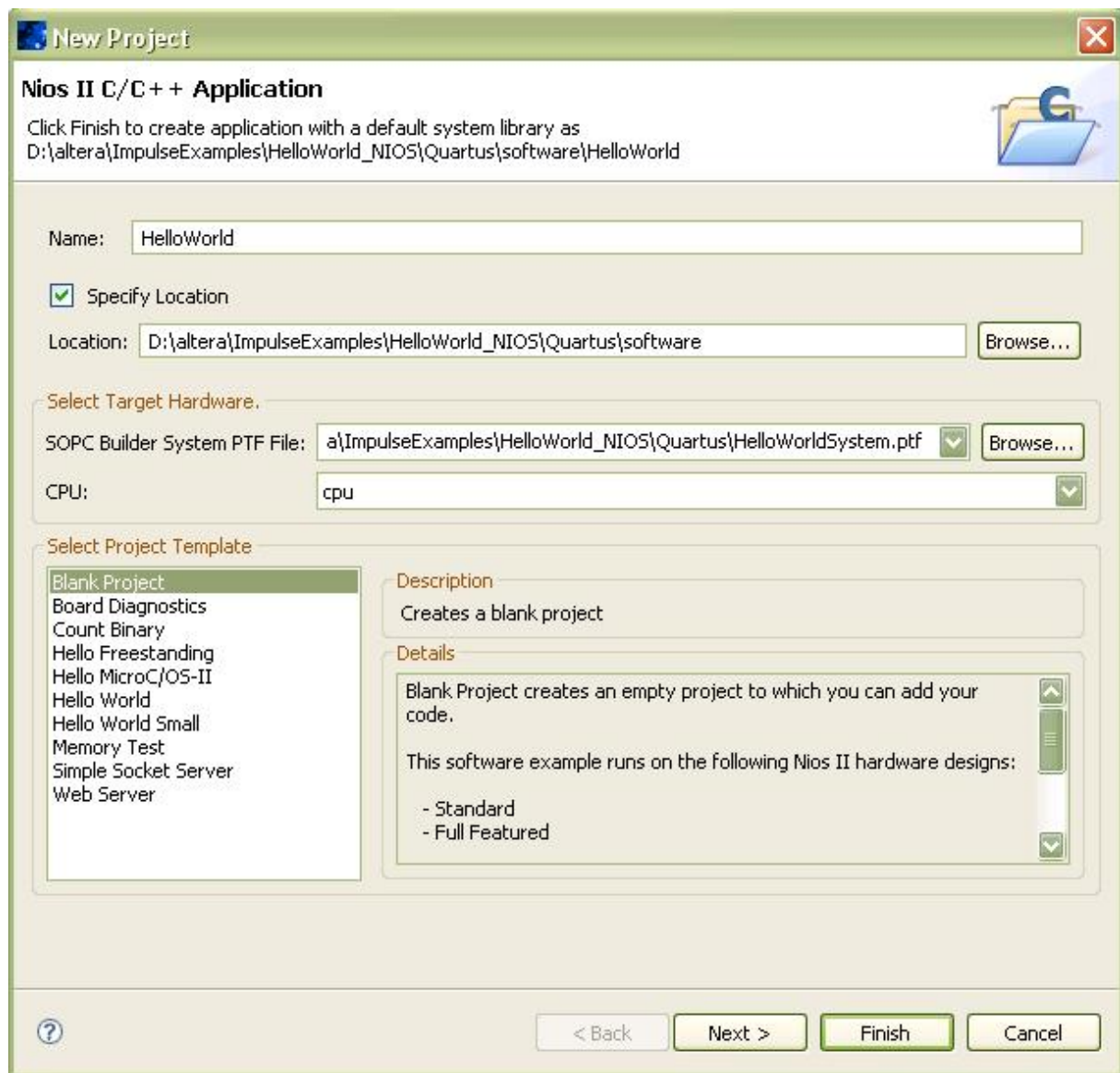
[Step 4.](#))

SOPC Builder System:

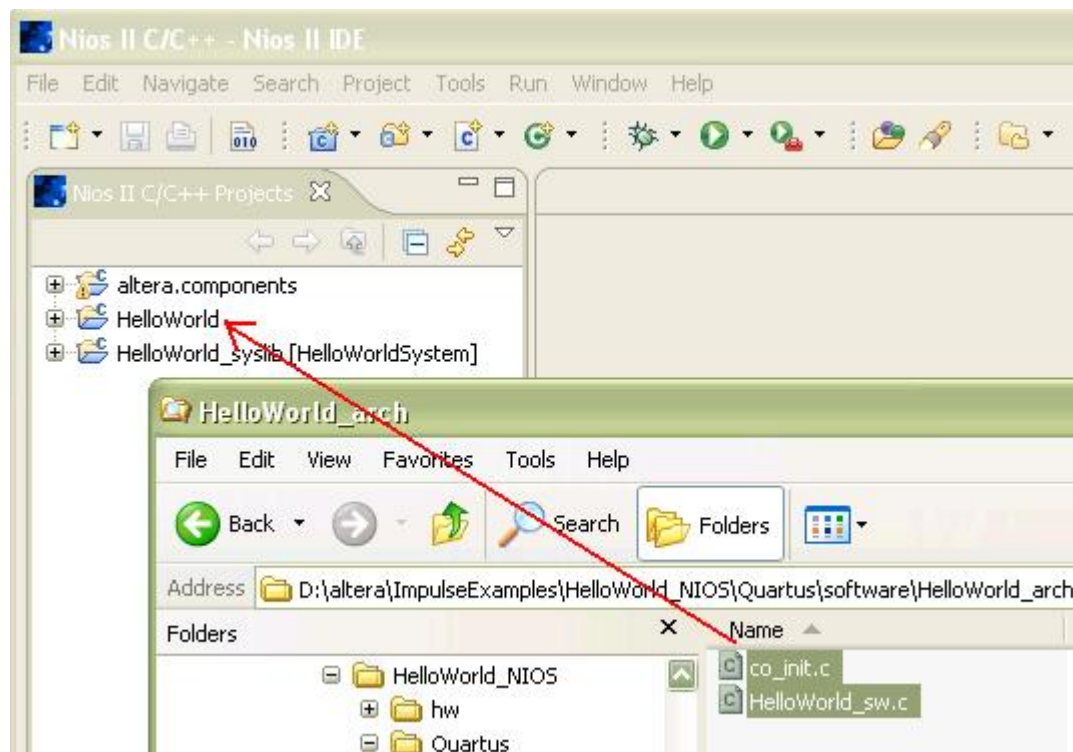
D:\altera\ImpulseExamples\HelloWorld_NIOS\Quartus\HelloWorldSystem.ptf
(This is the system .ptf file generated by SOPC Builder in [Step 8.](#))

CPU: cpu

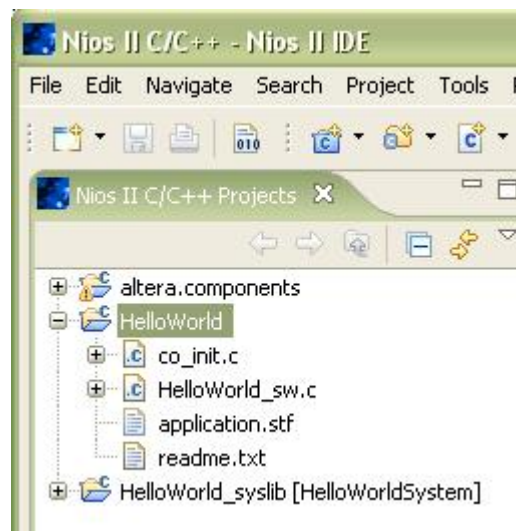
Select Project Template: Blank Project



Click **Finish** to create the new project. Two new projects (**HelloWorld** and **HelloWorld_syslib**) should appear in the **Nios II C/C++ Projects** window in the Nios II IDE. Copy the software files that were exported in [Step 4](#) (**co_init.c** and **HelloWorld_sw.c**) to the **HelloWorld** project as shown below.



The software files will appear under the **HelloWorld** project as shown below:

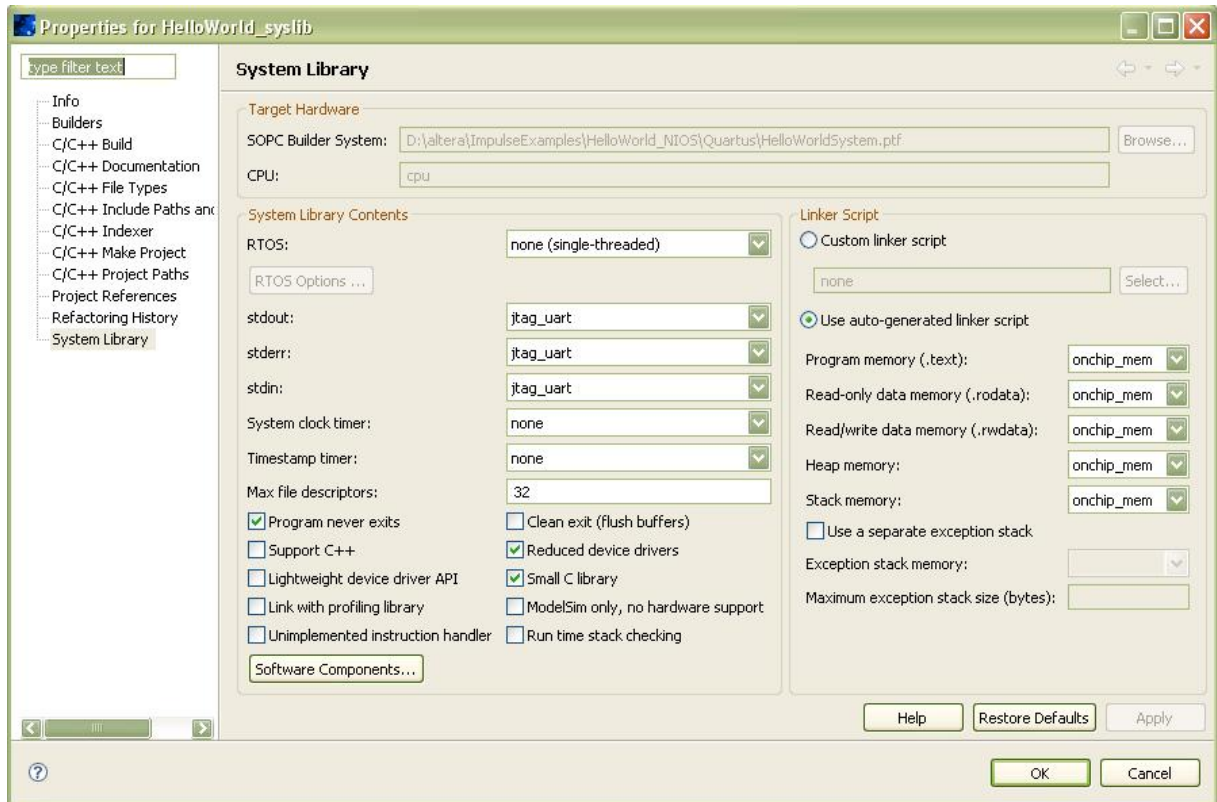


Before you compile the project, you need to adjust some of the project settings to minimize the size of the executable file in order to fit the 20 KBytes onchip memroy. Right-click **HelloWorld** and select **System Library Properties**. The **Poperties** dialog box of **HelloWorld_syslib** will appear.

Change to following settings:

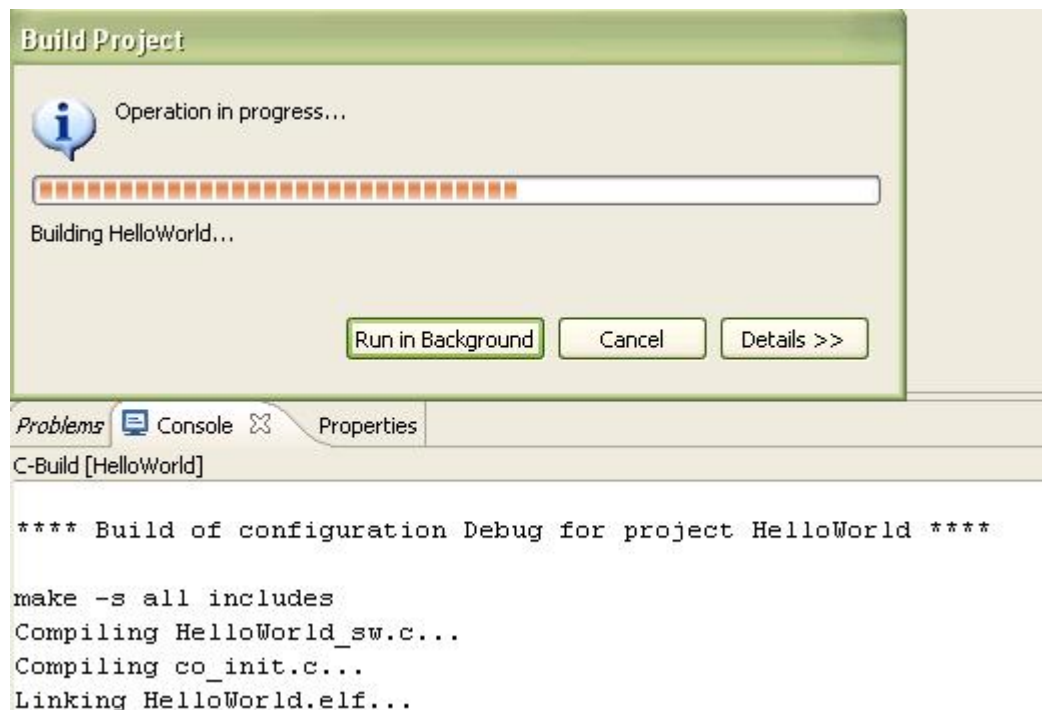
- check **Programs never exits**
- uncheck **Support C++**
- uncheck **Clean exit (flush buffers)**

- check **Reduced device drivers**
- check **Small C library**

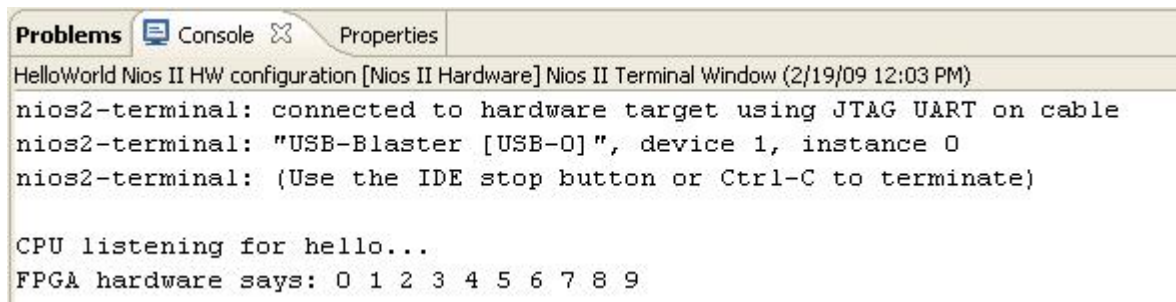


Click **OK** to save the changes.

Now build the project by right-clicking the **HelloWorld** project and selecting **Build Project**. The IDE will build the **HelloWorld_syslib** system library, which includes a driver for the Impulse C hardware module created by **CoBuilder**, along with the application software code in the **HelloWorld** project.



Once the software has finished building, you are ready to run the application on the hardware platform. Right-click the **HelloWorld** project and select **Run As -> Nios II Hardware**. You should see the following output in the Console window:



Compare the output to that of the desktop simulation done in [Step 2](#)--they should be the same.

Congratulations! You have successfully compiled a complete hardware/software application on the target FPGA device, without the need to write any low-level HDL code.

Although this sample application was trivial (in terms of the amount of logic generated on the FPGA), the process for substantially larger, more complex applications is essentially the same. You can follow these steps when compiling your own Impulse C applications.

See Also

[Tutorial 1: Hello World on the Nios II platform](#)