

My First Nios II for

Altera DE2-115 Board

CONTENTS

Chapter 1	Hardware Design.....	1
	1.1 Required Features.....	1
	1.2 Creation of Hardware Design.....	1
	1.3 Download Hardware Design to Target FPGA.....	44
Chapter 2	NIOS II IDE Build Flow	47
	2.1 Create the hello_world Example Project	47
	2.2 Build and Run the Program.....	51
	2.3 Edit and Re-Run the Program	53
	2.4 Why the LED Blinks	55
	2.5 Debugging the Application	56
	2.6 Configure System Library.....	58
Chapter 3	Programming the CFI Flash	60
	3.1 Modify the SOPC of the Project.....	60
	3.2 Modify the myfirst_niosii.v	71
	3.3 Re-assign pins	73
	3.4 Re-Configure System Library	75
	3.5 Programming the CFI Flash.....	76

Chapter 1 *Hardware Design*

This tutorial provides comprehensive information that will help you understand how to create a FPGA based SOPC system implementing on your FPGA development board and run software upon it.

1.1 Required Features

The Nios II processor core is a soft-core central processing unit that you could program onto an Altera field programmable gate array (FPGA). This tutorial illustrates you to the basic flow covering hardware creation and software building. You are assumed to have the latest Quartus II and NIOS II EDS software installed and quite familiar with the operation of Windows OS. If you use a different Quartus II and NIOS II EDS version, there will have some small difference during the operation. You are also be assumed to possess a DE2-115 development board (other kinds of dev. Board based on Altera FPGA chip also supported).

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

- Nios II processor core, that's where the software will be executed
- On-chip memory to store and run the software
- JTAG link for communication between the host computer and target
- hardware (typically using a USB-Blaster cable)
- LED peripheral I/O (PIO), be used as indicators

1.2 Creation of Hardware Design

This section describes the flow of how to create a hardware system including SOPC feature.

1. Launch Quartus II then select **File->New Project Wizard**, start to create a new project. See Figure 1-1 and Figure 1-2.

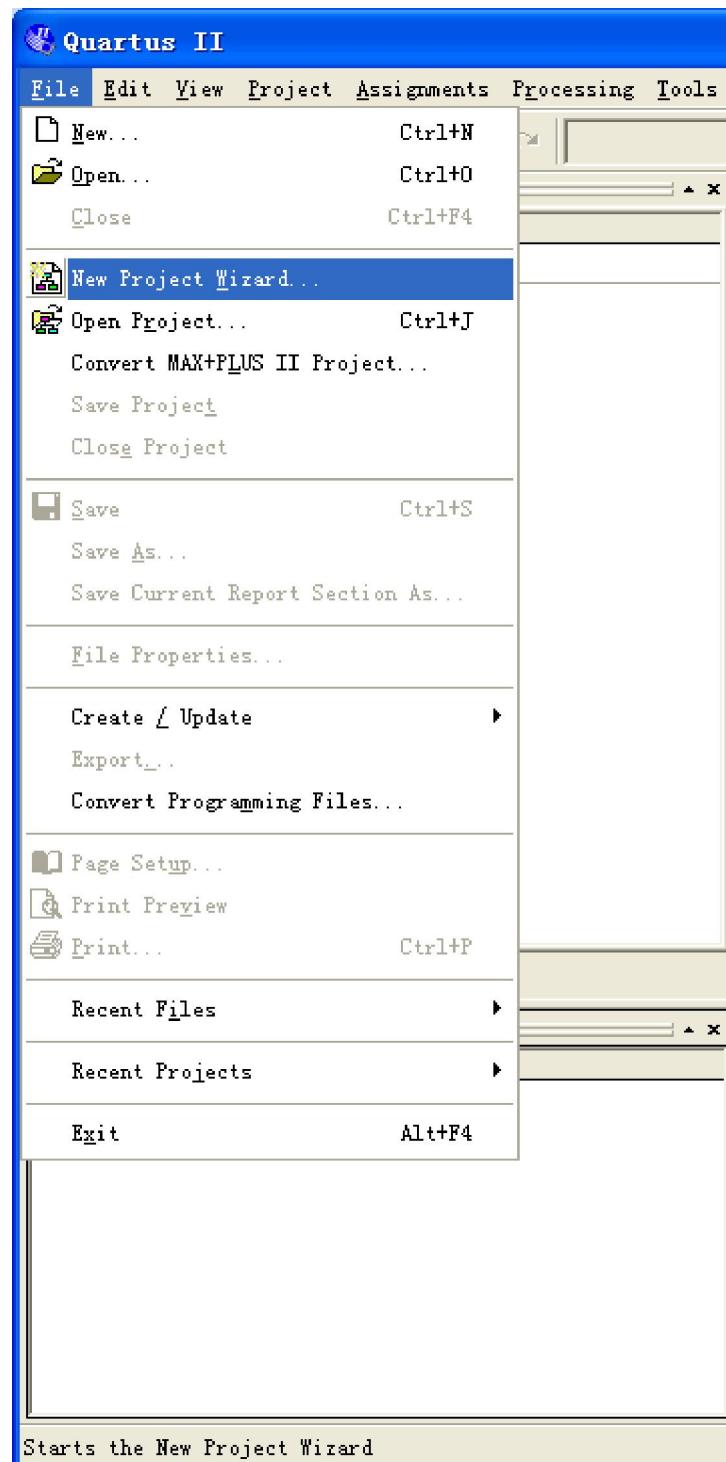


Figure 1-1 Start to Create a New Project

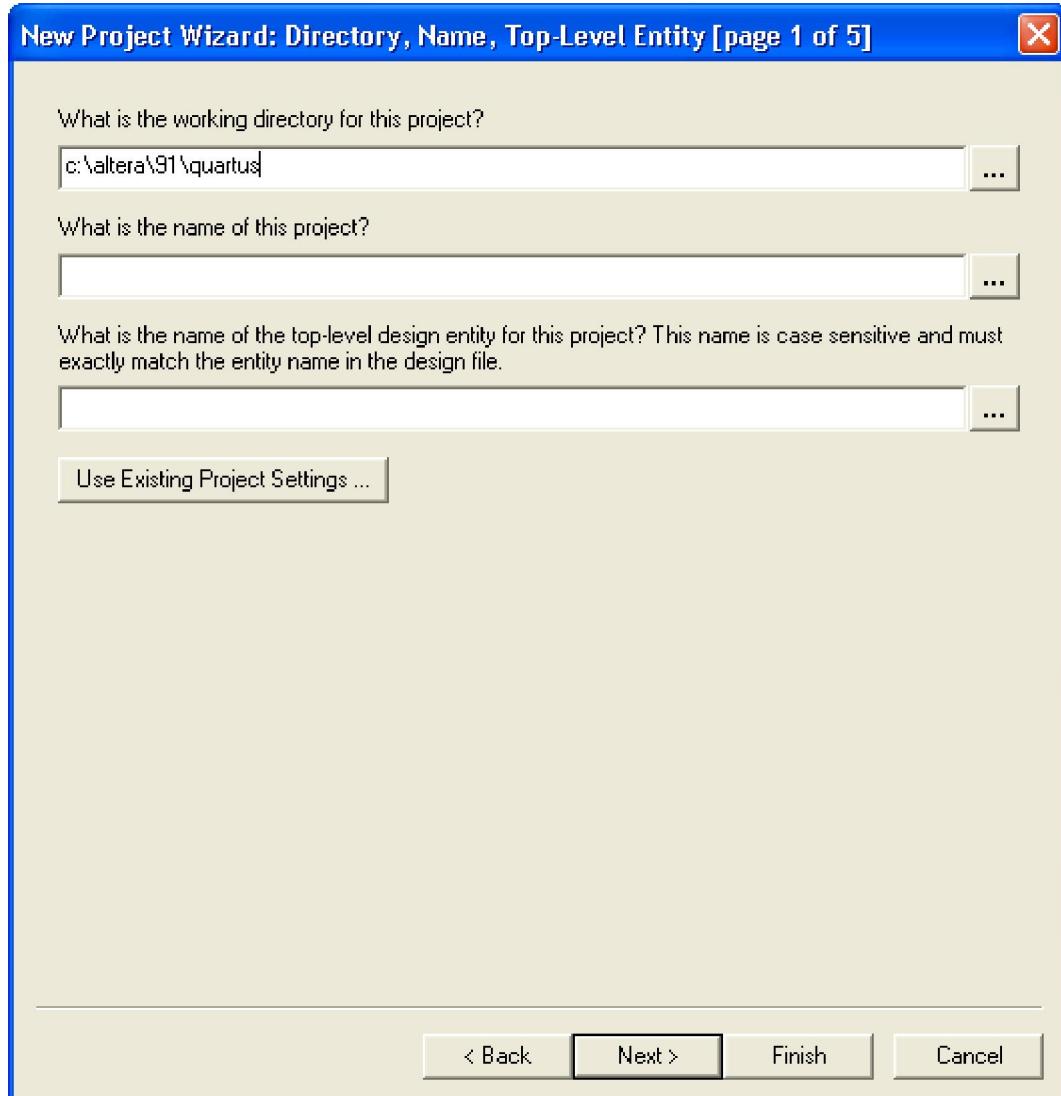


Figure 1-2 New Project Wizard

2. Choose a working directory for this project, type project name and top-level entity name as shown in Figure 1-3. Then click **Next**, you will see a window as shown in Figure 1-4.

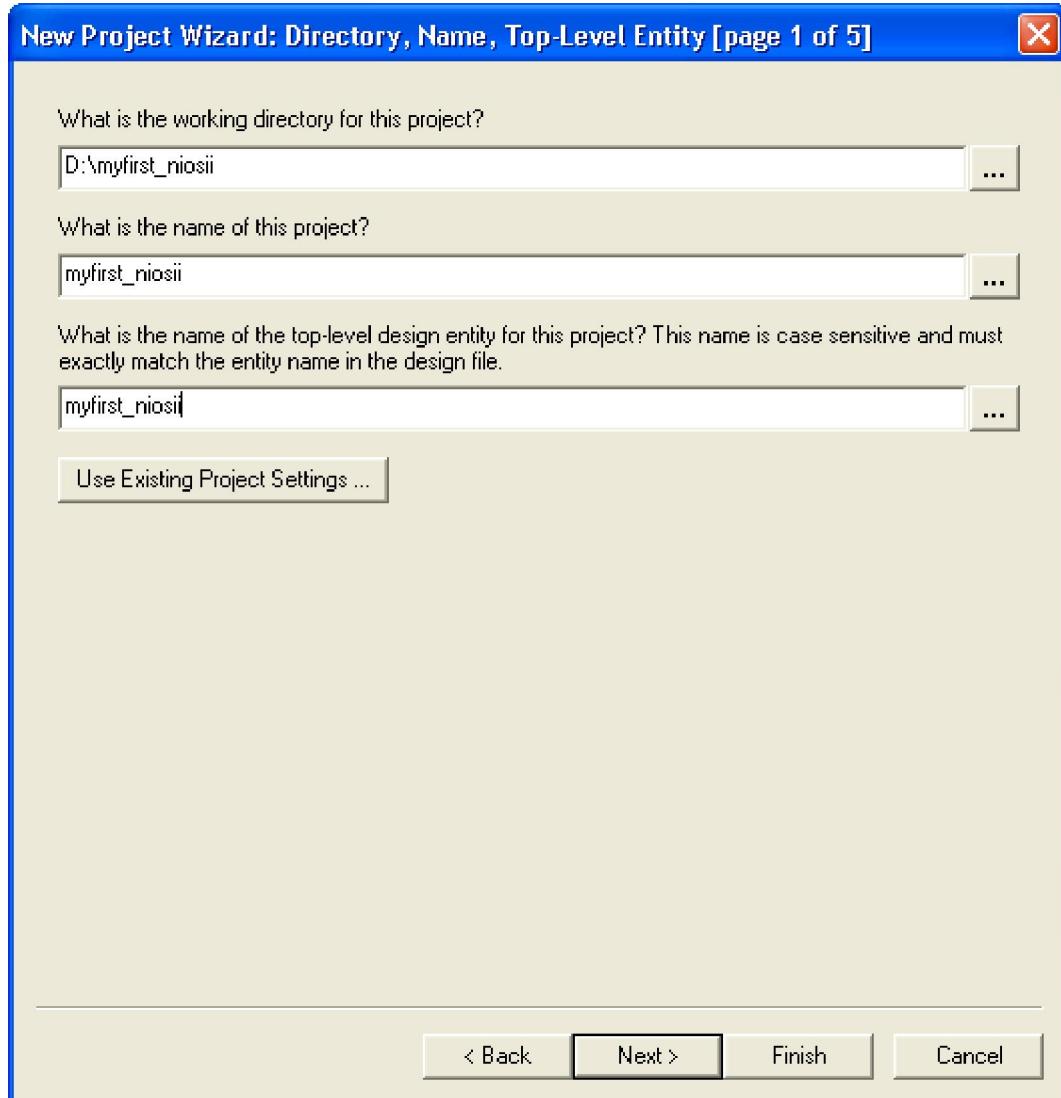


Figure 1-3 Input the working directory, the name of project, top-level design entity

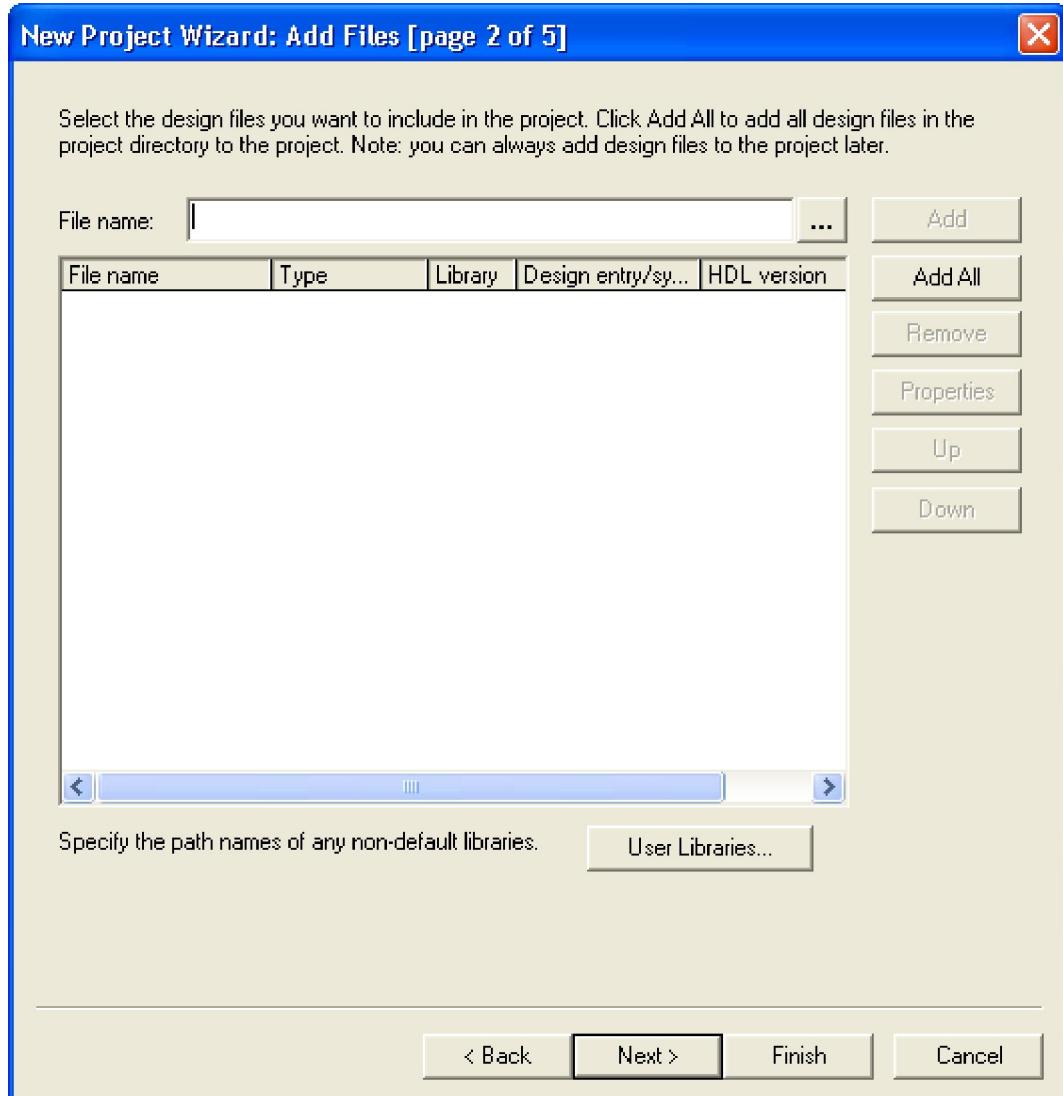


Figure 1-4 New Project Wizard: Add Files [page 2 of 5]

3. Click **Next** to next window. We choose device family and device settings. You should choose settings the same as the **Figure 1-5**. Then click **Next** to next window as shown in **Figure 1-6**.

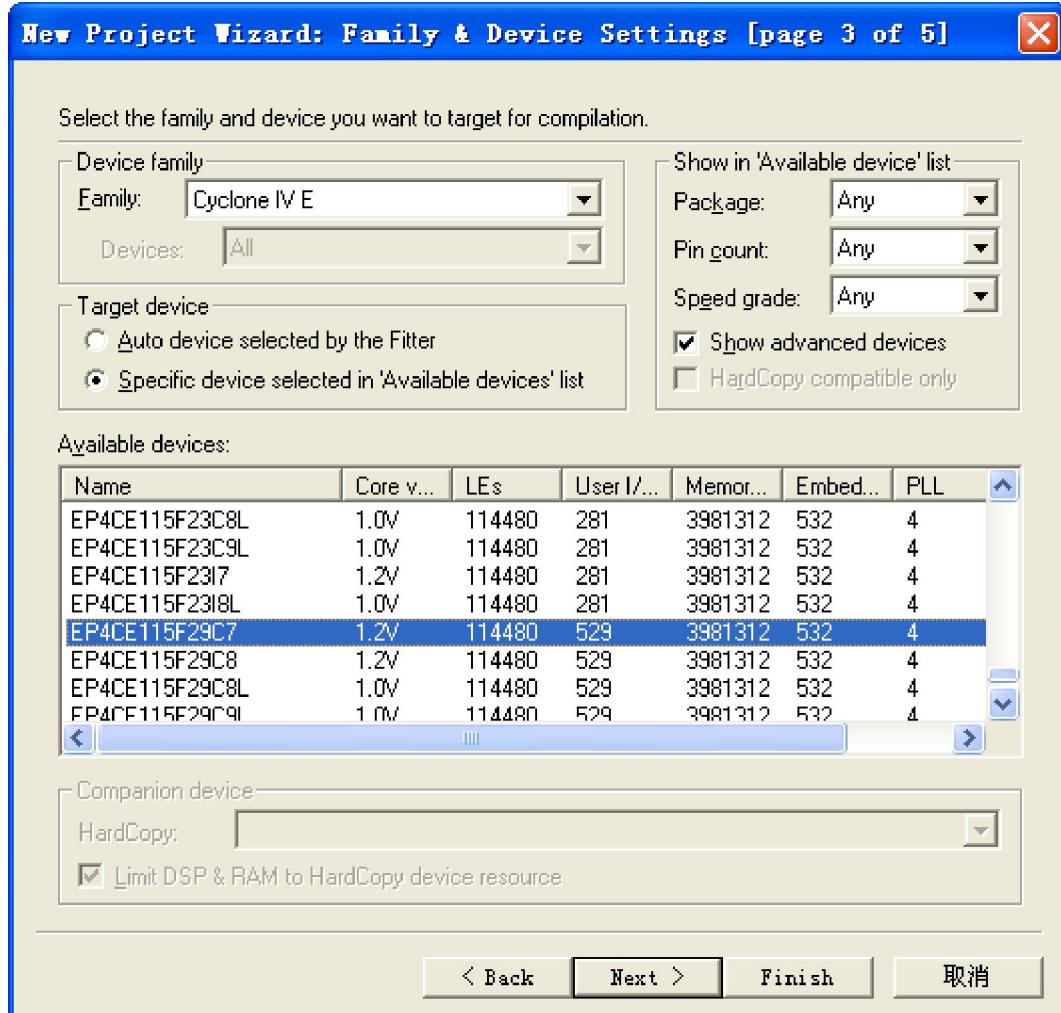


Figure 1-5 New Project Wizard: Family & Device Settings [page 3 of 5]

4. Click **Next** and will see a window as shown in Figure 1-7. Figure 1-7 is a summary about our new project. Click **Finish** to finish new project. Figure 1-8 show a new complete project.

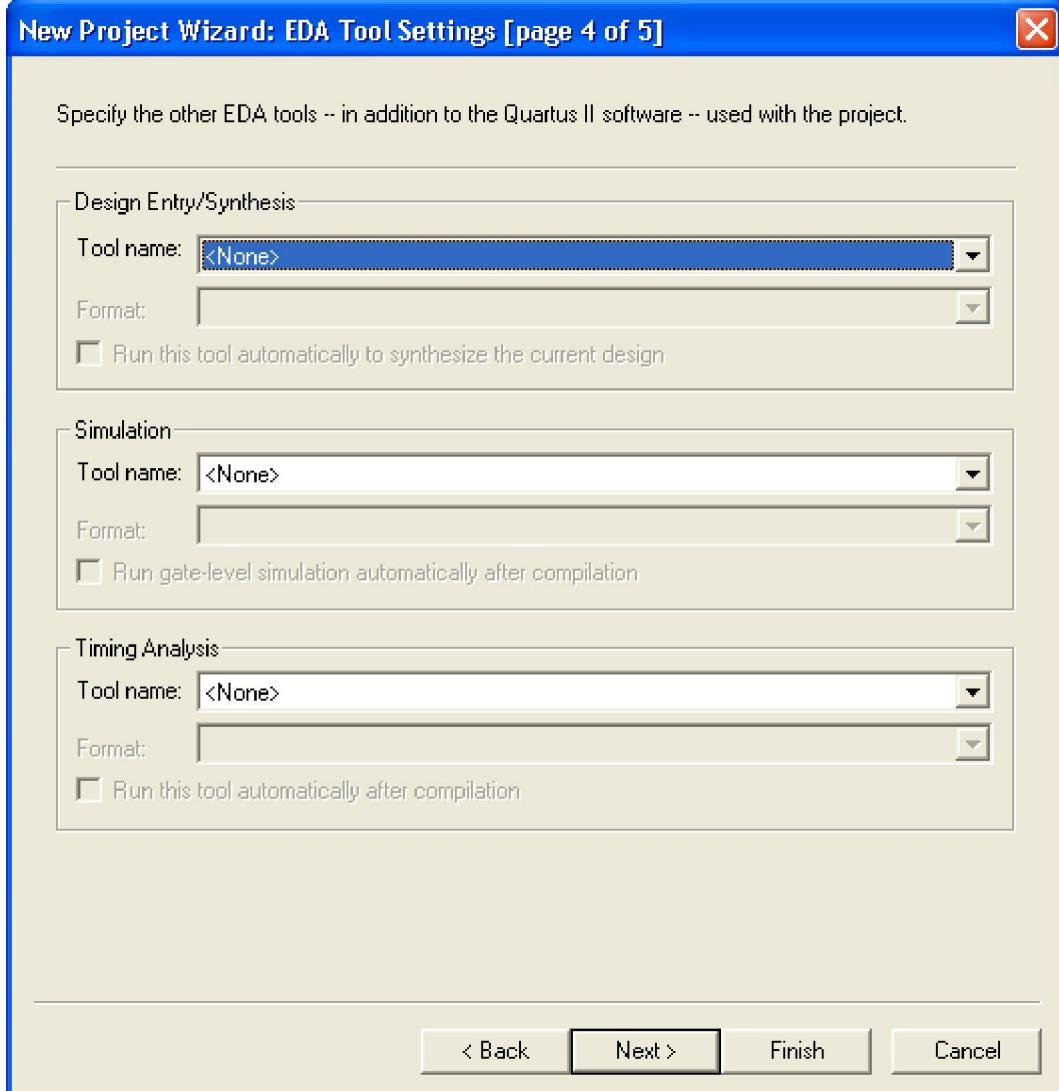


Figure 1-6 New Project Wizard: EDA Tool Settings [page 4 of 5]

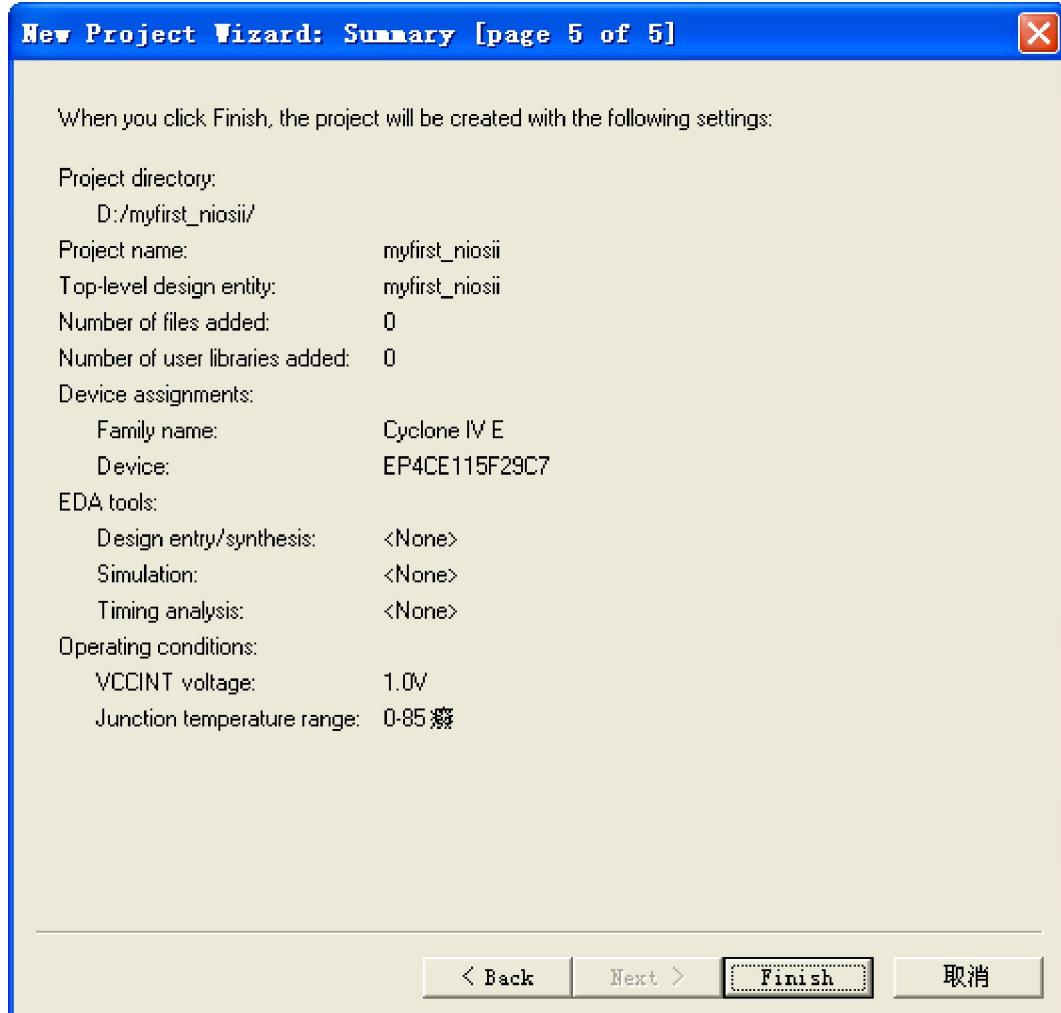


Figure 1-7 New Project Wizard: Summary [page 5 of 5]

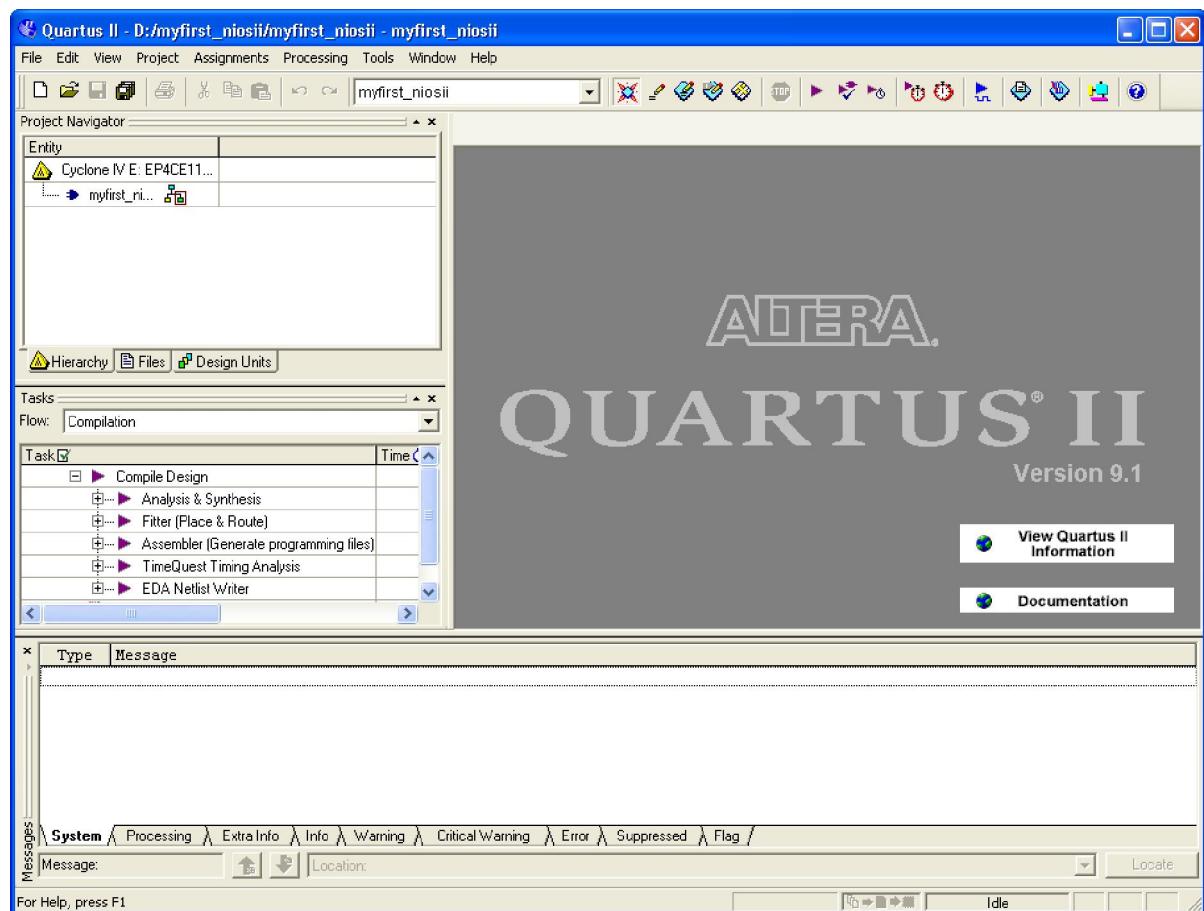


Figure 1-8 A New Complete Project

5. Choose **Tools > SOPC Builder** to open new SOPC system wizard .See **Figure 1-9** and **Figure 1-10**.

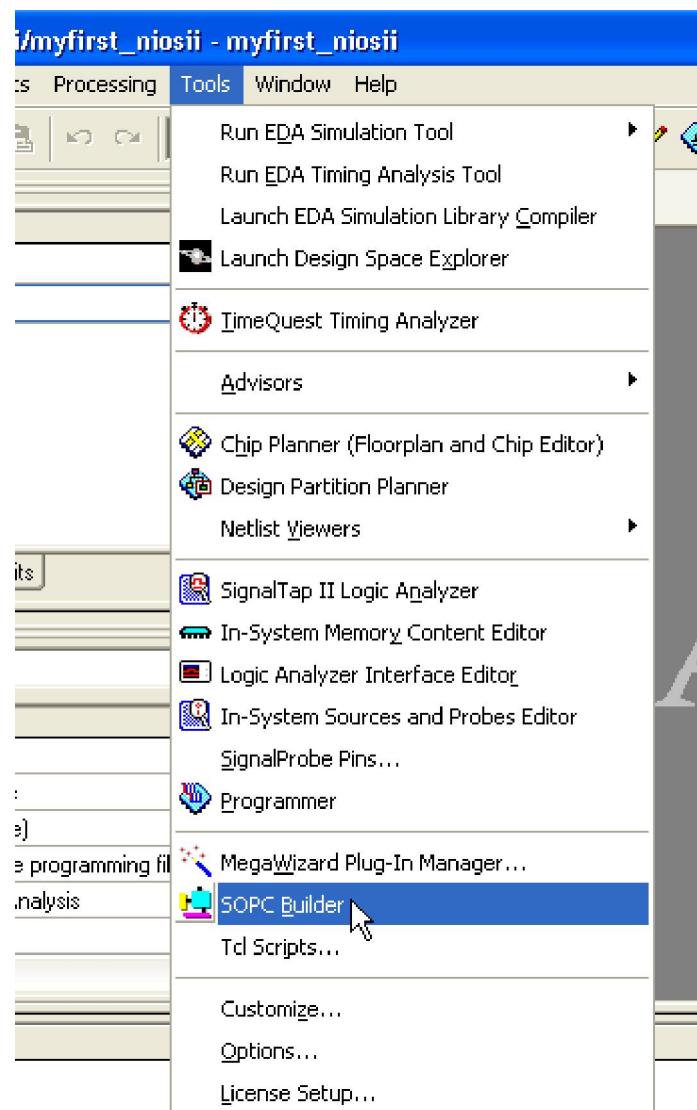


Figure 1-9 SOPC Builder Menu

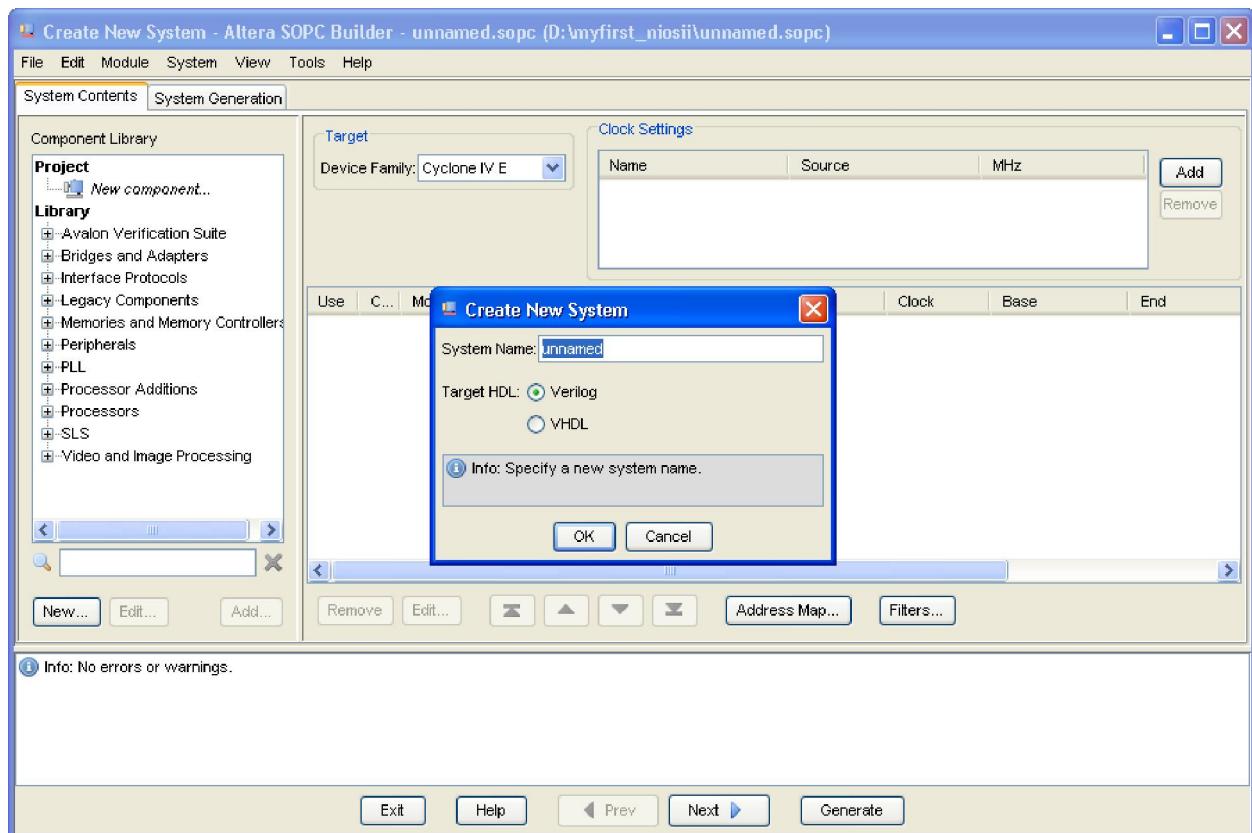


Figure 1-10 Create New SOPC System [0]

6. Rename **System Name** as shown in Figure 1-11. Click **OK** and you will see a window as shown in Figure 1-12.



Figure 1-11 Create New System [1]

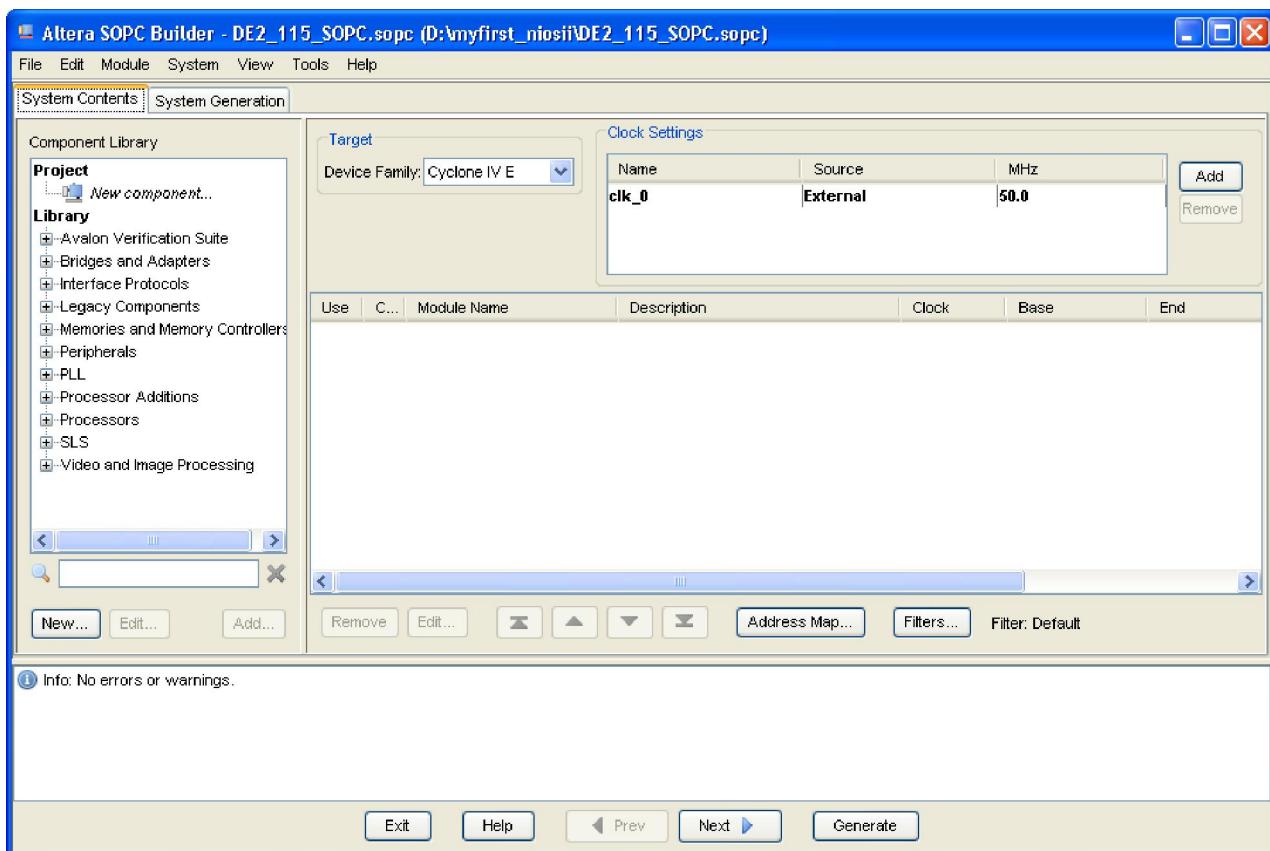


Figure 1-12 Create New System [2]

7. Click the **Name** of the **Clock Settings** table, rename **clk_0** to **clk_50**. Press Enter to complete the update. See Figure 1-13.

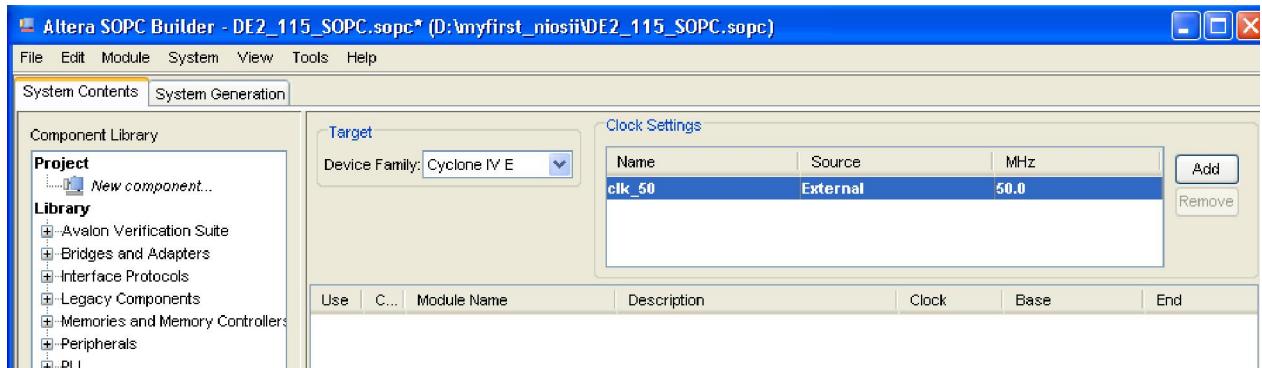


Figure 1-13 Rename Clock Name

8. Choose **Library > Processors > Nios II Processor** to open wizard of adding cpu component. See **Figure 1-14** and **Figure 1-15**.

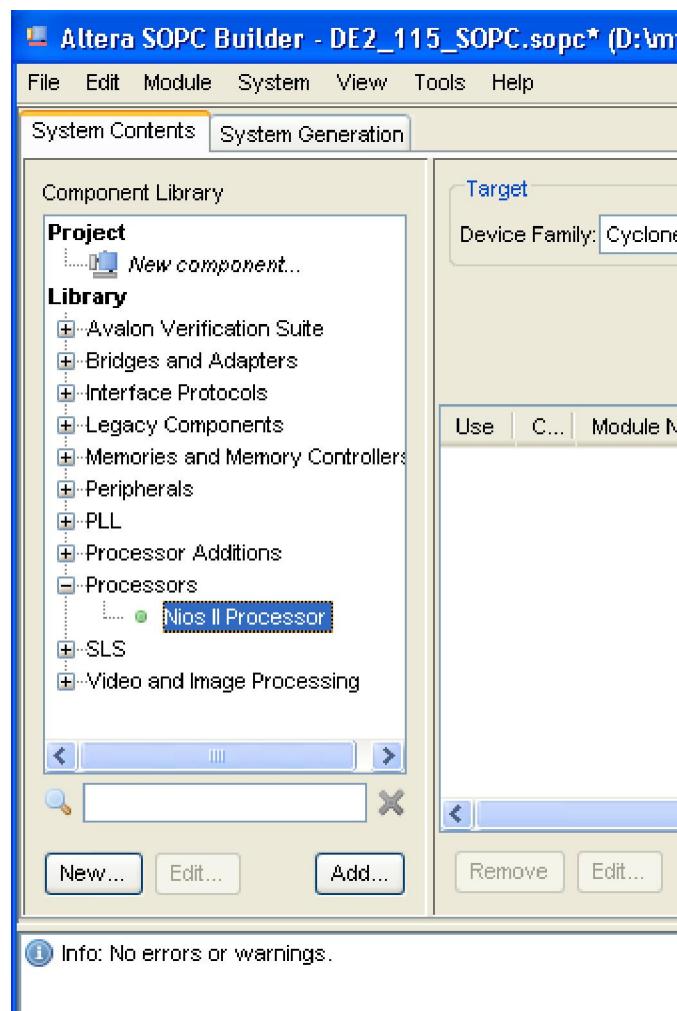


Figure 1-14 Add Nios II Processor

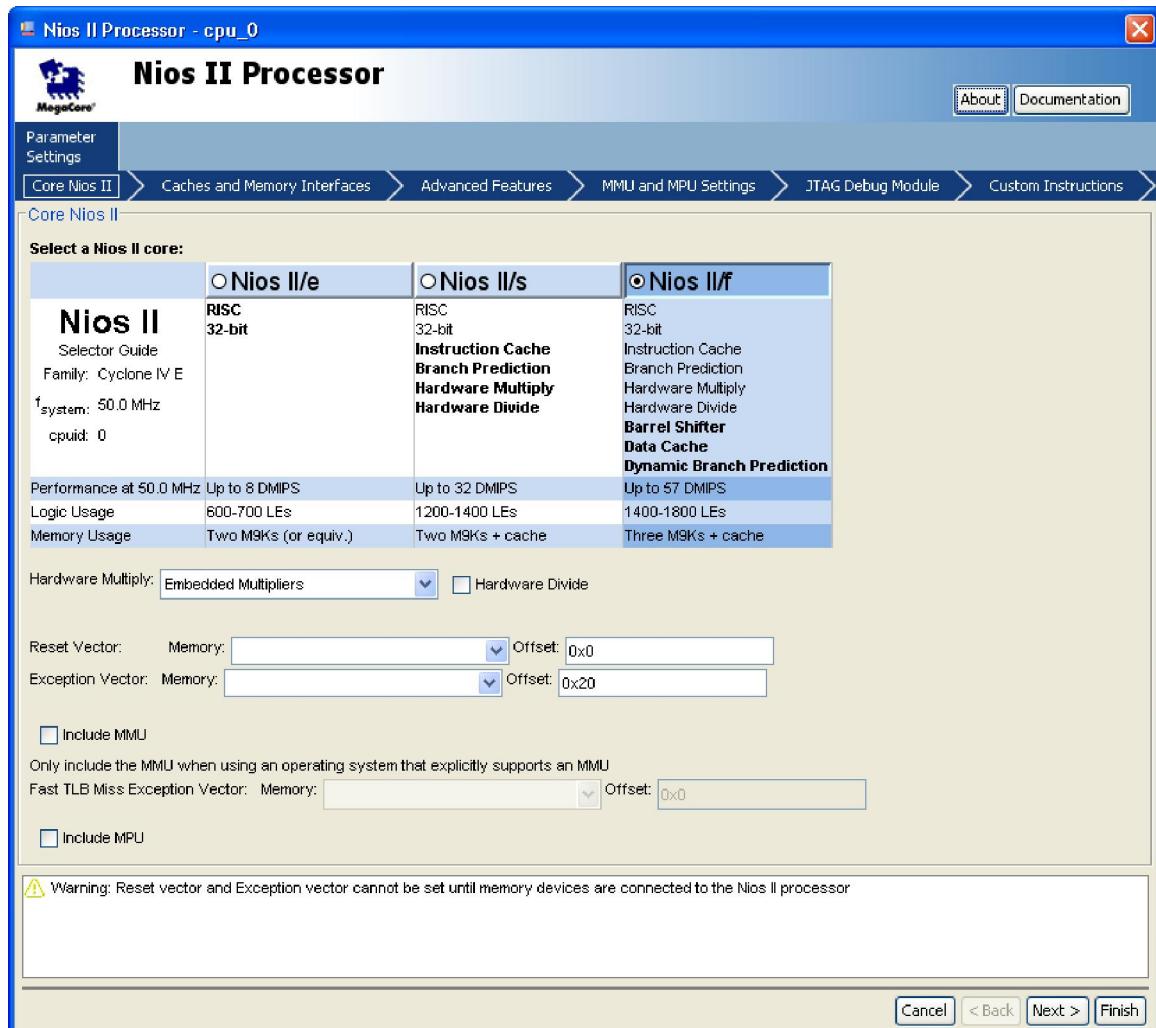


Figure 1-15 Nios II Processor

9. Click **Finish** to return to main window as shown in Figure 1-16.

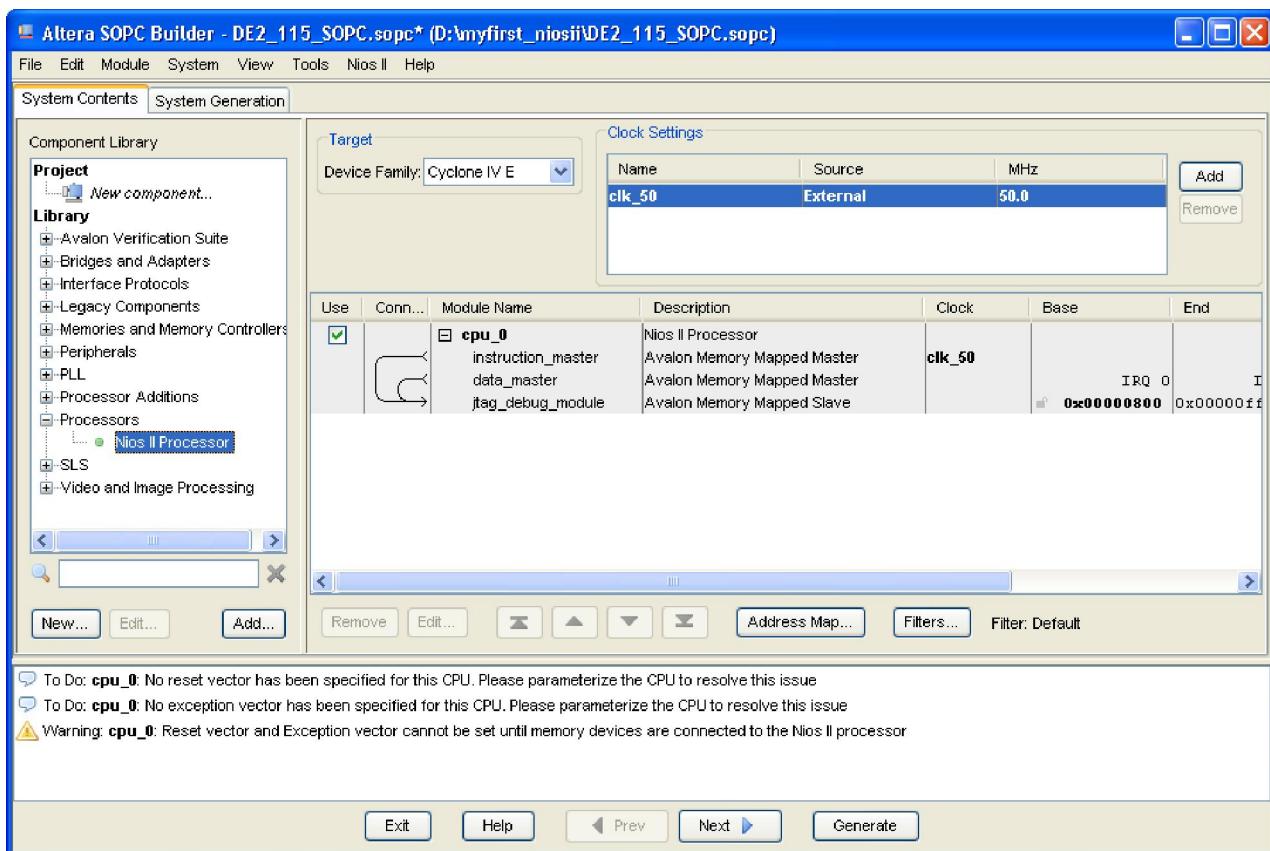


Figure 1-16 Add Nios II CPU completely

10. Choose **cpu_0** and right-click then choose **rename**, after this, you can update **cpu_0** to **cpu**.

See Figure 1-17 and Figure 1-18.

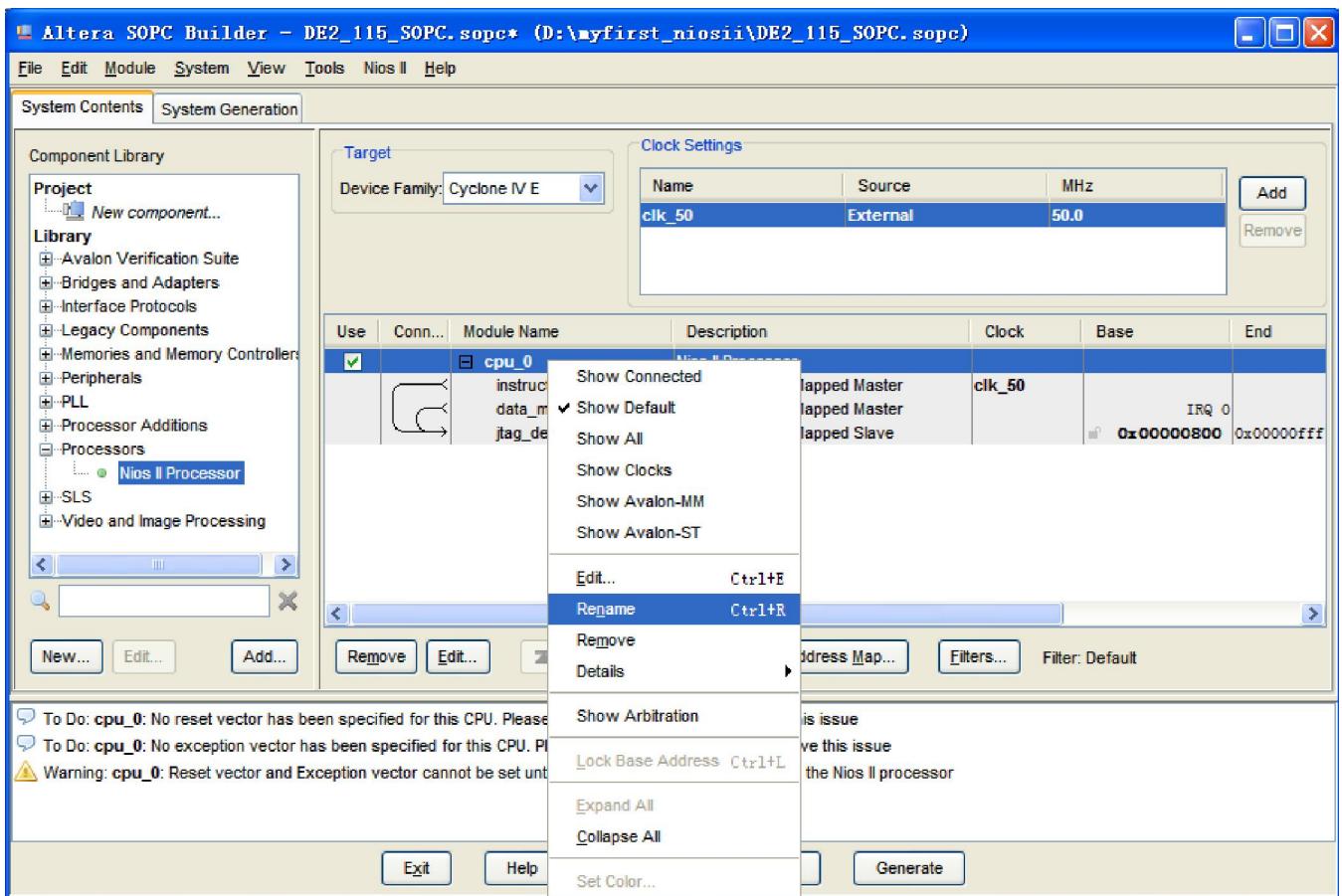


Figure 1-17 Rename CPU name (1)

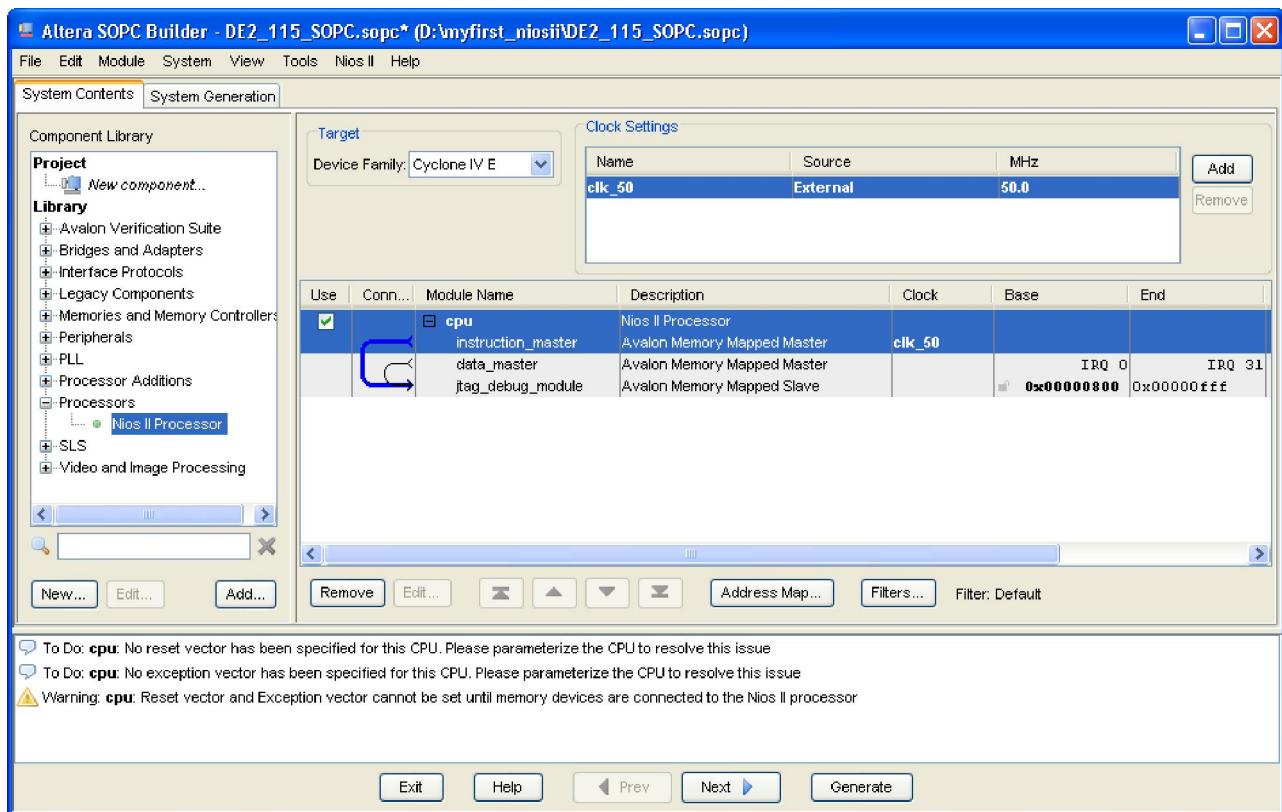


Figure 1-18 Rename CPU Name (2)

11. Choose **Library > Interface Protocols > Serial > JTAG UART** to open wizard of adding JTAG UART. See Figure 1-19 and Figure 1-20.

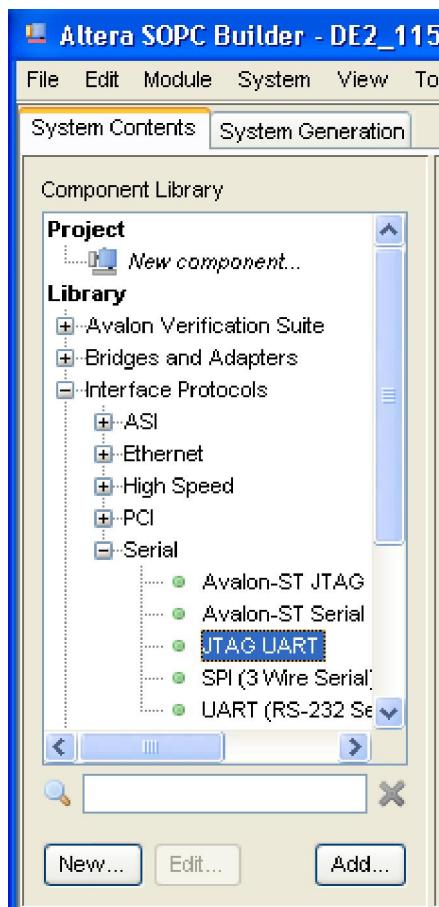


Figure 1-19 Add JTAG UART (1)



Figure 1-20 JTAG UART

12. Click **Finish** to close the wizard and return to the window as shown in Figure 1-21.

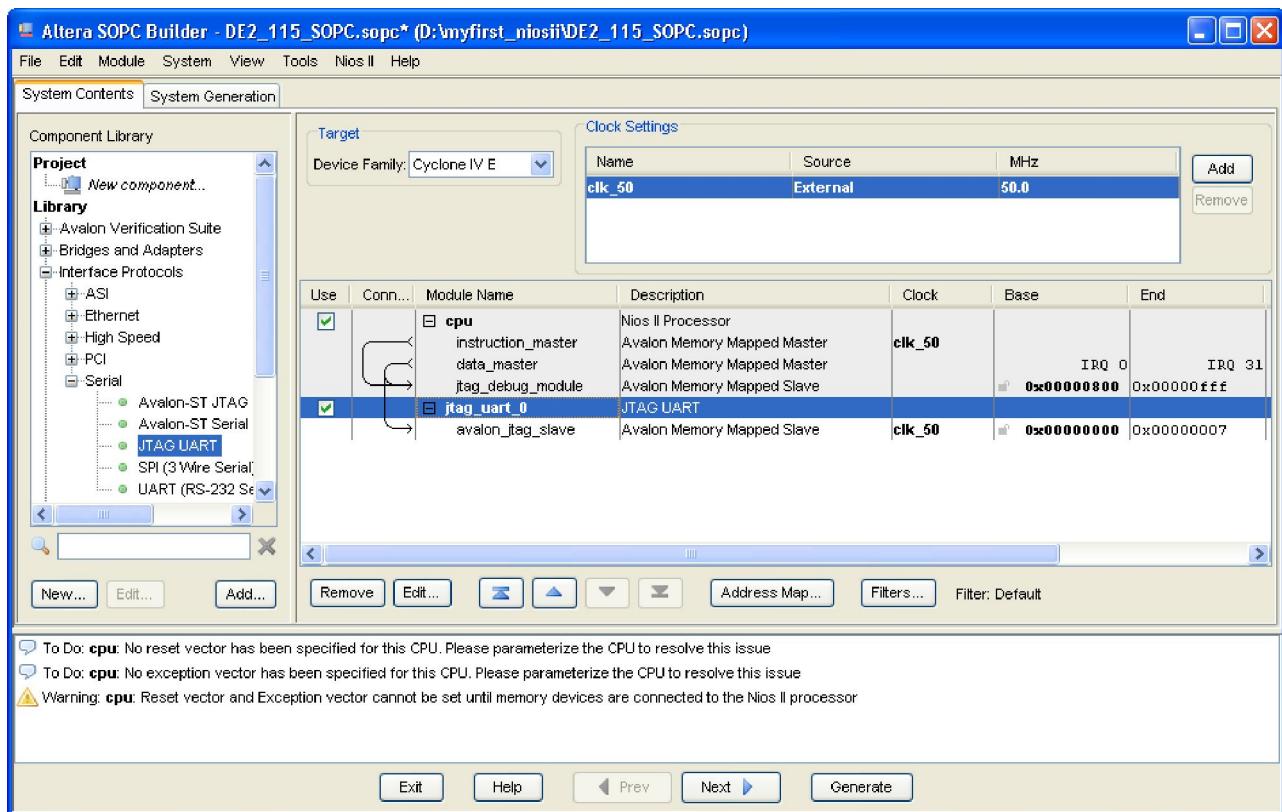


Figure 1-21 JTAG UART

13. Choose **jtag_uart_0** and rename it to **jtag_uart** as shown in Figure 1-22.

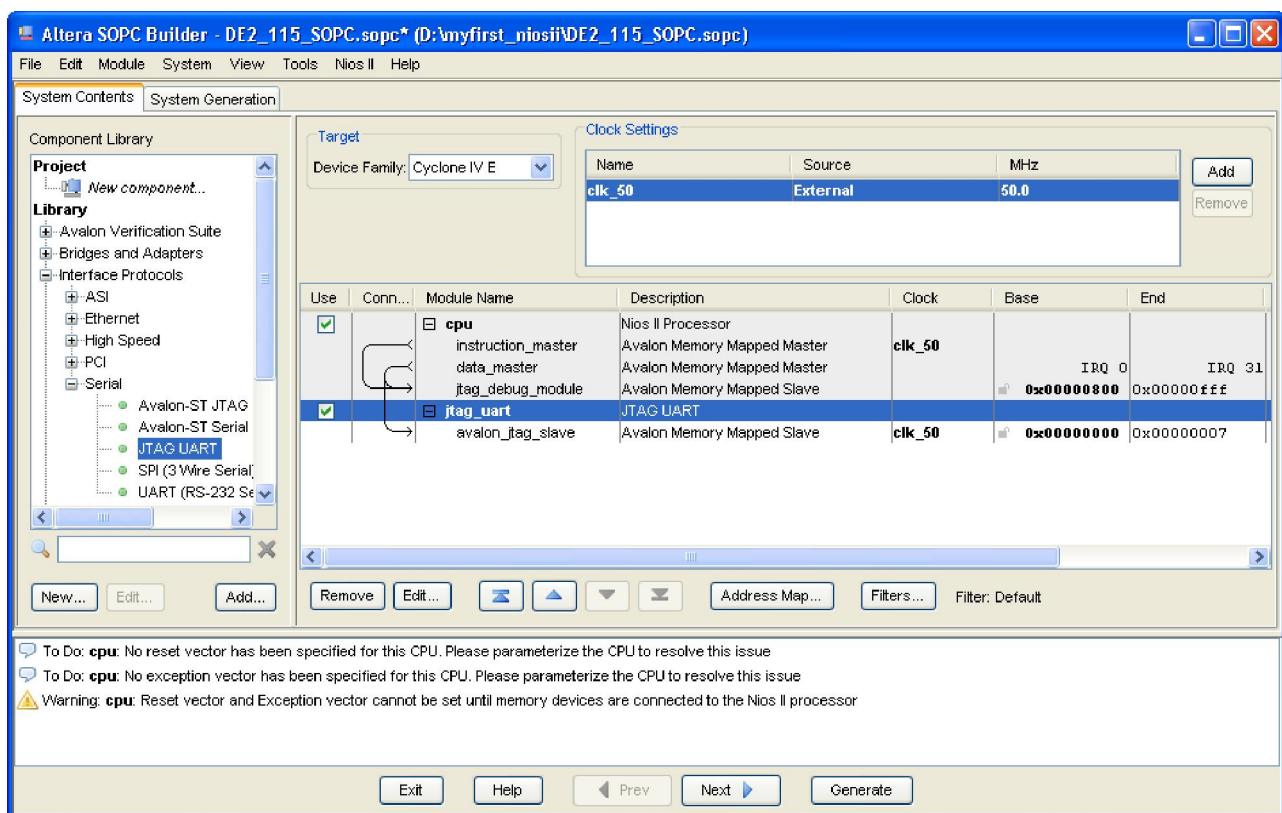


Figure 1-22 Rename JTAG UART

15. Choose **Library > Memories and Memory Controllers > On-Chip > On-Chip Memory (RAM or ROM)** to open wizard of adding On-Chip memory. See Figure 1-23 and Figure 1-24.

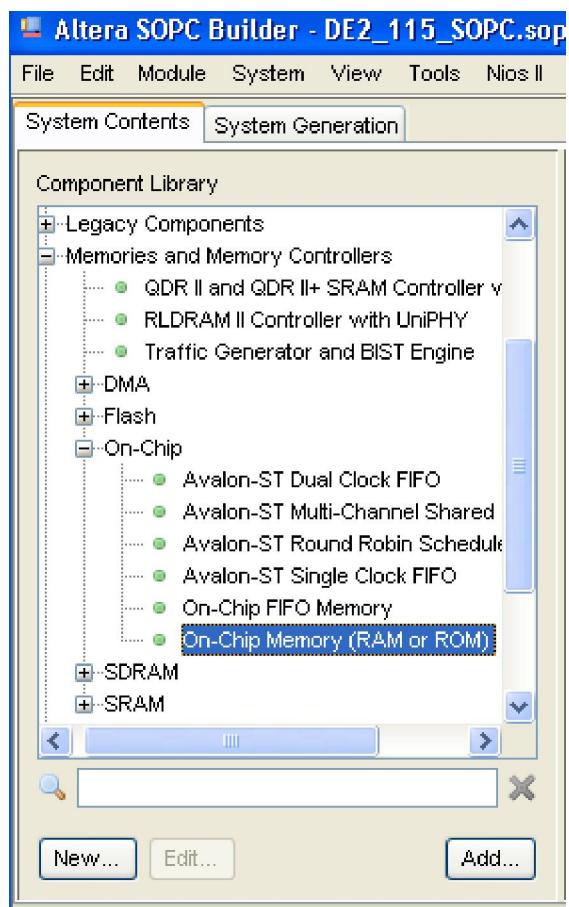


Figure 1-23 Add On-Chip Memory

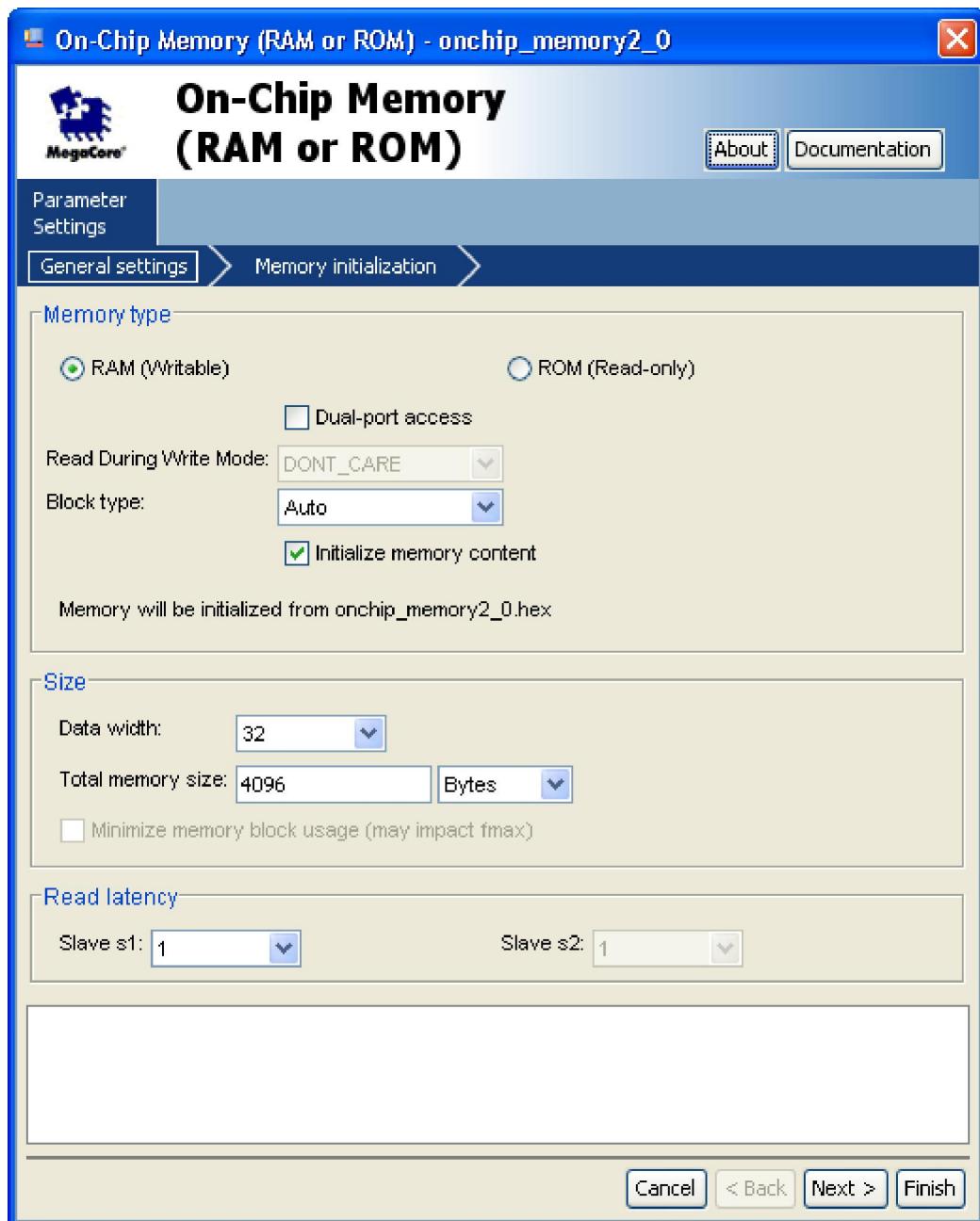


Figure 1-24 On-Chip Memory Box

16. Modify **Total memory size** to **204800** as shown in Figure 1-25. Click **Finish** to return to the window as in Figure 1-26.

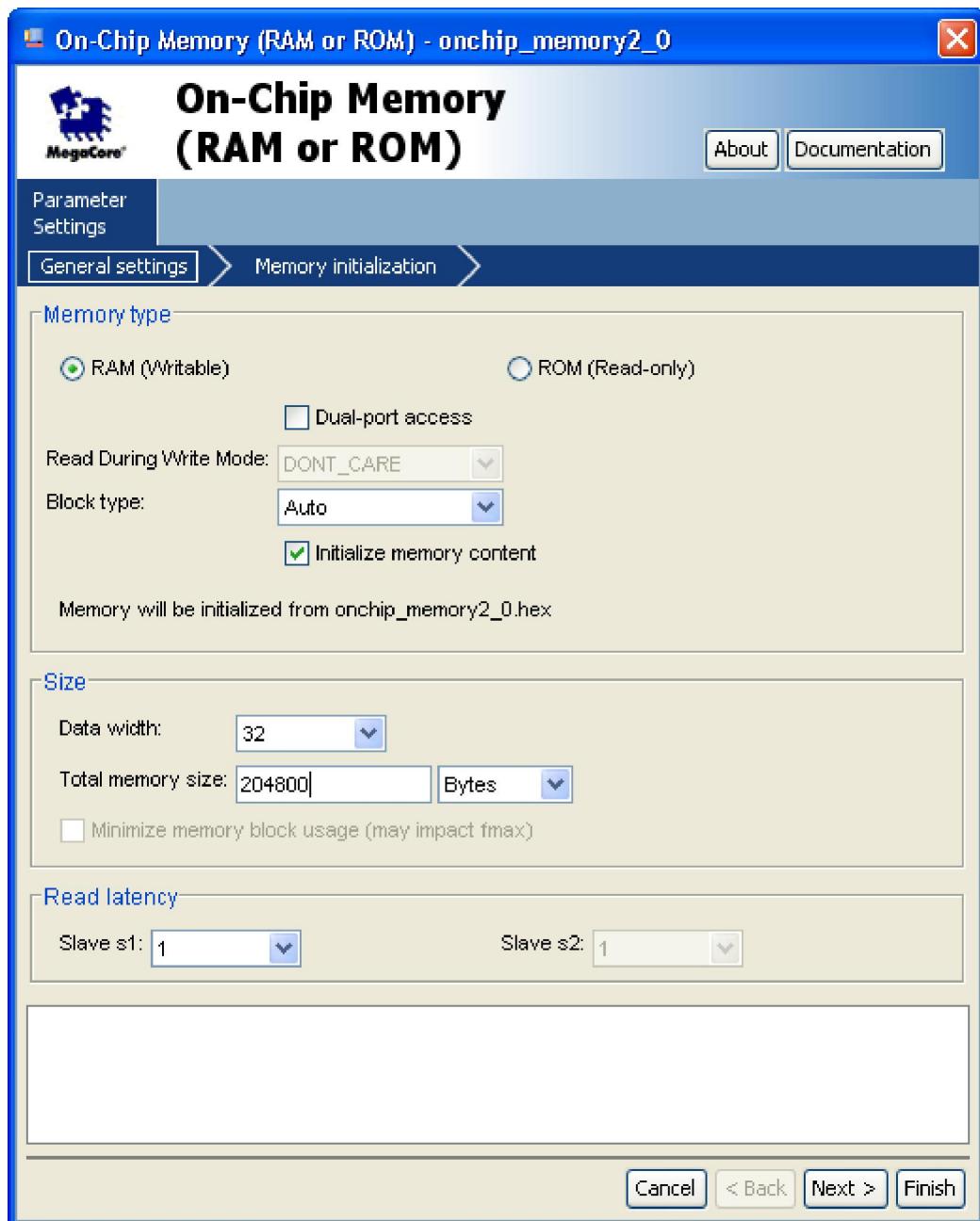


Figure 1-25 Update Total memory size

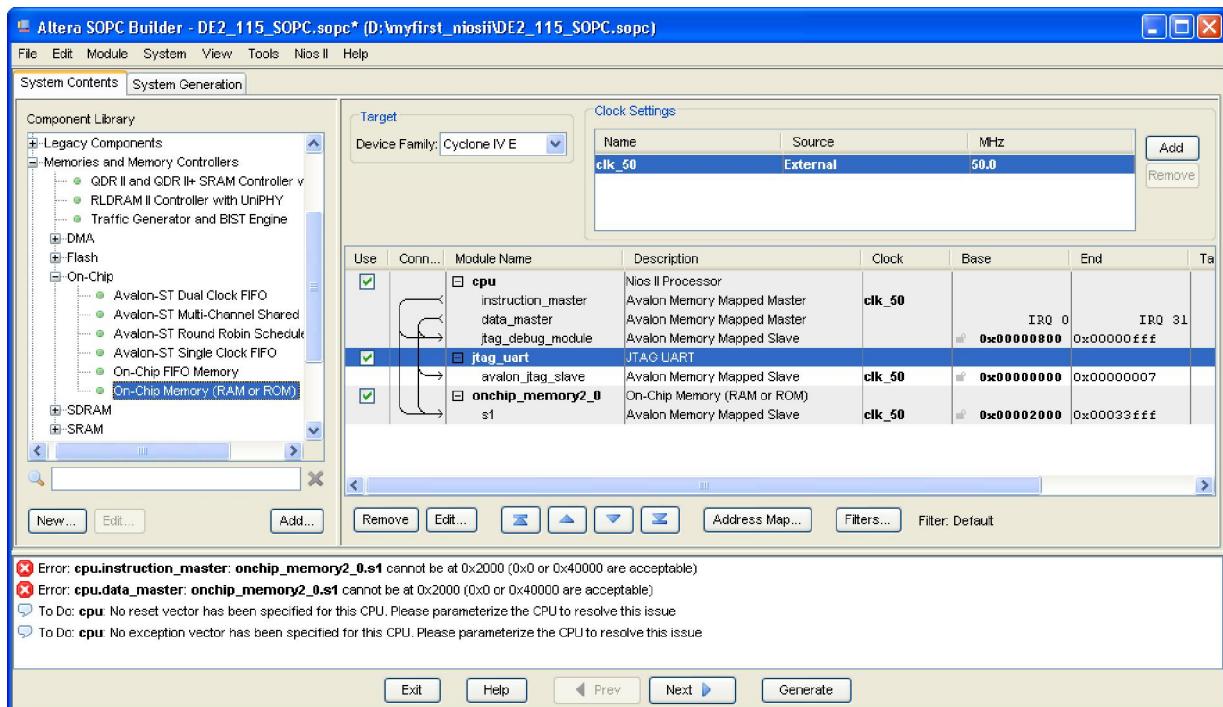


Figure 1-26 Add On-Chip memory Completely

17. Rename **onchip_memory2_0** to **onchip_memory2** as shown in Figure 1-27.

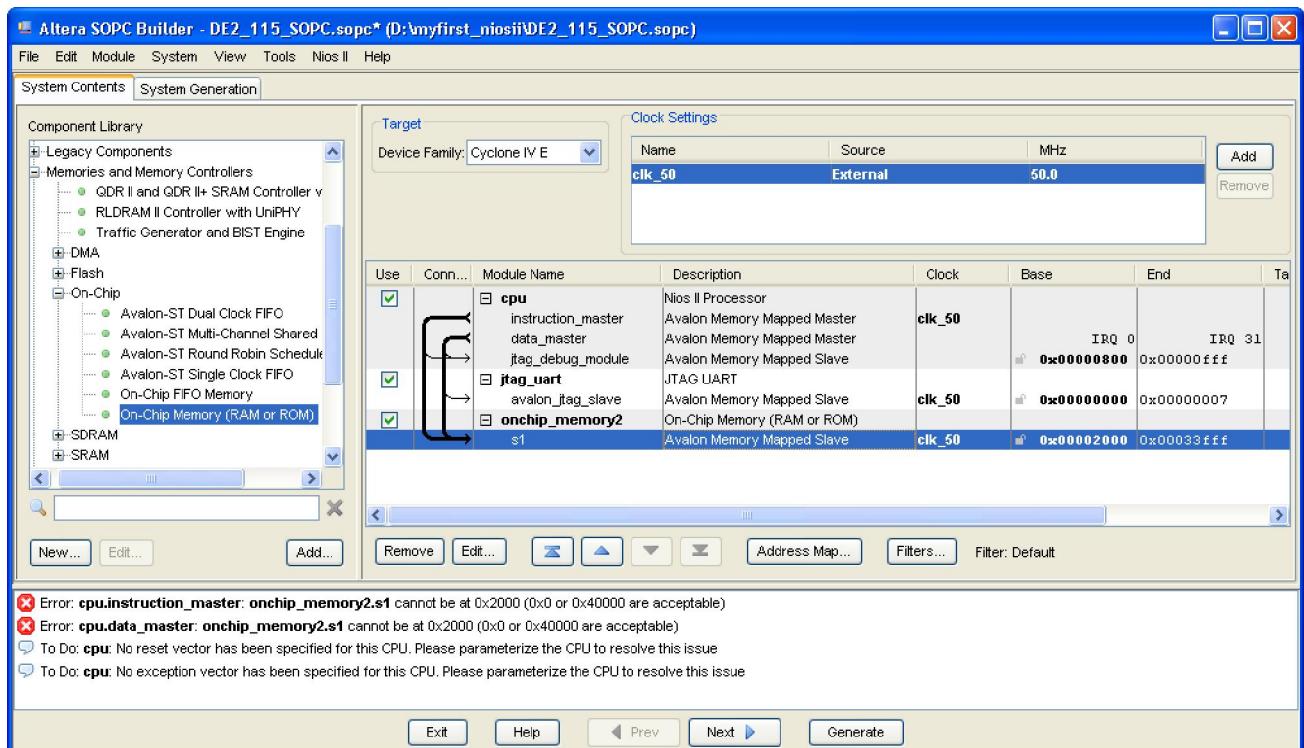


Figure 1-27 Rename On-Chip memory

18. Click **cpu** in the component list on the right part to edit the component. Update **Reset vector** and **Exception Vector** as shown in **Figure 1-28**. Then click **Finish** to return to the window as shown **Figure 1-29**.

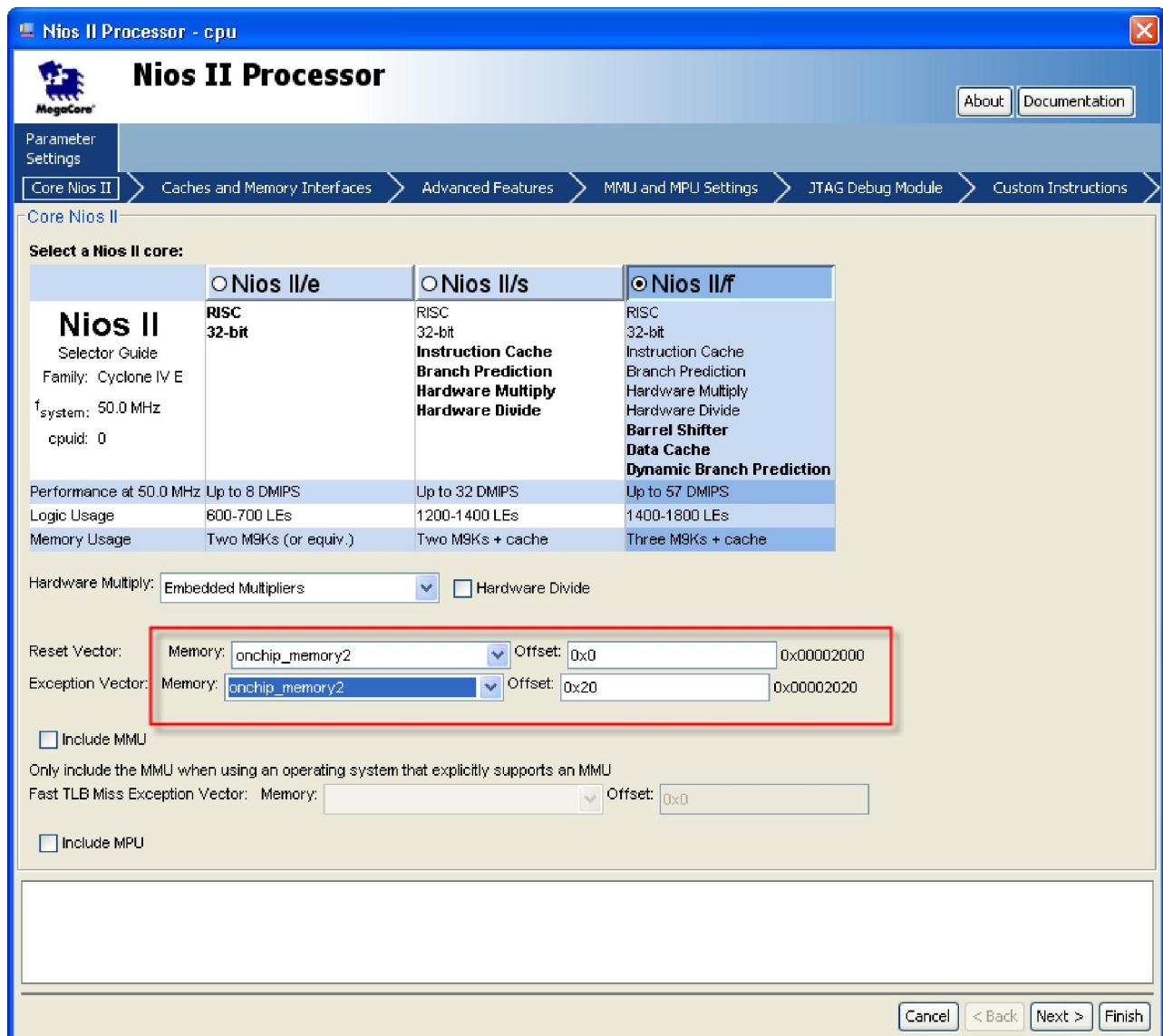


Figure 1-28 Update CPU settings

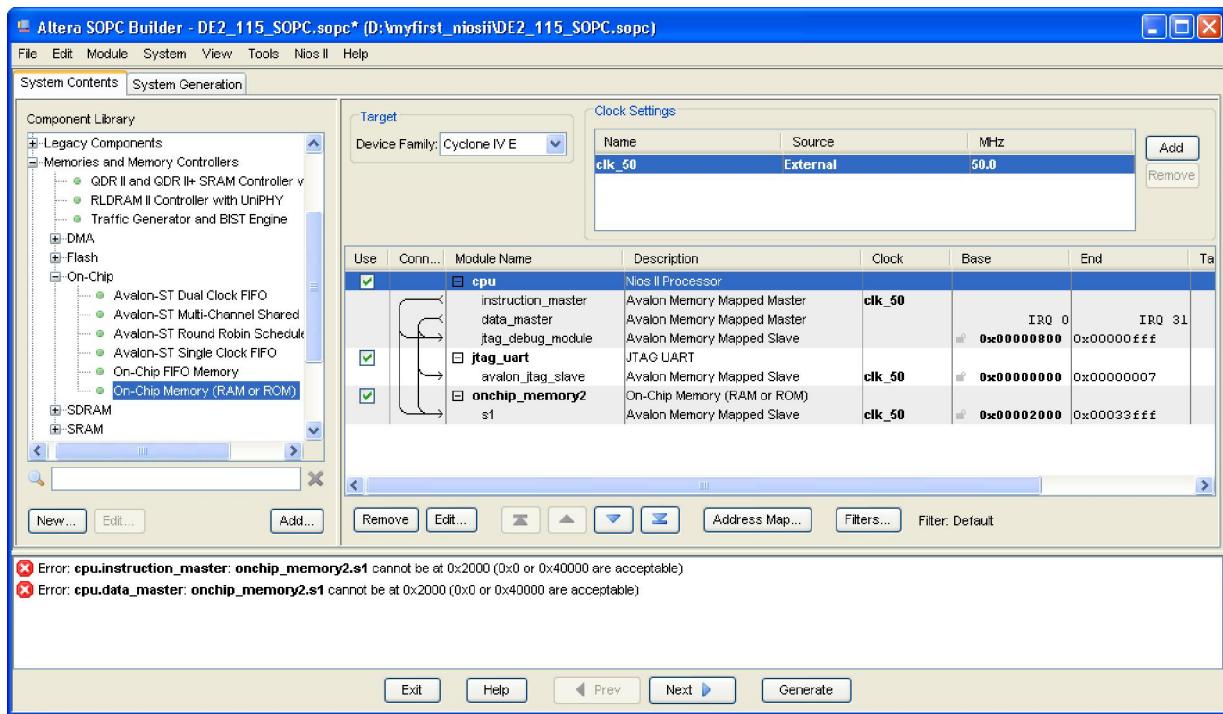


Figure 1-29 Update CPU settings Completely

19. Choose **Library > Peripherals > Microcontroller Peripherals >PIO (Parallel I/O)** to open wizard of adding PIO. See **Figure 1-30** and **Figure 1-31**.

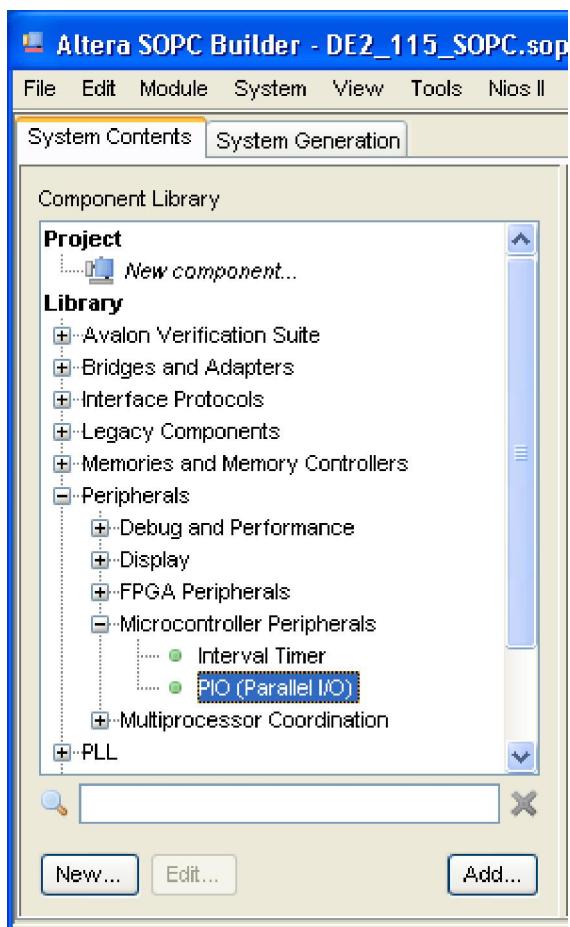


Figure 1-30 Add PIO

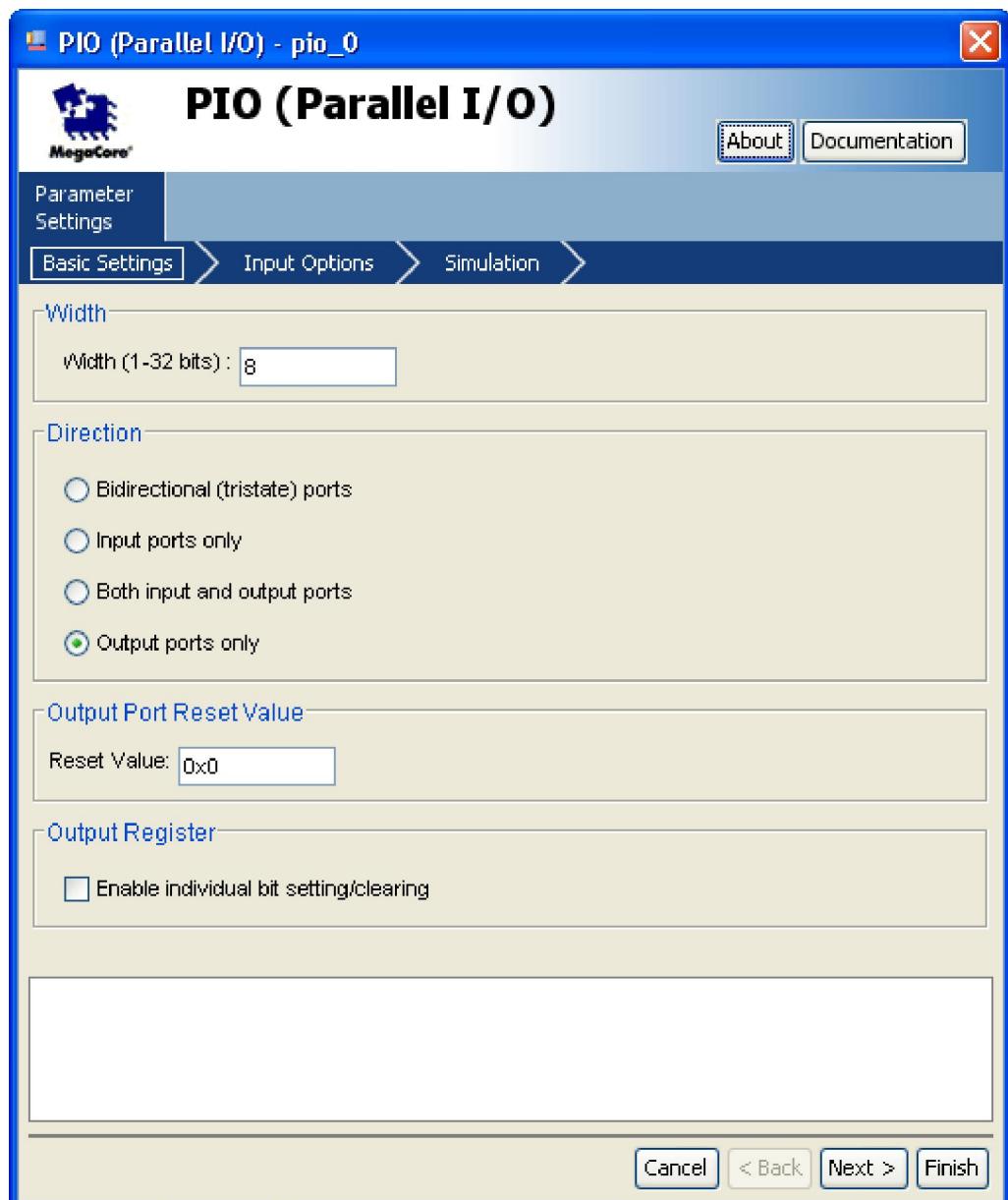


Figure 1-31 Add PIO

20. Click **Finish** to close PIO box and return to the window as shown in **Figure 1-32**.

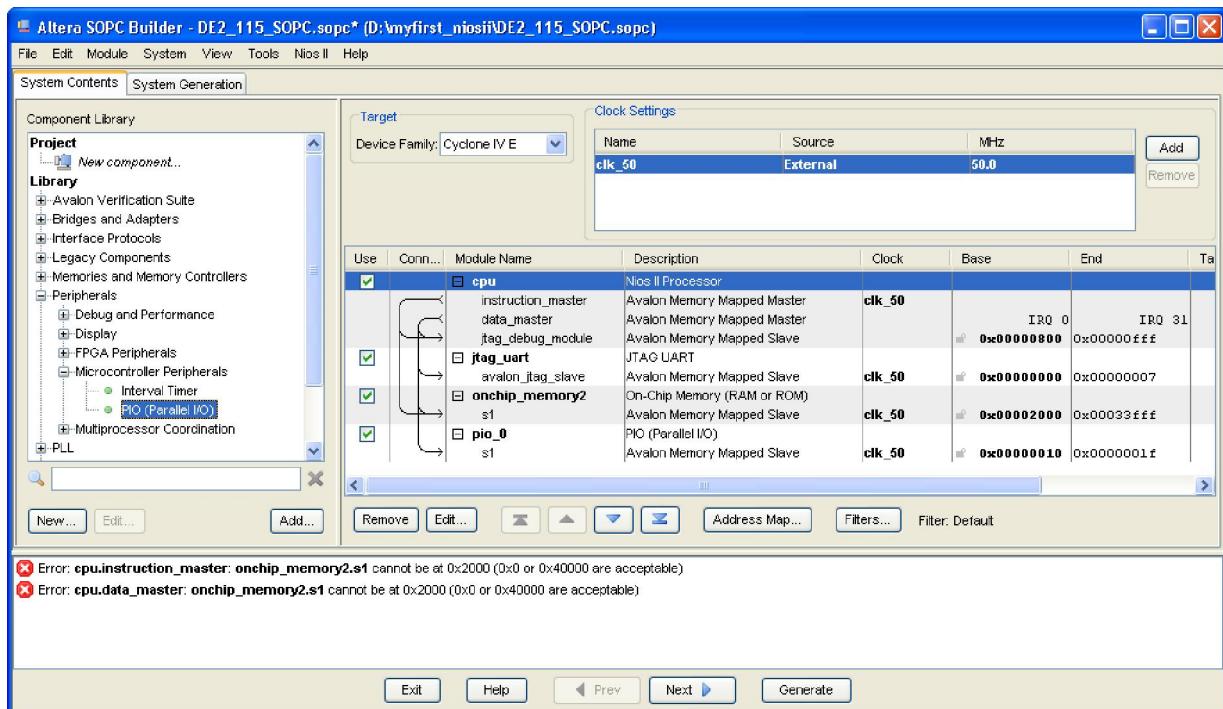


Figure 1-32 PIO

21. Rename **pio_0** to **pio_led** as shown in **Figure 1-33**.

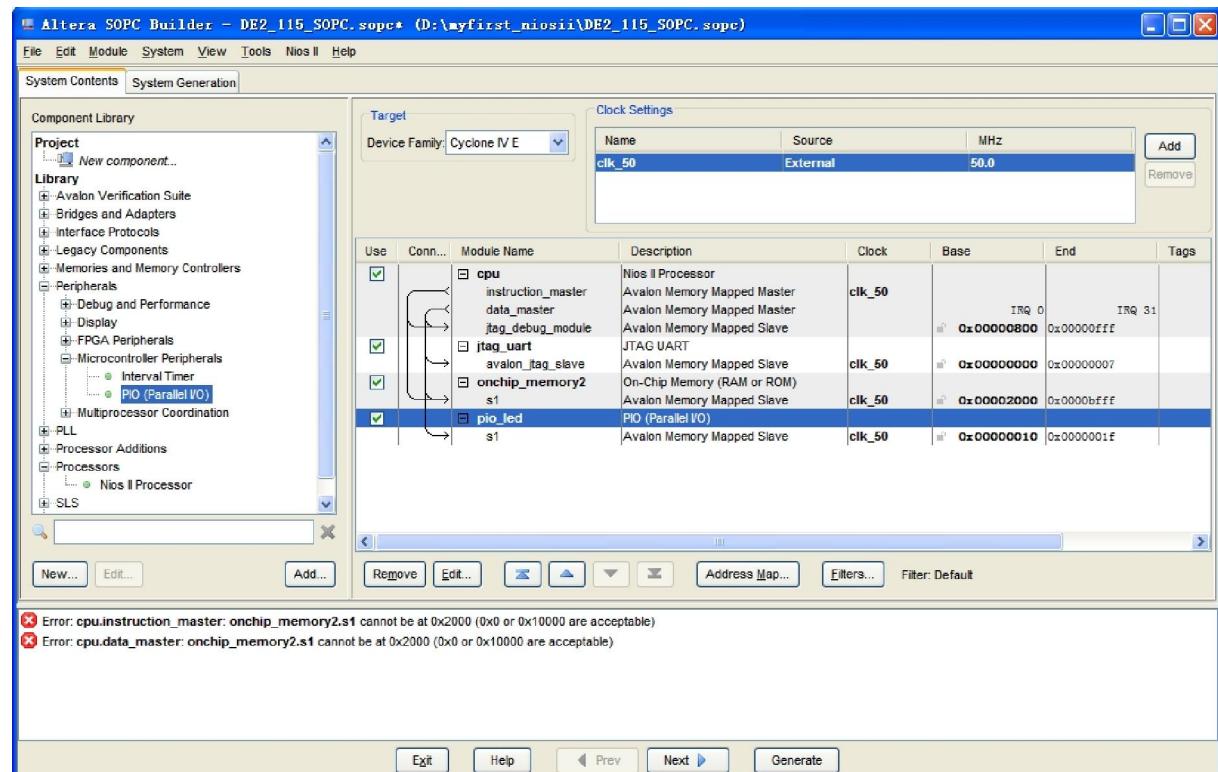


Figure 1-33 Rename PIO

22. Choose **System > Auto-Assign Base Addresses** as shown in Figure 1-34. After that, you will find that there is no error in the message window as shown in Figure 1-35.

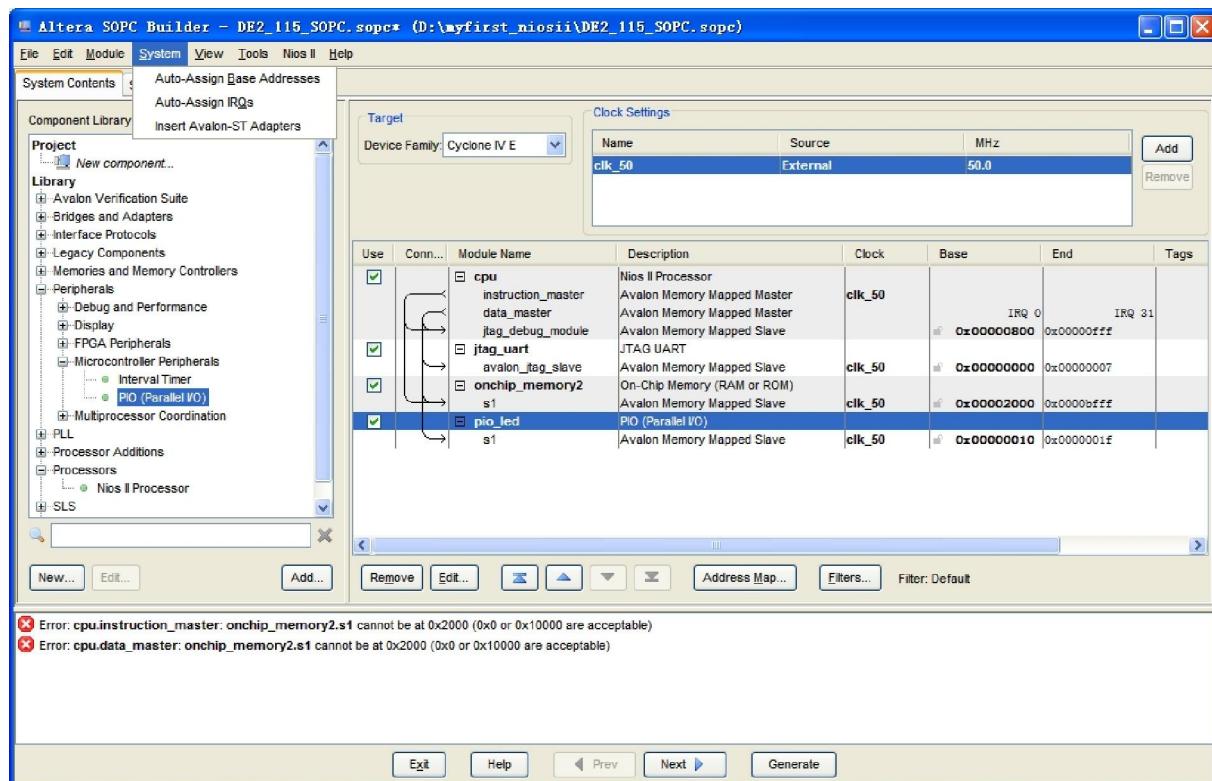


Figure 1-34 Auto-Assign Base Addresses

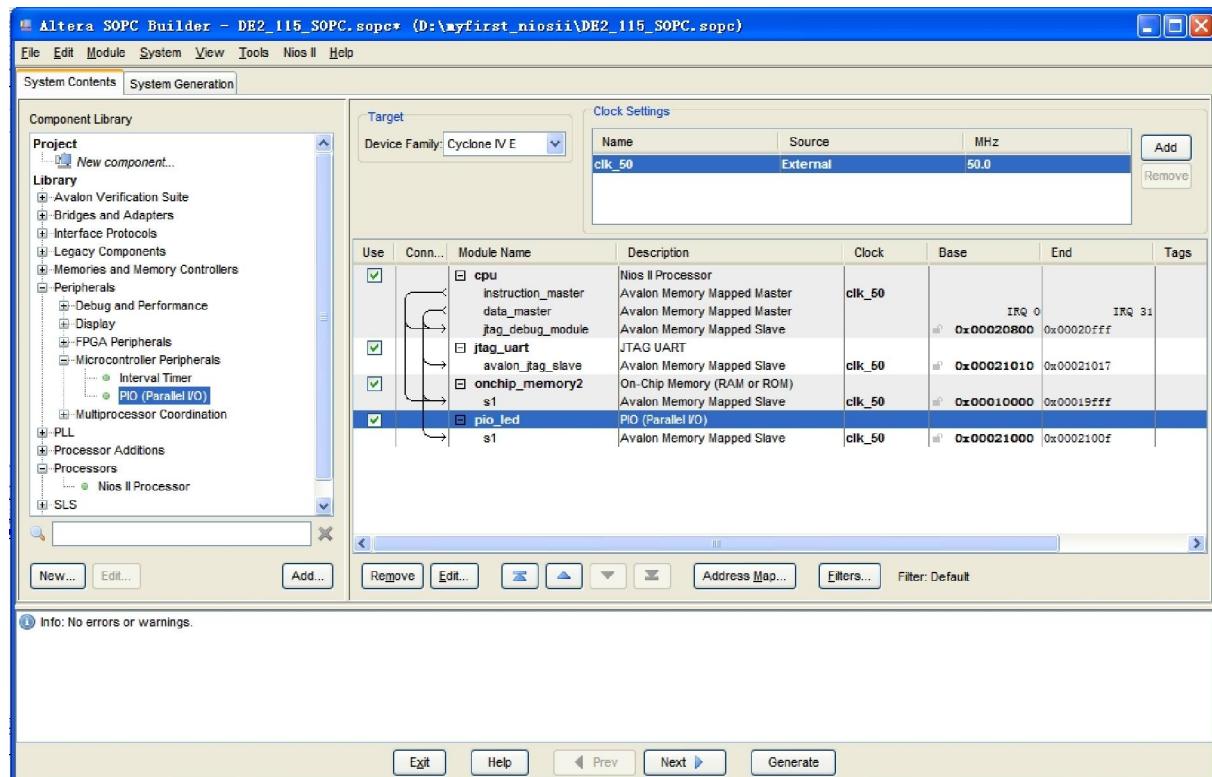


Figure 1-35 No errors or warnings

23. Click **Generate** and then pop a window as shown in Figure 1-36. Click **Save** and the generation start. Figure 1-37 shows the generate process. If there is no error in the generation, the window will show successful as shown in Figure 1-38.

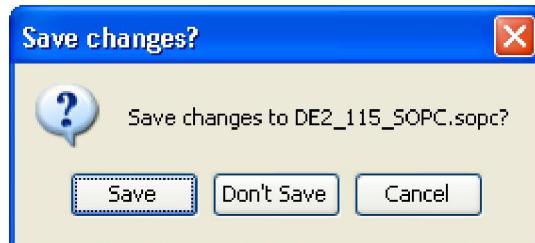


Figure 1-36 Generate SOPC

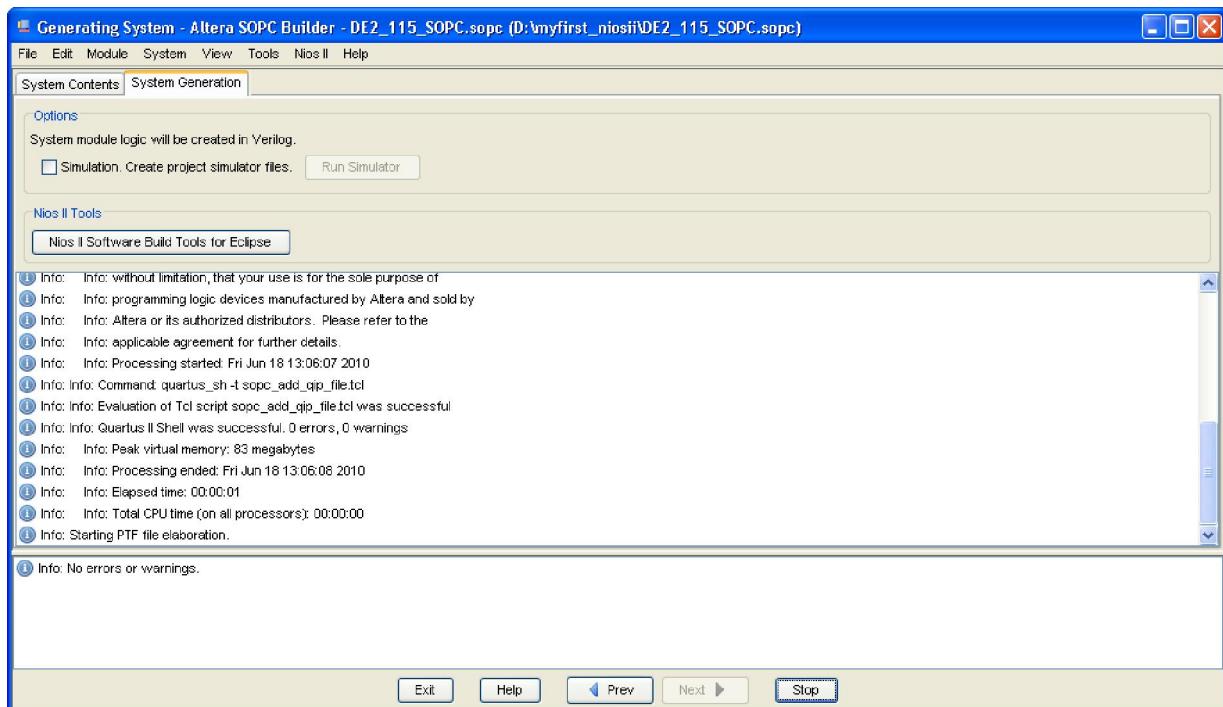


Figure 1-37 Generate SOPC

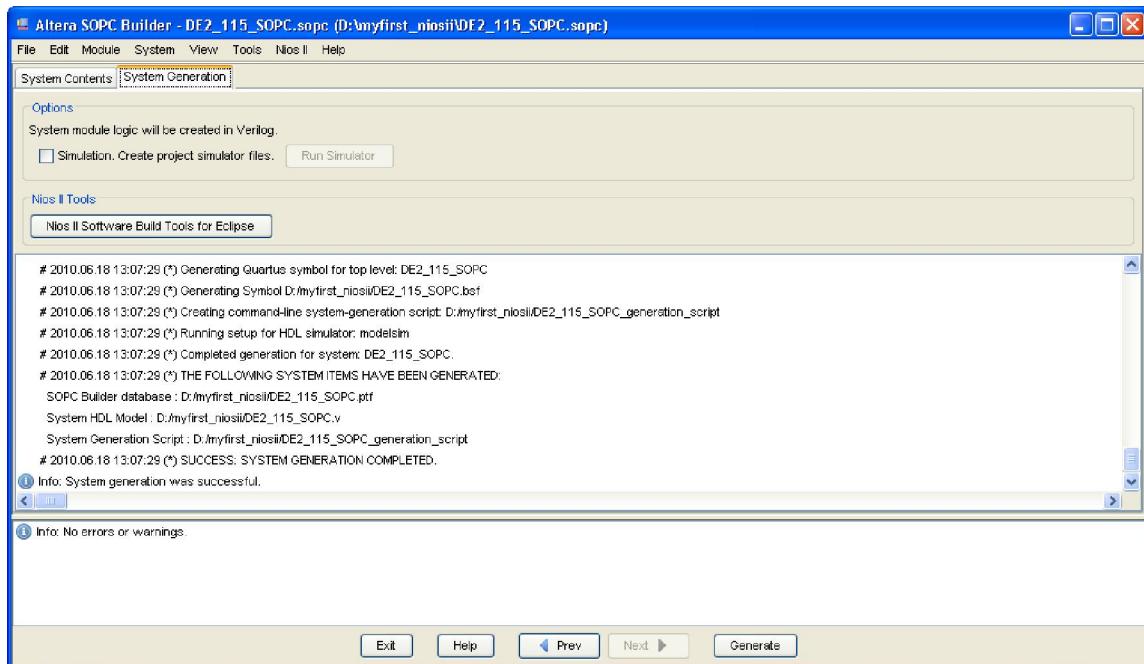


Figure 1-38 Generate SOPC Completely

24. Click **Exit** to exit the SOPC Builder and return to the window as shown in **Figure 1-39**.

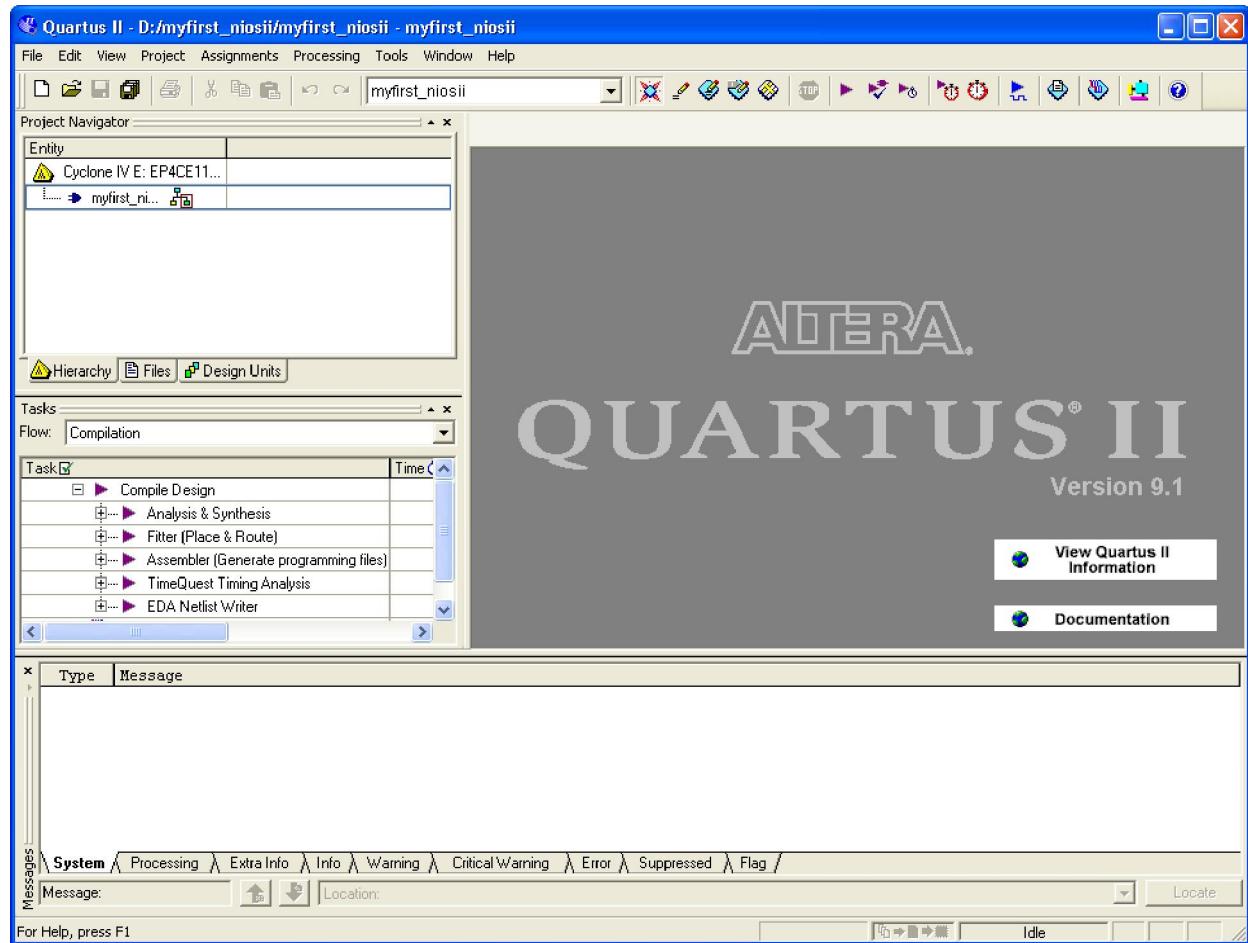
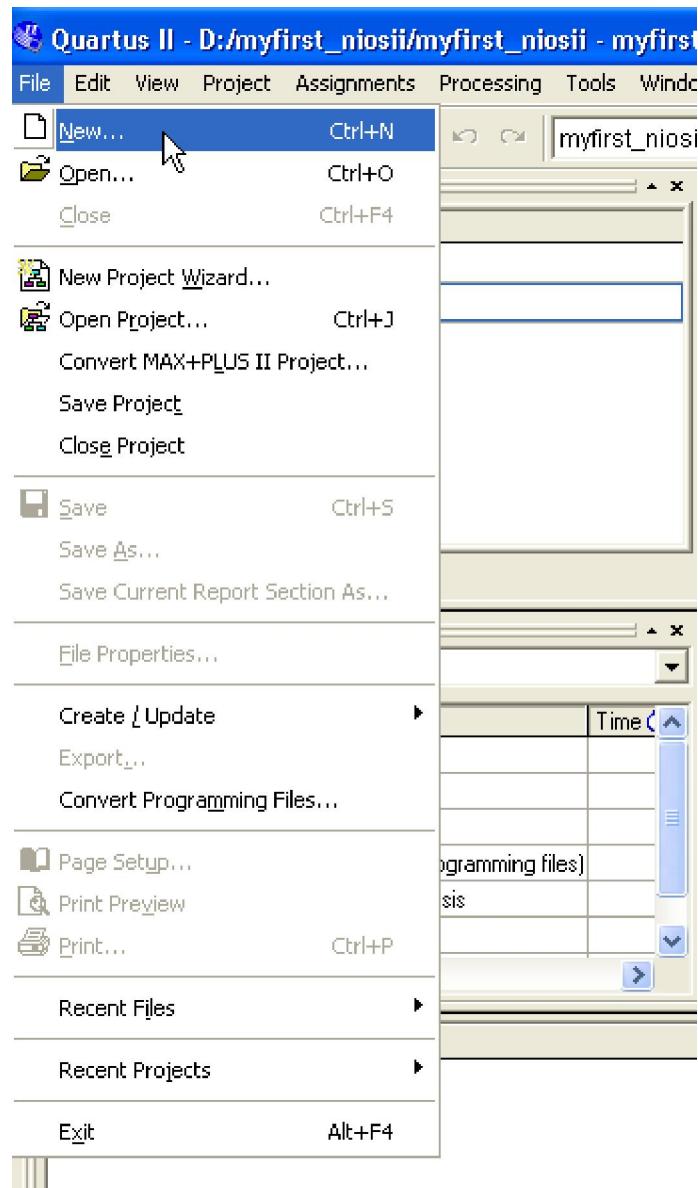


Figure 1-39 Exit SOPC

25. Choose **File > New** to open new files wizard. See **Figure 1-40** and **Figure 1-41**.

**Figure 1-40 New Verilog file**

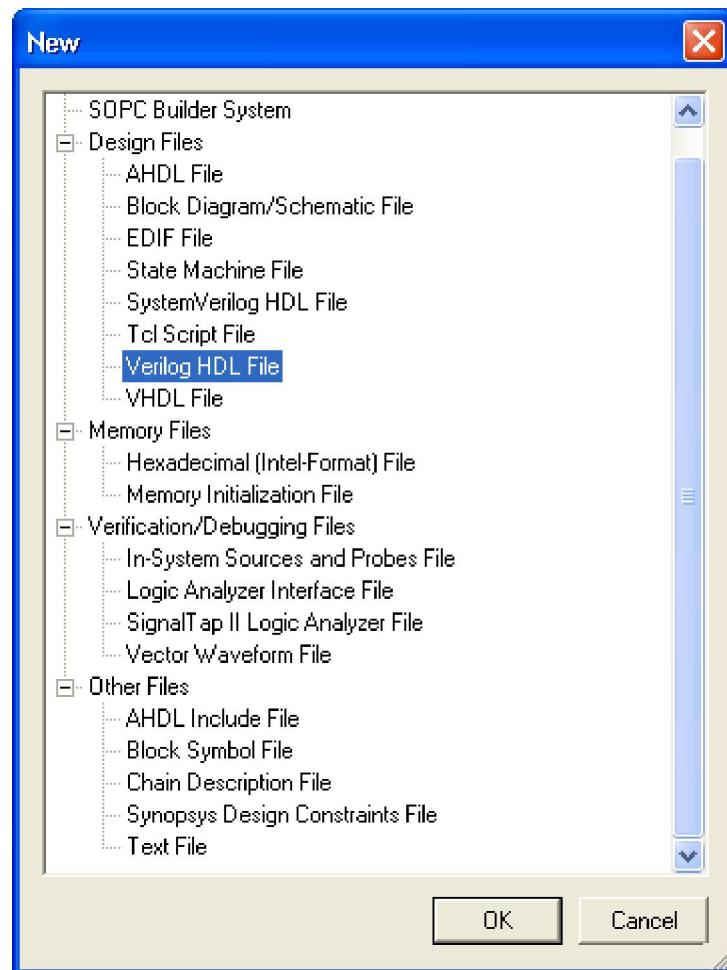


Figure 1-41 New Verilog File

33. Choose **Verilog HDL File** and click **OK** to return to the window as shown in Figure 1-42.

Figure 1-42 show a blank verilog file.

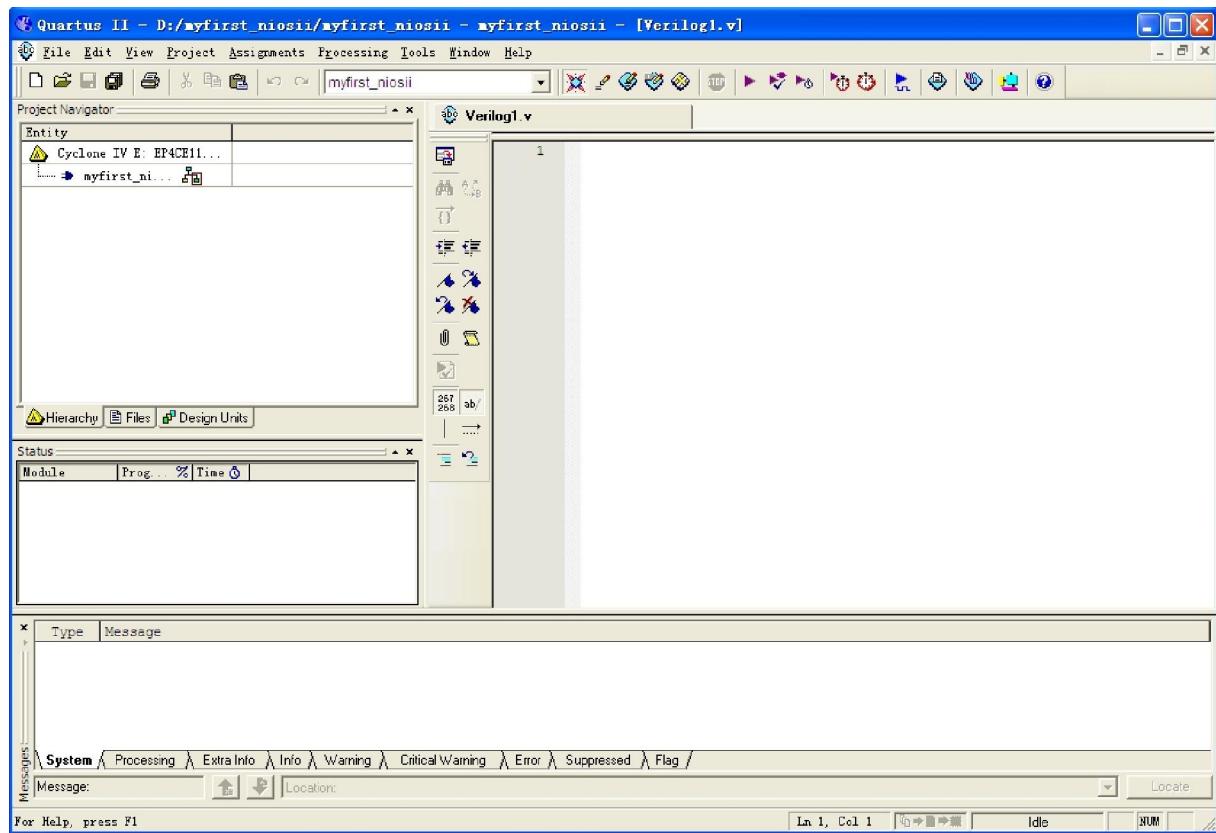


Figure 1-42 A blank verilog file

34. Type verilog the following script as shown in Figure 1-43. The module **DE2_115_SOPC** of the code is

from **DE2_115_SOPC.v** of the project. See

Figure 1-44 and Figure 1-45.

```
module myfirst_niosii
(
    CLOCK_50,
    LED
);
input      CLOCK_50;
output [7:0]  LED;
DE2_115_SOPC  DE2_115_SOPC_inst
(
    .clk_50          (CLOCK_50),
    .out_port_from_the_pio_led  (LED),
    .reset_n         (1'b1)
);

endmodule
```

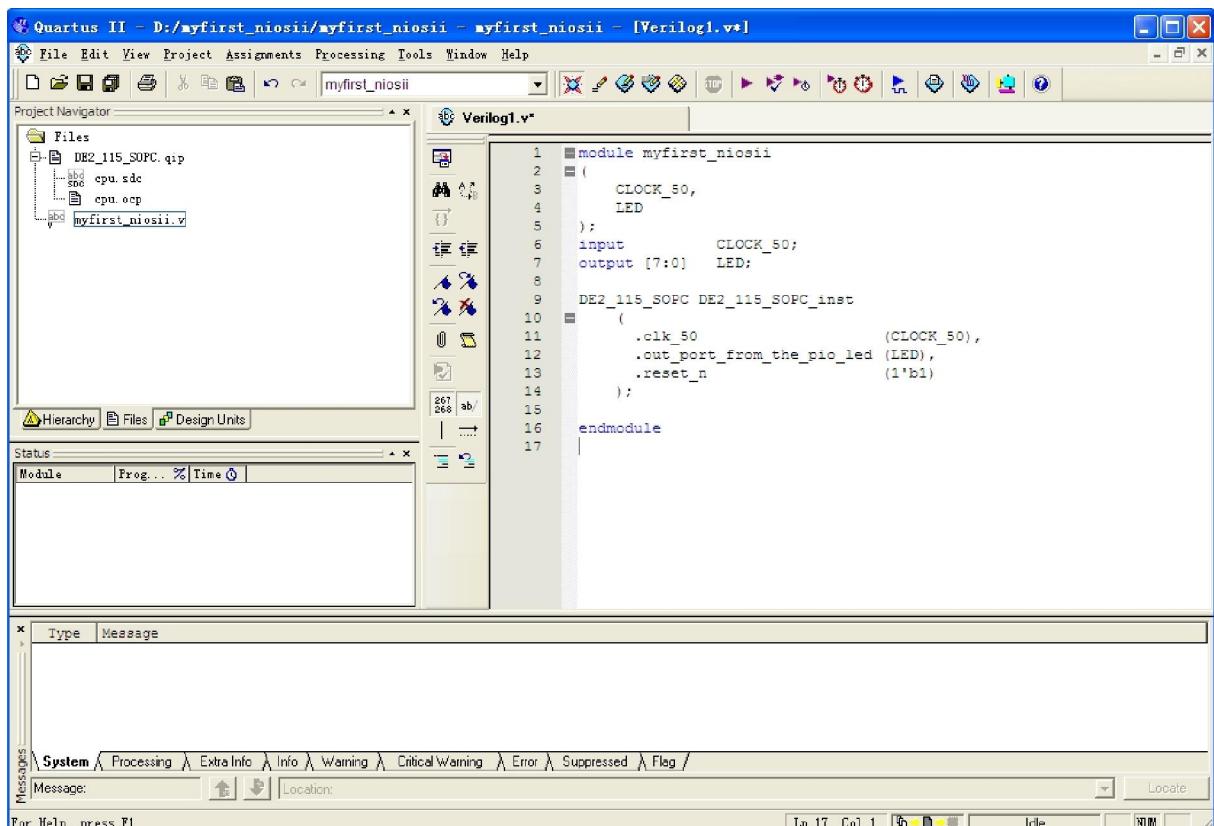


Figure 1-43 Input verilog Text

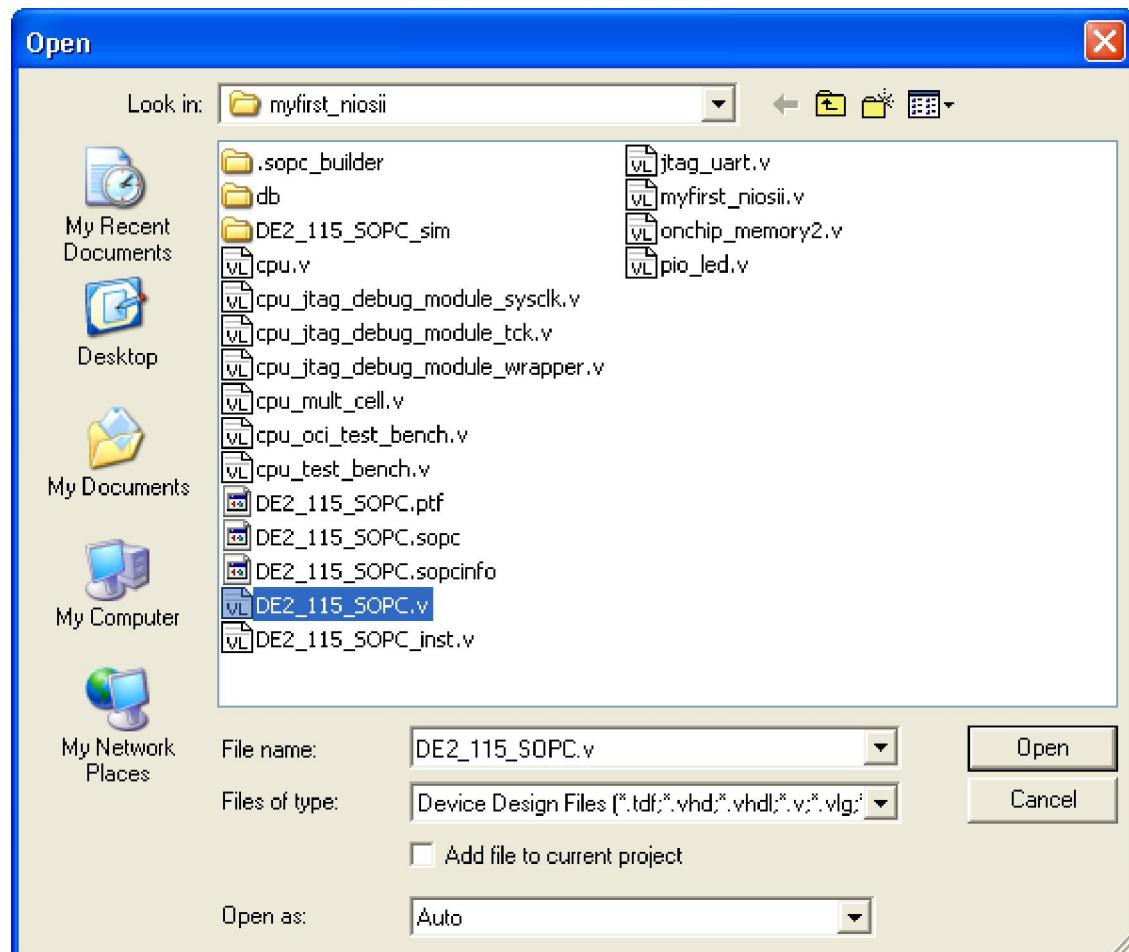


Figure 1-44 Open DE2_115_SOPC.v

```
2079 // altera message_level Level1
2080 // altera message_off 10034 10035 10036 10037 10230 102
2081
2082 module DE2_115_SOPC (
2083                                     // 1) global signals:
2084                                     clk_50,
2085                                     reset_n,
2086
2087                                     // the_pio_led
2088                                     out_port_from_the_pio_led
2089 );
2090 ;
2091
2092     output [ 7: 0] out_port_from_the_pio_led;
2093     input      clk_50;
2094     input      reset_n;
2095
2096     wire      clk_50_reset_n;
2097     wire [19: 0] cpu_data_master_address;
2098     wire [19: 0] cpu_data_master_address_to_slave;
2099     wire [ 3: 0] cpu_data_master_bytenable;
2100     wire      cpu_data_master_debugaccess;
2101     wire      cpu_data_master_granted;
2102     wire      cpu_data_master_granted_dly;
2103
2104     // 2) local signals:
2105     wire      led;
2106
2107     // 3) component declarations:
2108
2109     component DE2_115_SOPC
2110     port
2111         begin
2112             out_port_from_the_pio_led;
2113             clk_50;
2114             reset_n;
2115
2116             clk_50_reset_n;
2117             cpu_data_master_address;
2118             cpu_data_master_address_to_slave;
2119             cpu_data_master_bytenable;
2120             cpu_data_master_debugaccess;
2121
2122             led;
2123
2124             end
2125     endcomponent
2126
2127     // 4) connections:
2128
2129     assign out_port_from_the_pio_led = led;
2130
2131     assign clk_50 = clk_50;
2132     assign reset_n = reset_n;
2133
2134     assign clk_50_reset_n = clk_50_reset_n;
2135     assign cpu_data_master_address = cpu_data_master_address;
2136     assign cpu_data_master_address_to_slave = cpu_data_master_address_to_slave;
2137     assign cpu_data_master_bytenable = cpu_data_master_bytenable;
2138     assign cpu_data_master_debugaccess = cpu_data_master_debugaccess;
2139
2140     assign led = out_port_from_the_pio_led;
2141
2142     assign out_port_from_the_pio_led = led;
```

Figure 1-45 DE2_115_SOPC module

35. Choose **Save** Icon in the tool bar. There will appear a window as shown in **Figure 1-46**. Click **Save**.

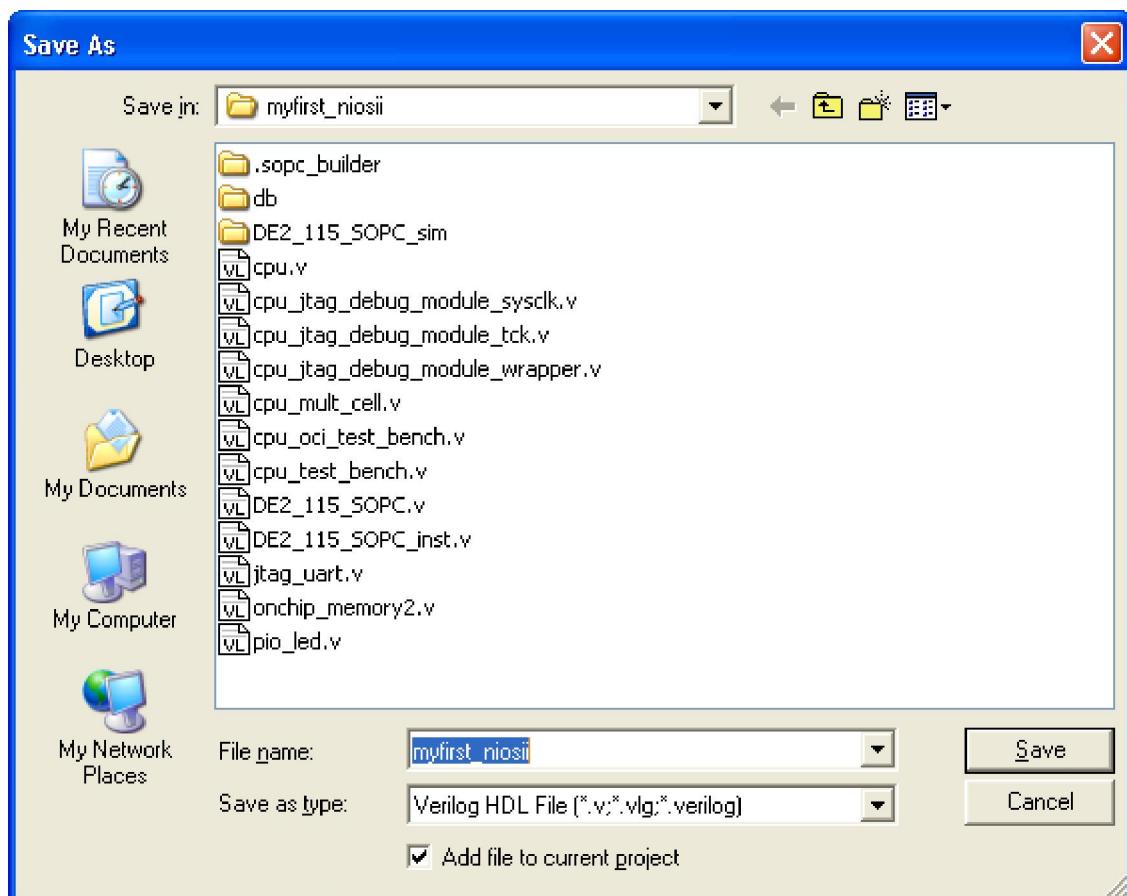


Figure 1-46 Save Verilog file

36. Choose **Processing > Start Compilation** as shown in Figure 1-47. Figure 1-48 shows the compilation process.

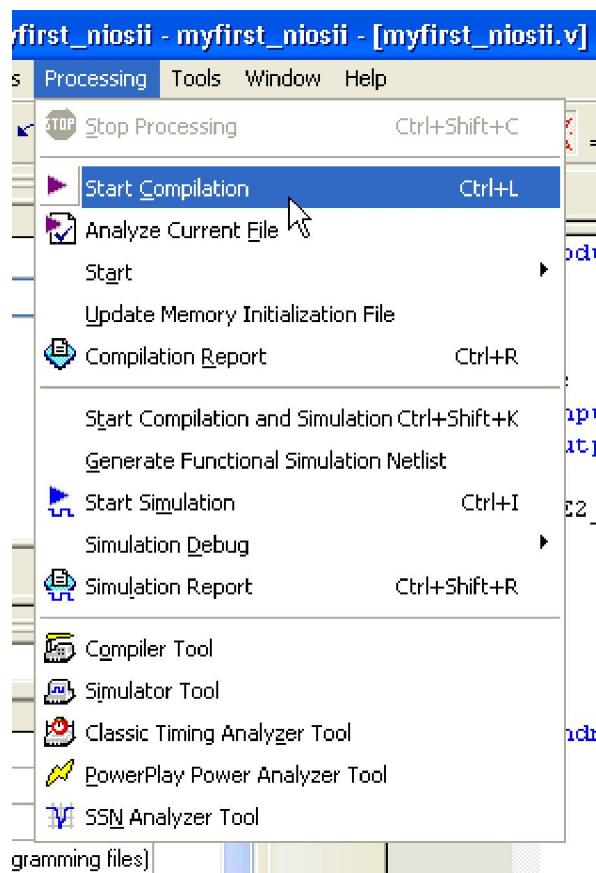


Figure 1-47 Start Compilation

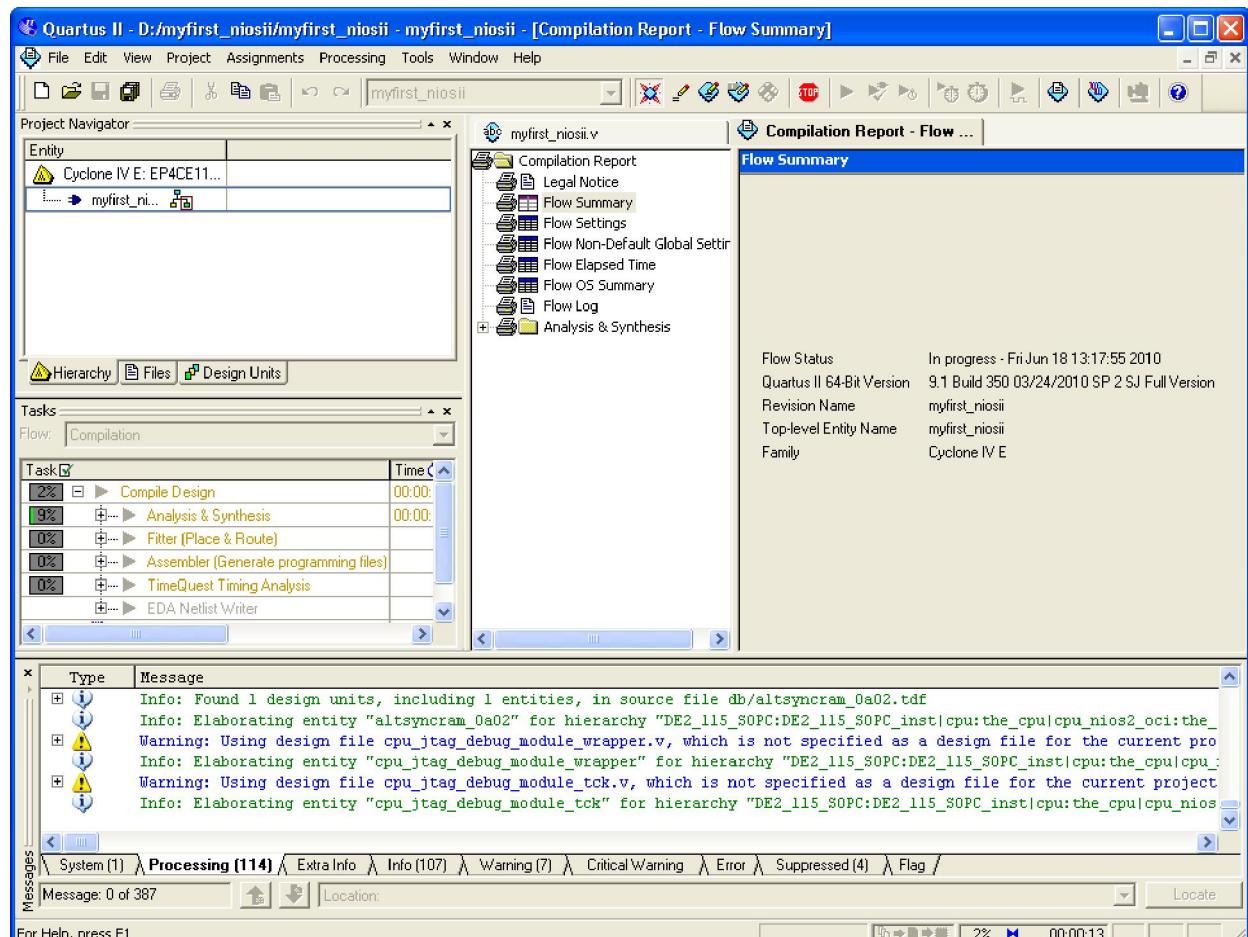


Figure 1-48 Execute Compilation

Note: In the compilation, if there is the error which shows “Error: The core supply voltage of ‘1.0v’ is illegal for the currently selected part.”, you should modify the text “set_global_assignment -name NOMINAL_CORE_SUPPLY_VOLTAGE 1.0V” to “set_global_assignment -name NOMINAL_CORE_SUPPLY_VOLTAGE 1.2V” in the myfirst_niosii.qsf of the project.

37. A window that shows successfully will appear as shown in **Figure 1-49**.

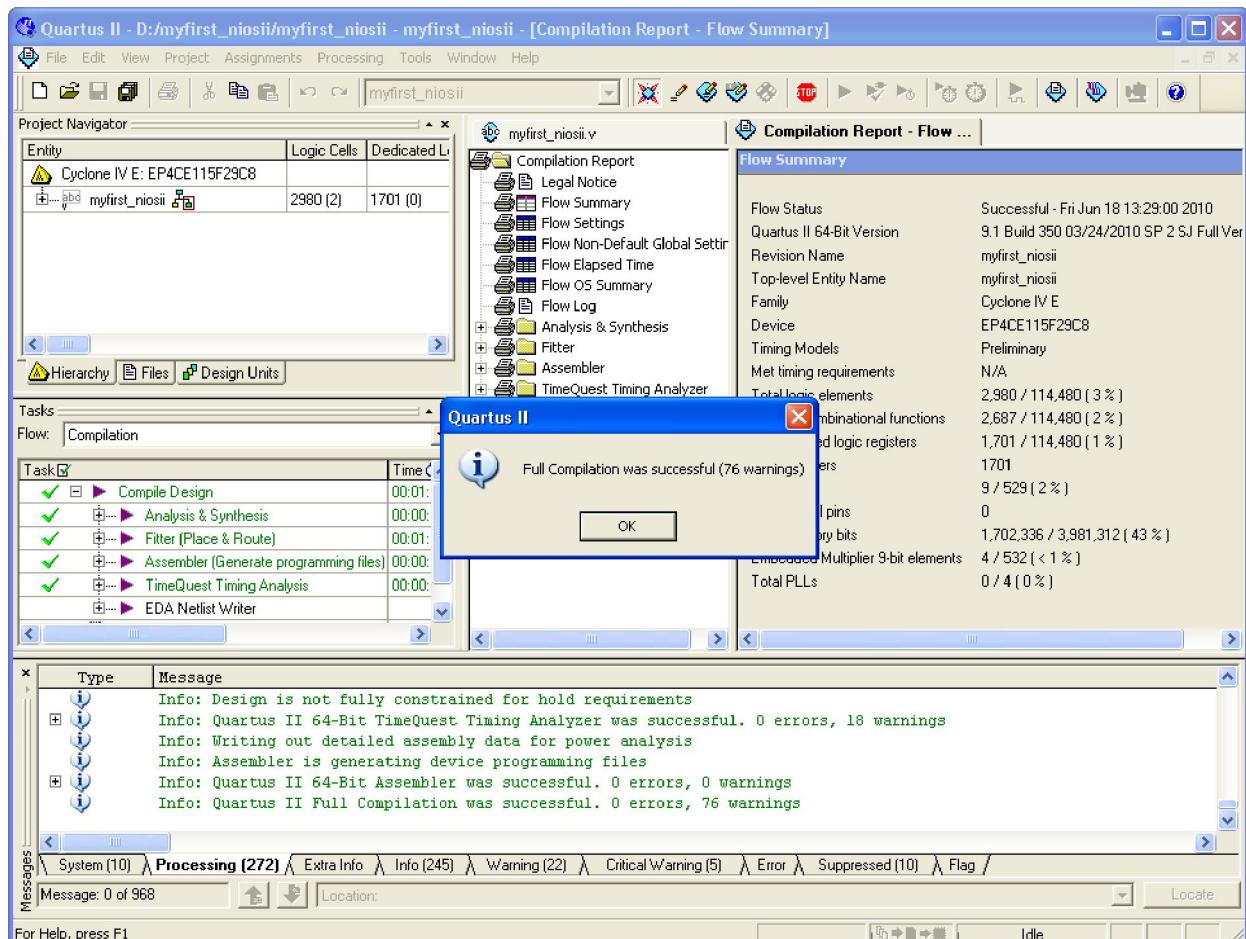


Figure 1-49 Compilation project completely

45. Choose **Assignments > Pins** to open pin planner as shown in Figure 1-50. Figure 1-51 show blank pins.

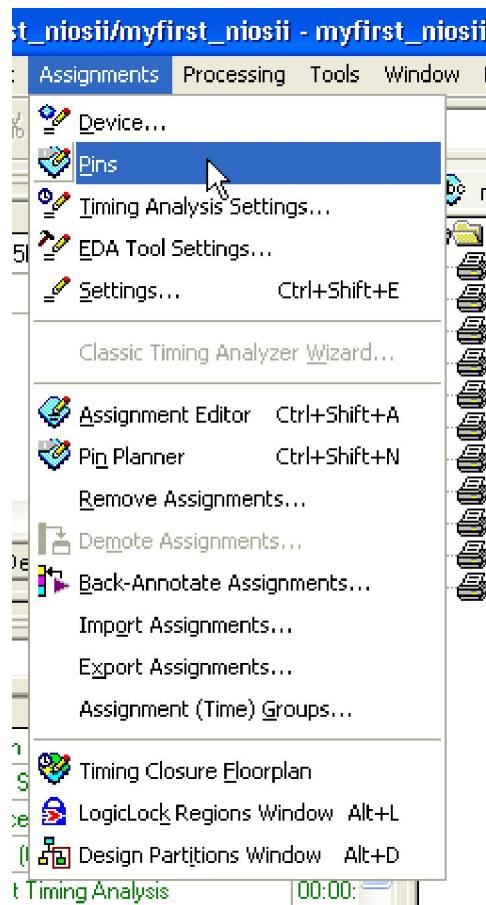


Figure 1-50 Pins menu

Named: <input type="button" value="New"/> <input type="button" value="Edit"/> <input type="button" value="X"/> <input type="button" value="V"/>							
	Node Name	Location	I/O Standard	I/O Bank	Direction	VREF Group	Reserved
1	CLOCK_50		2.5 V (default)		Input		
2	LED[7]		2.5 V (default)		Output		
3	LED[6]		2.5 V (default)		Output		
4	LED[5]		2.5 V (default)		Output		
5	LED[4]		2.5 V (default)		Output		
6	LED[3]		2.5 V (default)		Output		
7	LED[2]		2.5 V (default)		Output		
8	LED[1]		2.5 V (default)		Output		
9	LED[0]		2.5 V (default)		Output		
10	<<new node>>						

Figure 1-51 Blank Pins

46. Input Location value as shown in Figure 1-52.

	Node Name	Location	I/O Standard	I/O Bank	Direction	VREF Group	Reserved
1	CLOCK_50	PIN_Y2	2.5 V (default)	2	Input	B2_N0	
2	LED[0]	PIN_E21	2.5 V (default)	7	Output	B7_N0	
3	LED[1]	PIN_E22	2.5 V (default)	7	Output	B7_N0	
4	LED[2]	PIN_E25	2.5 V (default)	7	Output	B7_N1	
5	LED[3]	PIN_E24	2.5 V (default)	7	Output	B7_N1	
6	LED[4]	PIN_H21	2.5 V (default)	7	Output	B7_N2	
7	LED[5]	PIN_G20	2.5 V (default)	7	Output	B7_N1	
8	LED[6]	PIN_G22	2.5 V (default)	7	Output	B7_N2	
9	LED[7]	PIN_G21	2.5 V (default)	7	Output	B7_N1	
10	<<new node>>						

Figure 1-52 Set Pins

47. Close the **pin planner**. Restart compilation the project.

1.3 Download Hardware Design to Target FPGA

This section describes how to download the configuration file to the board.

Download the FPGA configuration file (i.e. the SRAM Object File (.sof) that contains the NIOS II standard system) to the board 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.
4. After the welcome page appears, click **Workbench**.
5. Choose **Tools->Quartus II Programmer**.
6. Click **Auto Detect**. The device on your development board should be detected automatically.
7. Click the top row to highlight it.
8. Click **Change File**.
9. Browse to the myfirst_niosii project directory.
10. Select the programming file (myfirst_niosii.sof) for your board.
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.

Note: If the appropriate download cable does not appear in the list, you must first install drivers for the cable. Refer to Quartus II Help for information on how to install the driver. See Figure 1-53.

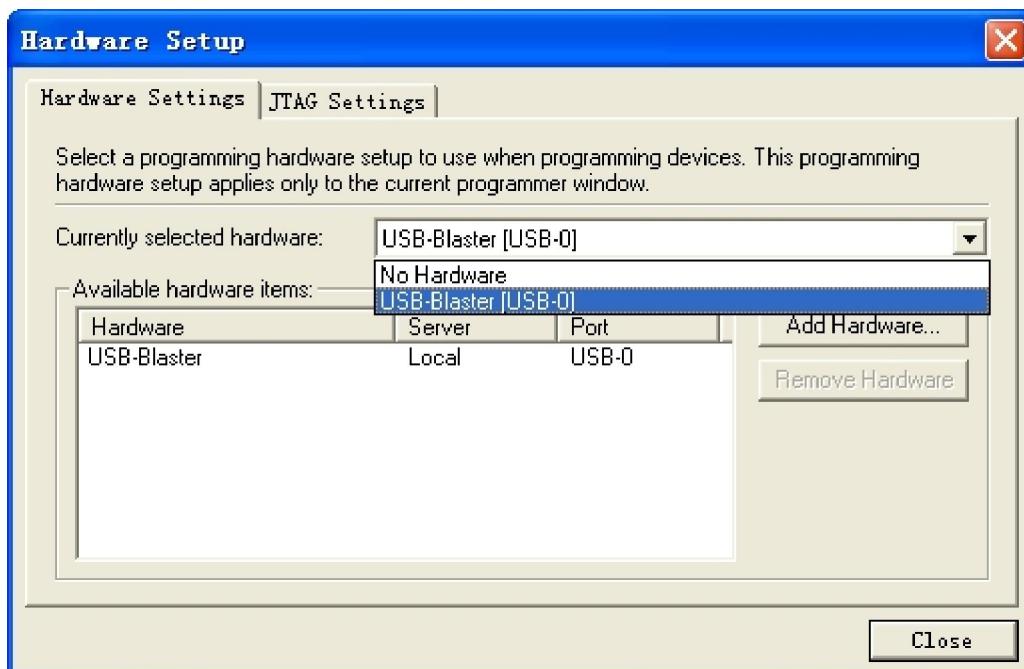


Figure 1-53 Hardware Setup Window

14. Click **Close**.

15. Turn on the **Program/Configure** option for the programming file.(See Figure 1-54 for an example).

16. Click **Start**.

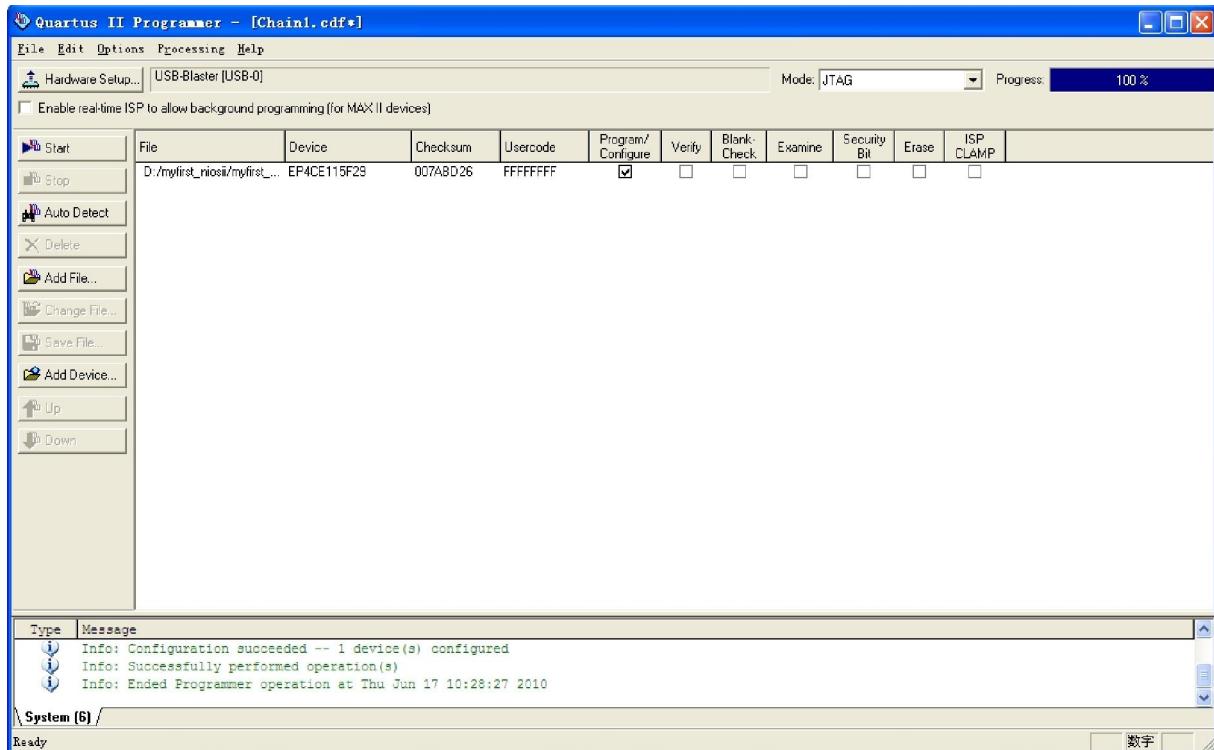


Figure 1-54 Quartus II Programmer

The Progress meter sweeps to 100% after the configuration finished. 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.

Chapter 2 **NIOS II IDE Build Flow**

This Chapter covers build flow of Nios II C coded software program.

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 C-to-Hardware (C2H) compiler GUI. The included example software application templates make it easy for new software programmers to get started quickly. In this section you will use the Nios II IDE to compile a simple C language example 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.

2.1 Create the hello_world Example Project

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

1. Return to the NIOS II IDE.

Note: you can close the Quartus II Programmer or leave it open in the background if you want to reload the processor system onto your development board quickly.

2. Choose **File > Switch Workspace** to switch workspace. See Figure 2-1 and Figure 2-2.

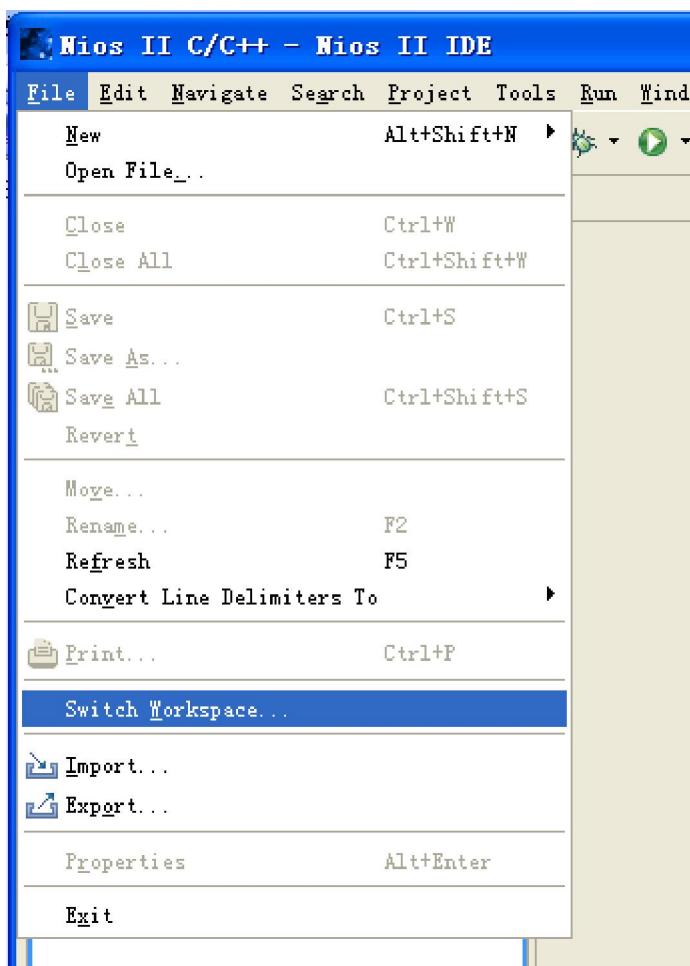


Figure 2-1 Switch Workspace (1)

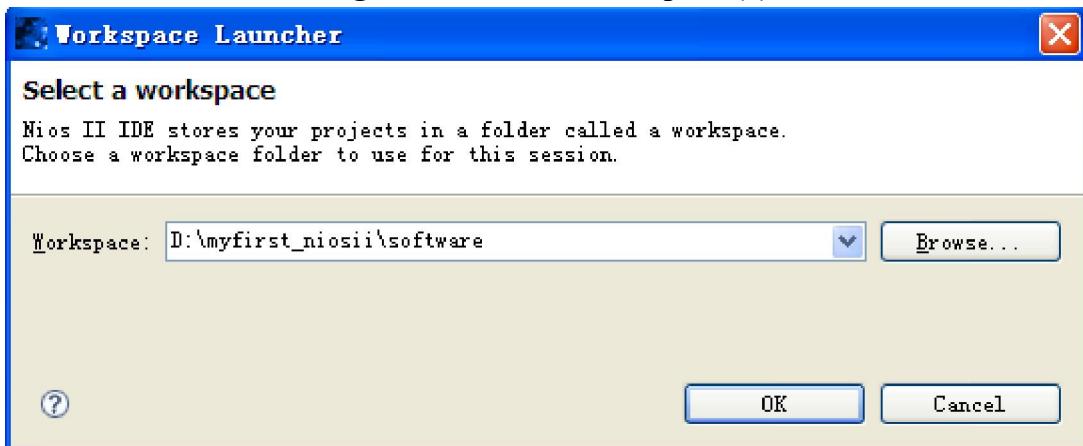


Figure 2-2 Switch Workspace (2)

3. Choose **File->New->NIOS II C/C++ Application** to open the New Project Wizard.
4. In the New Project wizard, make sure the following things:

- Select the **Hello World** project template.
- Give the project a name. (**hello_world_0** is default name)
- Select the target hardware system PTF file that locates in where the previously created hardware project resides as shown in Figure 2-3.

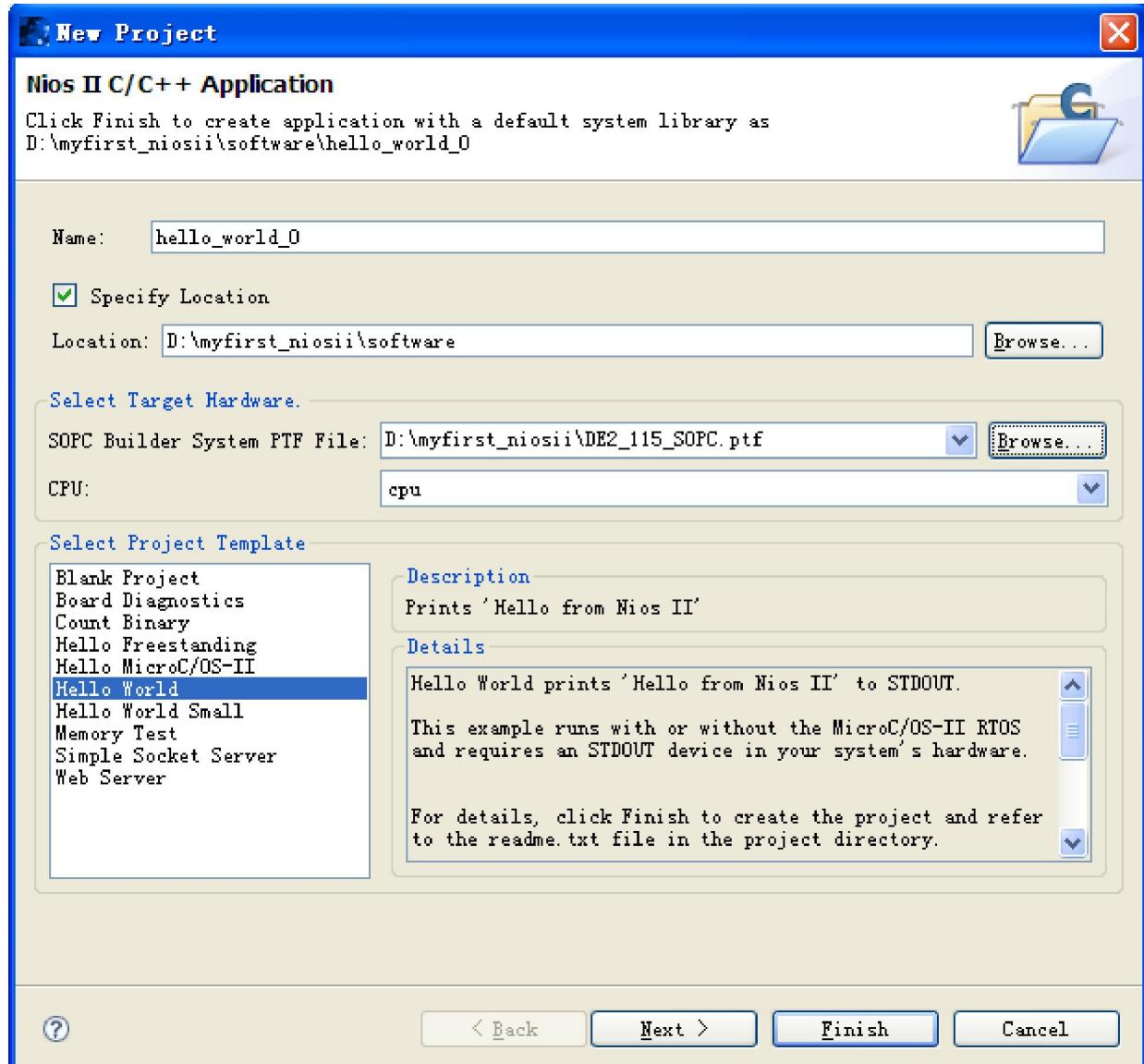


Figure 2-3 Nios II IDE New Project Wizard

5. Click **Finish**. The Nios II IDE creates the **hello_world_0** project and returns to the Nios II C/C++ project perspective. See **Figure 2-4**.

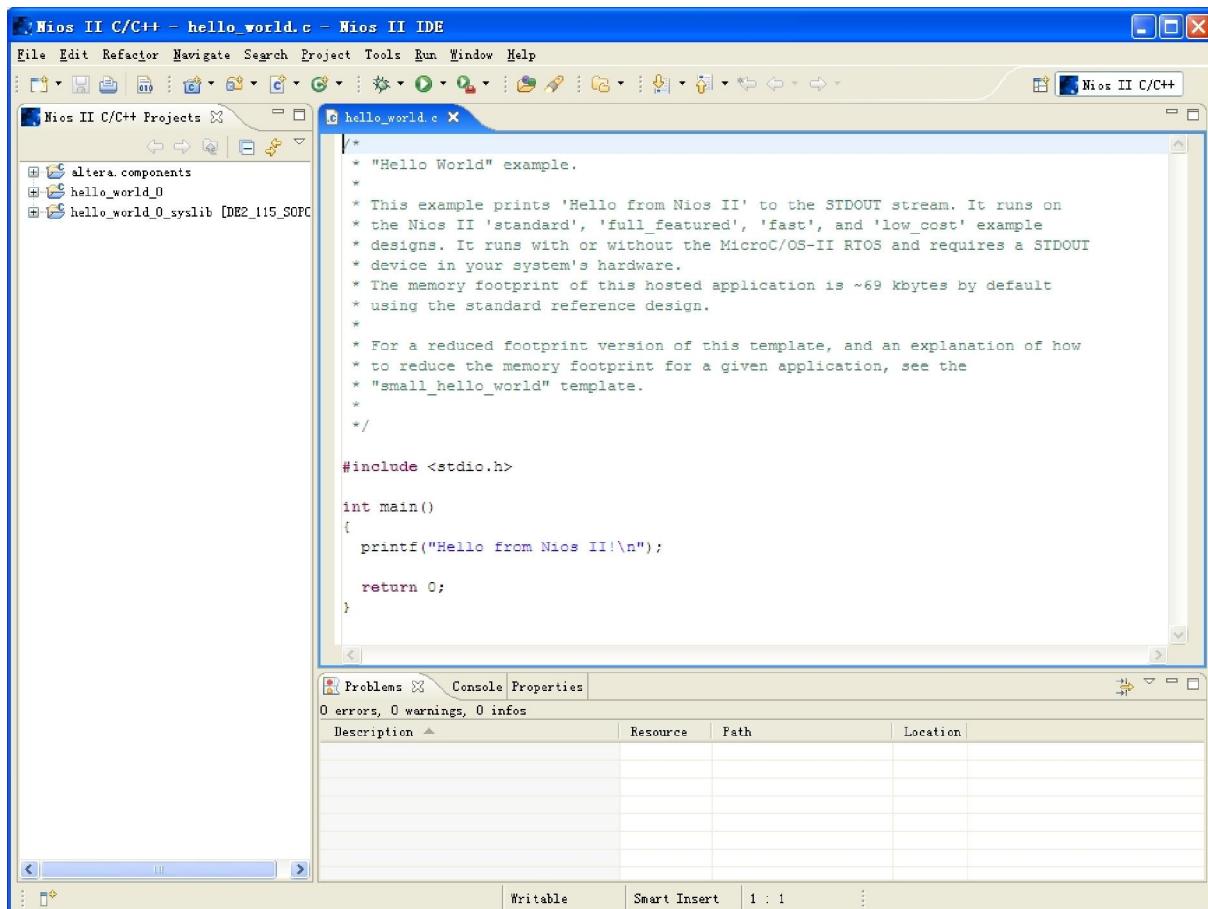


Figure 2-4 Nios II IDE C++ Project Perspective for hello_world_0

When 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 your C/C++ application project. This project contains the 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.

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

- Installed 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 software packages which includes NIOS II hardware abstraction layer, NicheStack TCP/IP

Network stack, NIOS II host file system, NIOS II read-only zip file system and Micrium's µC/OS-II real time 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 hello_world_0 folder under Debug.

2.2 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, a message ‘Build complete’ will appear in the Console tab. The compilation time varies depending on your system. See Figure 2-5 for an example.

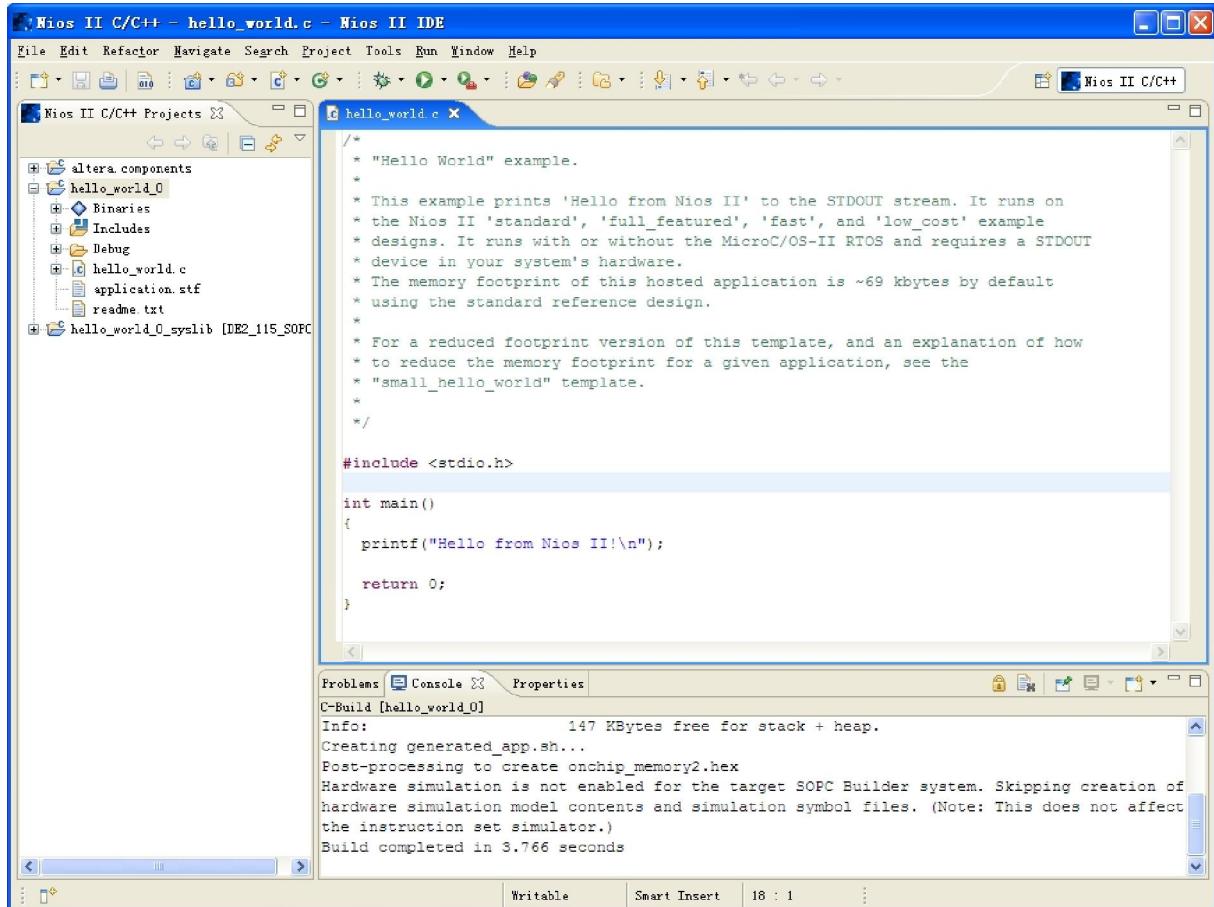


Figure 2-5 Nios II IDE hello_world_0 Build Completed

After compilation complete, right-click the **hello_world_0** project, choose **Run As**, and choose **NIOS II Hardware**. The IDE begins to download the program to the target FPGA development board and begins execution. When the target hardware begins executing the program, the message '**Hello from Nios II!**' appears in the NIOS II IDE Console tab. See Figure 2-6 for an example.

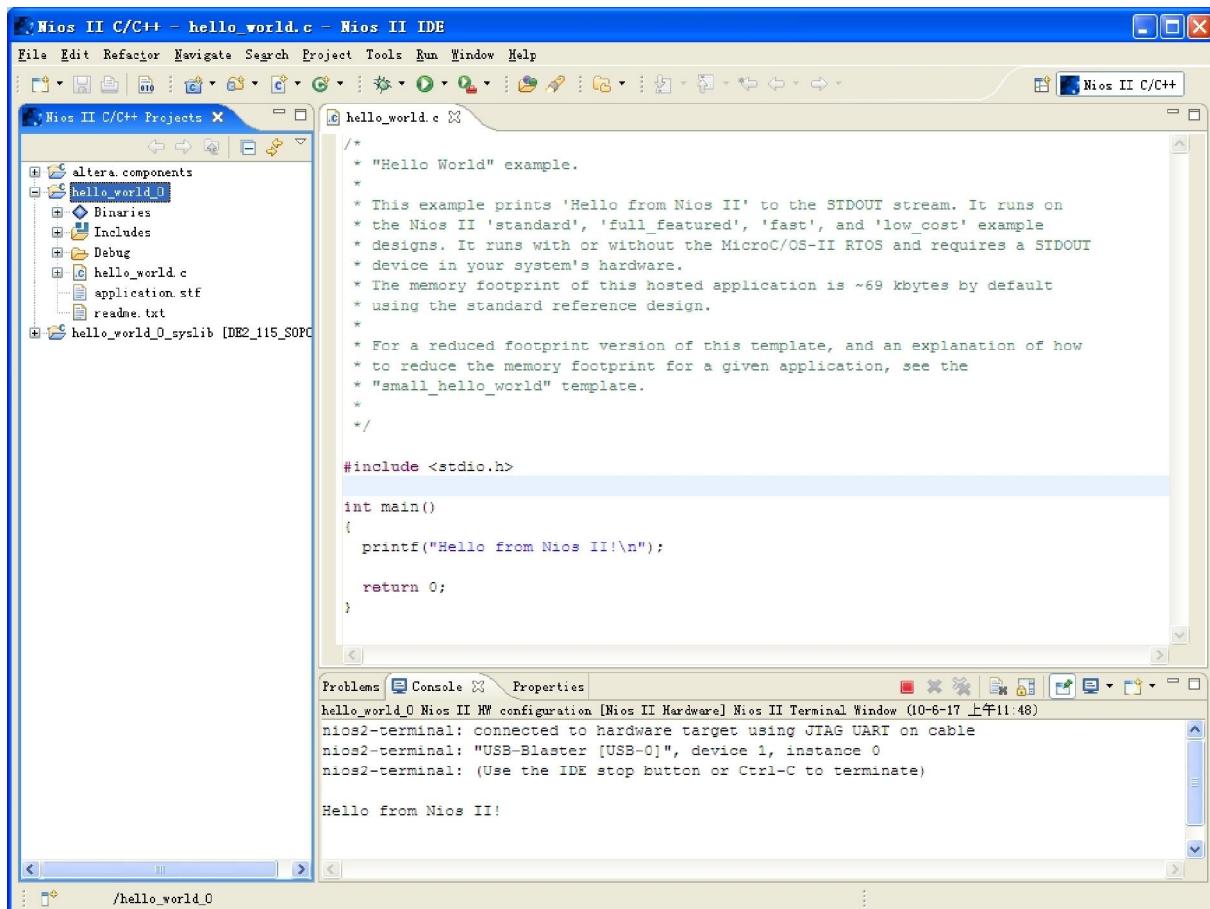


Figure 2-6 Hello_World_0 Program Output

Now you have created, compiled, and run your first software program based on NIOS II. And you can perform additional operations such as configuring the system properties, editing and re-building the application, and debugging the source code.

2.3 Edit and Re-Run the Program

You can modify the **hello_world.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 will make LEDG blink.

Perform the following steps to modify and re-run the program:

1. In the **hello_world.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(PIO_LED_BASE, count & 0x01);

delay = 0;

while(delay < 2000000)

{

delay++;

}

count++;

}

return 0;

}
```

2. Save the project.

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

Note: You do not need to build the project manually; the Nios II IDE automatically re-builds the

program before downloading it to the FPGA.

- Orient your development board so that you can observe LEDG blinking.

2.4 Why the LED Blinks

The Nios II system description header file, **system.h**, contains the software definitions, name, locations, base addresses, and settings for all of the components in the Nios II hardware system. The **system.h** file is located in the in the **hello_world_0_syslib\Debug\system_description** directory as shown in Figure 2-7.

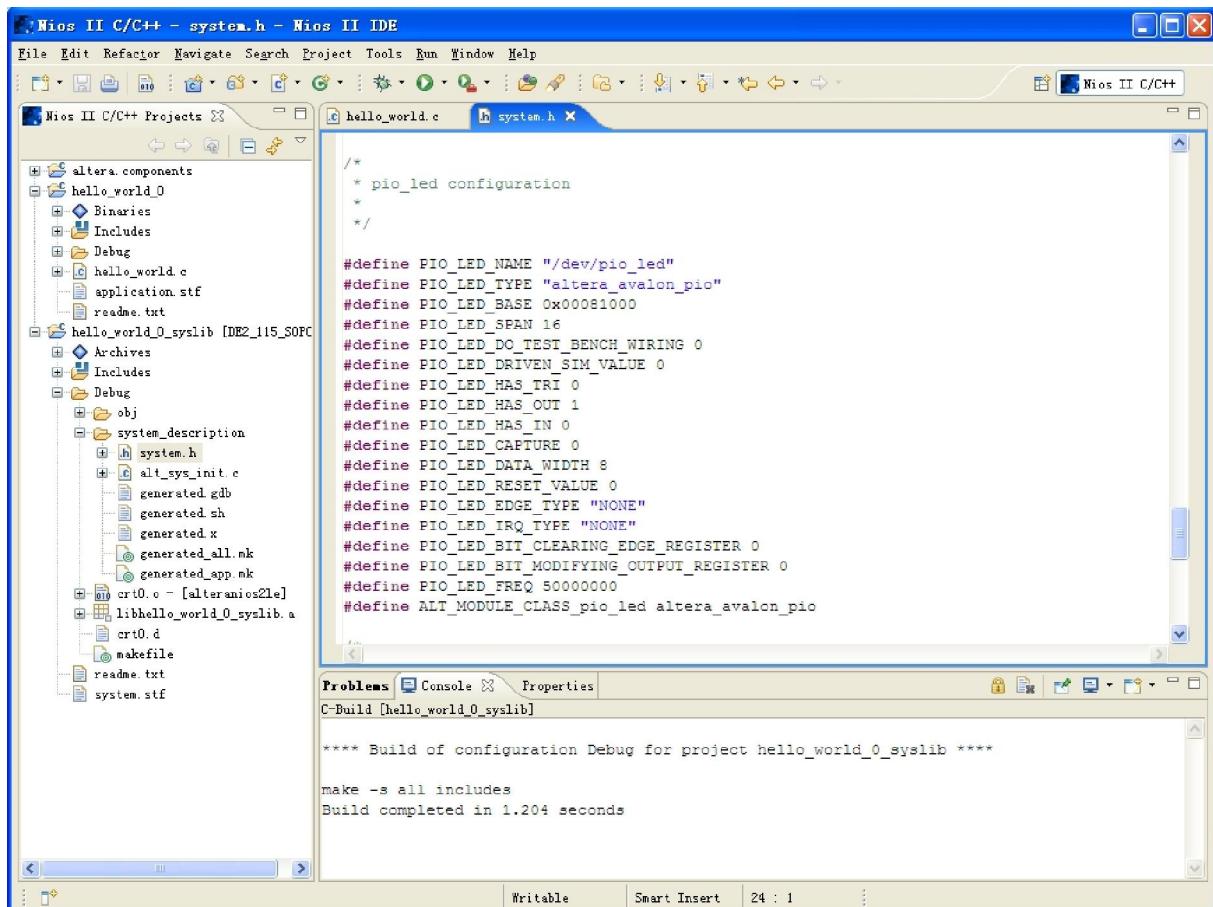


Figure 2-7 system.h Location

If you look at the **system.h** file for the Nios II project example used in this tutorial, you will notice the **pio_led** function. This function controls the LED. The Nios II processor 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**, **interrupt mask**, and **edge capture**. To turn the LED on and off, the application 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 **altera\<version number>\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 function

IOWR_ALTERA_AVALON_PIO_DATA(base, data)

can write to the PIO data register, turning 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.

2.5 Debugging the Application

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

1. In the **hello_world.c**, double-click the front of the line which is needed to set breakpoint. See Figure 2-8.

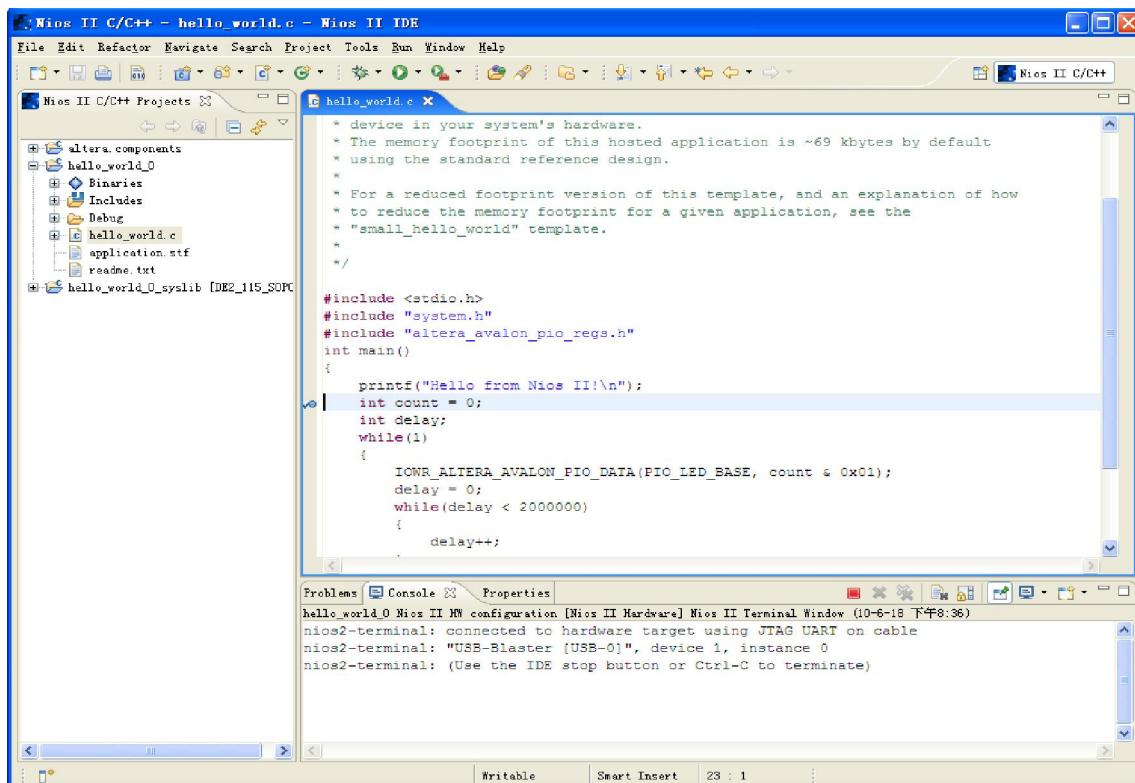


Figure 2-8 Set Breakpoint

2. To debug your application, right-click the application (**hello_world_0** by default) and choose **Debug as > Nios II Hardware**.

3. If the **Confirm Perspective Switch** message 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.

5. 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.

Note: To return to the Nios II C/C++ project perspective from the debug perspective, click the two arrows >> in the top right corner of the GUI.

2.6 Configure System Library

In this section you will learn how to configure some advanced options about the target memory or other things. By performing the following steps, you can charge all the available settings:

1. In the Nios II IDE, right-click **hello_world_0** and choose **System Library Properties**. The **Properties for hello_world_0_syslib** dialog box opens.
2. Click **System Library**. The **System Library** page contains settings related to how the program interacts with the underlying hardware. The settings have names that correspond to the targeted NIOS II hardware.
3. In the **Linker Script** box, observe which memory has been assigned for **Program memory(.text)**, **Read-only data memory(.rodata)**, **Read/write data memory(.rwdta)**, **Heap memory**, and **Stack memory**, see Figure 2-9. These settings determine which memory is used to store the compiled executable program when the example **hello_world_0** programs runs. You can also specify which interface you want to use for stdio , stdin, and stderr. You can also add and configure an RTOS for your application and configure build options to support C++, reduced device drivers, etc.
4. Choose **onchip_memory2** for all the memory options in the **Linker Script** box. See Figure 2-9 for an example.

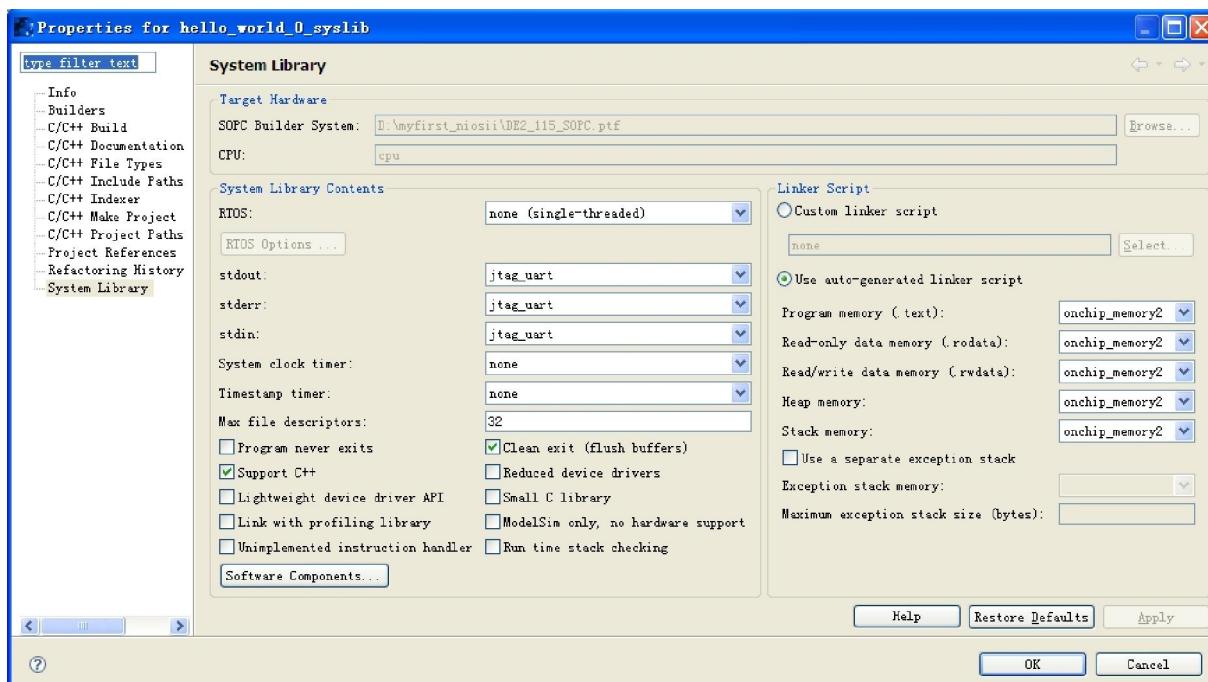


Figure 2-9 Configuring System Library Properties

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

Note: If you make changes to the system properties you must rebuild your project. To rebuild, right-click the **hello_world_0** project in the **Nios II C/C++ Projects** tab and choose **Build Project**.

Chapter 3 *Programming the CFI Flash*

With the density of FPGAs increasing, the need for larger configuration storage is also increasing. If your system contains a common flash interface (CFI) flash memory, you can use your system for FPGA configuration storage as well.

3.1 Modify the SOPC of the Project

1. Choose **Library > Bridges and Adapters > Memory Mapped >Avalon-MM Tristate Bridge** to open the Avalon-MM Tristate Bridge wizard. See Figure 3-1 and Figure 3-2.

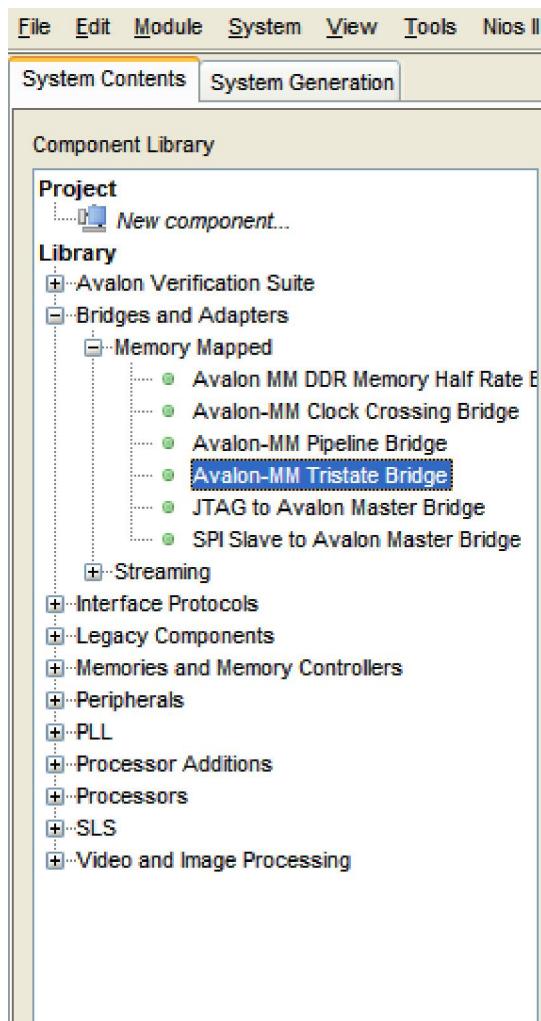


Figure 3-1 Avalon-MM Tristate Bridge

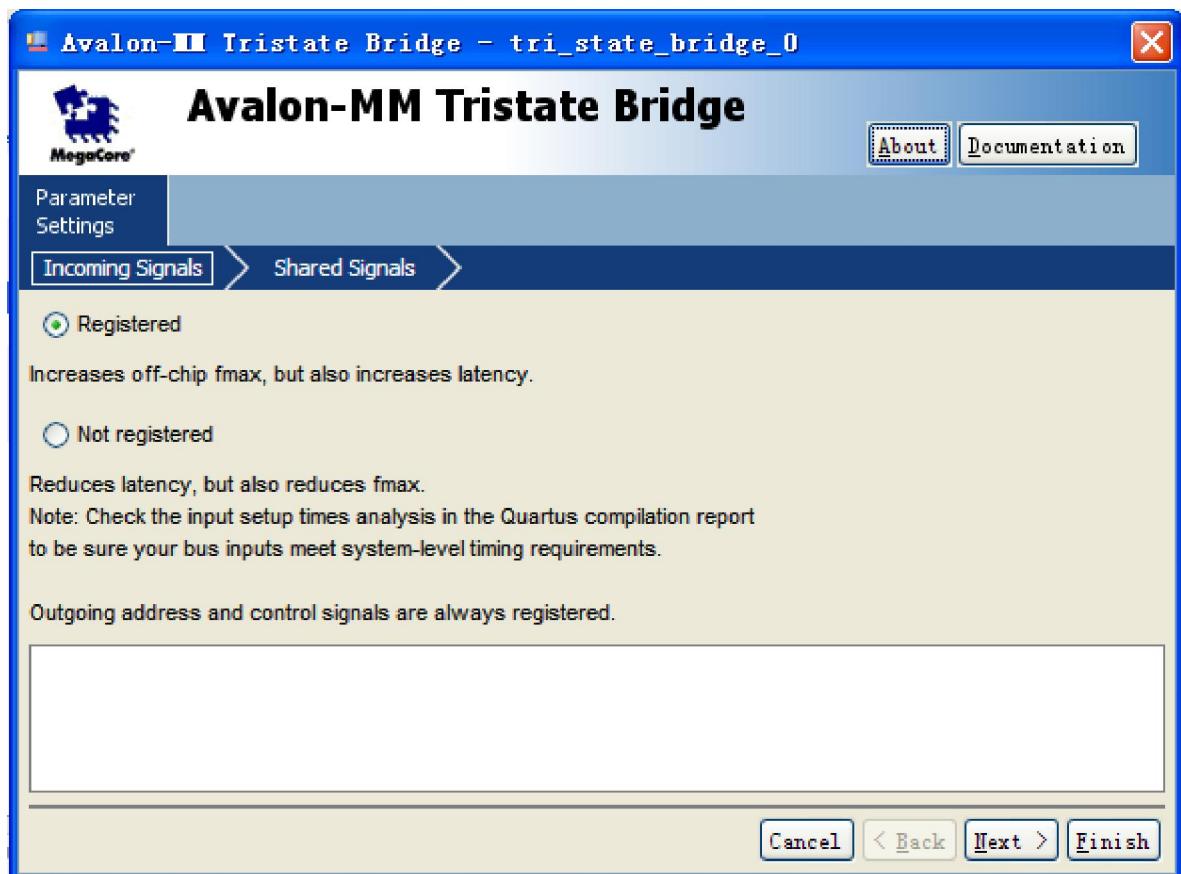


Figure 3-2 Avalon-MM Tristate Bridge Box

2. Click **Finish** to close Avalon-MM Tristate Bridge box, and return to the window as shown in Figure 3-3.

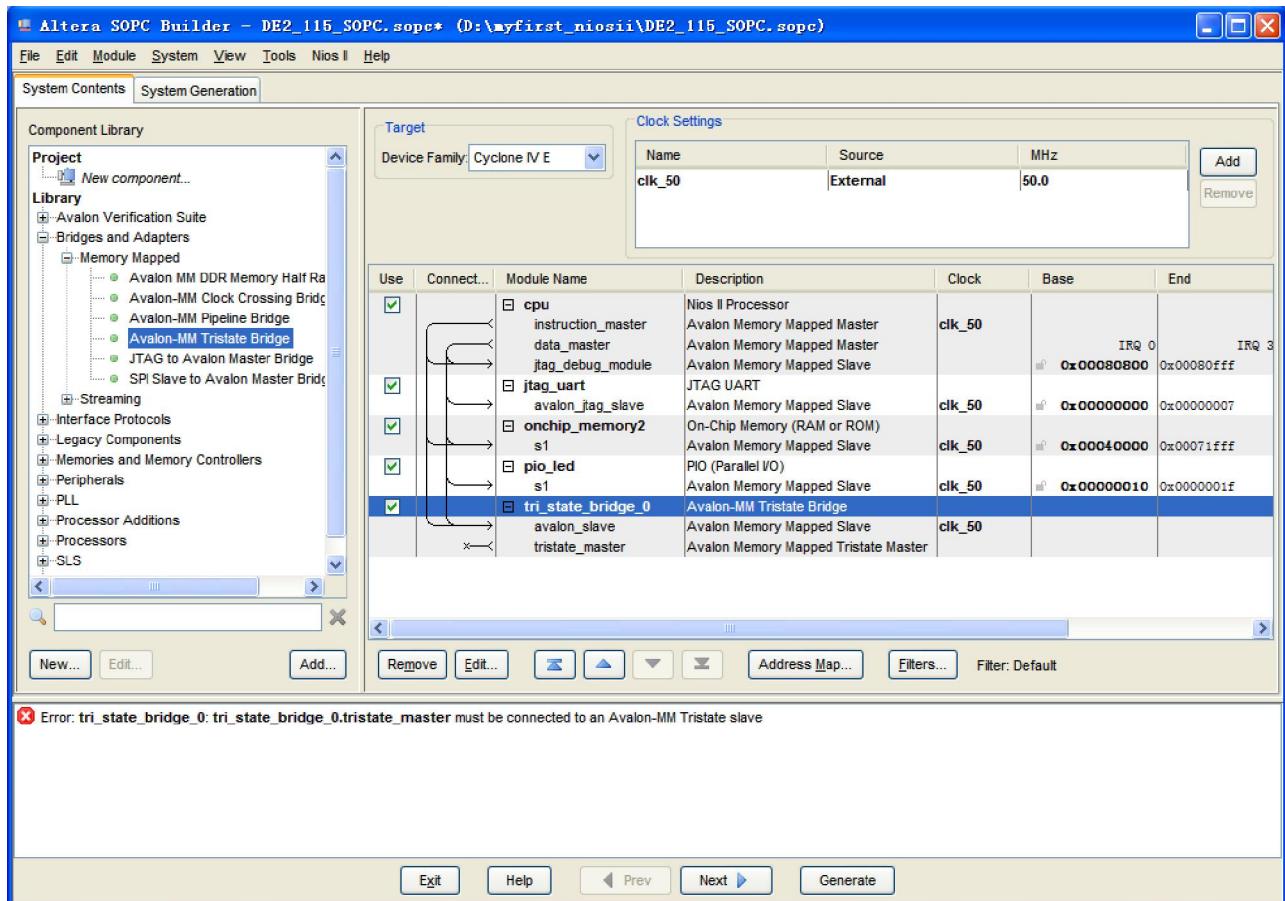


Figure 3-3 Add Avalon-MM Tristate Bridge Component

3. Rename **tri_state_bridge_0** to **tri_state_bridge_flash** as shown in **Figure 3-4**.

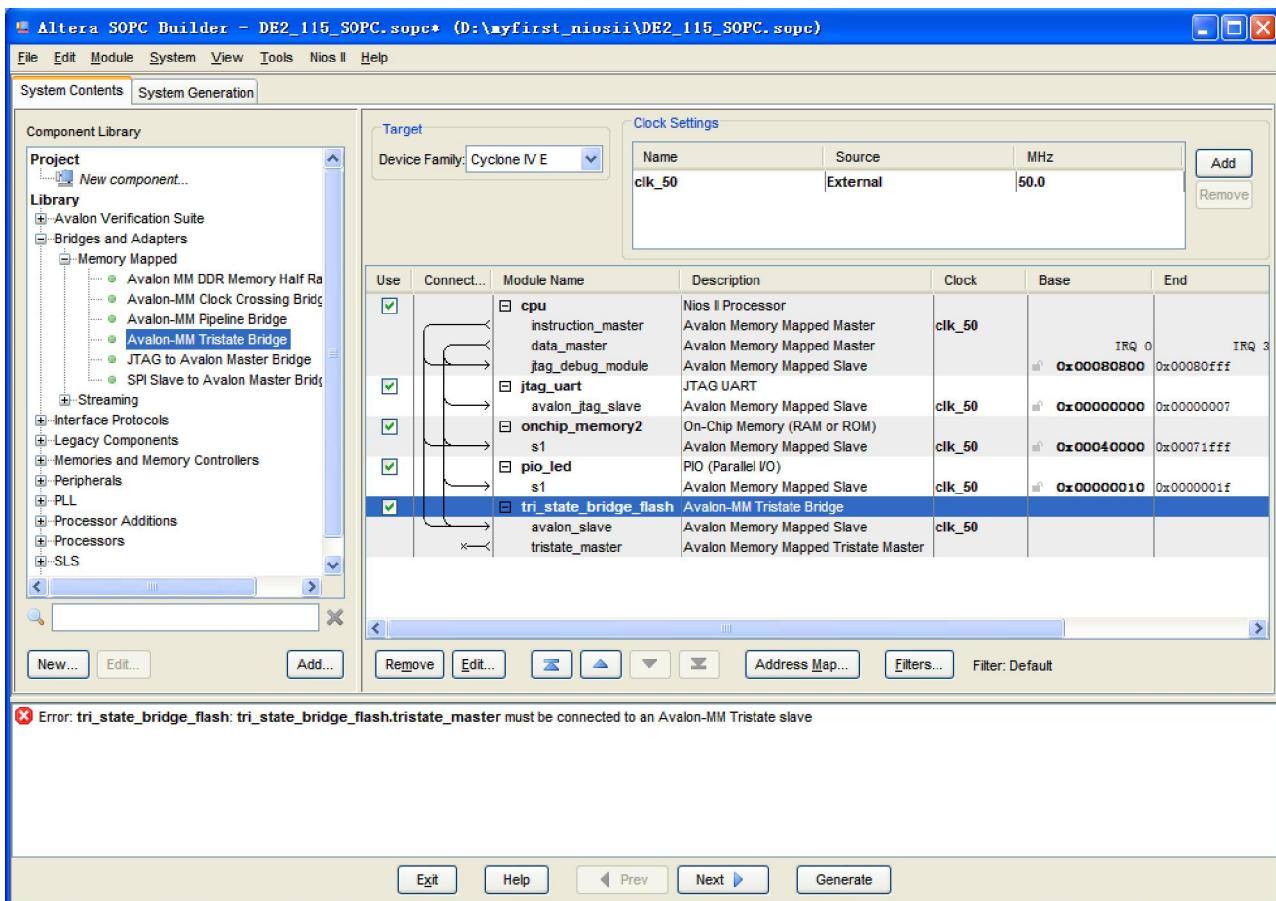


Figure 3-4 Rename Avalon-MM Tristate Bridge

4. Choose **Library > Memories and Memory Controllers > Flash > Flash Memory Interface (CFI)** to open Flash Memory Interface (CFI) wizard. See **Figure 3-5** and **Figure 3-6**.

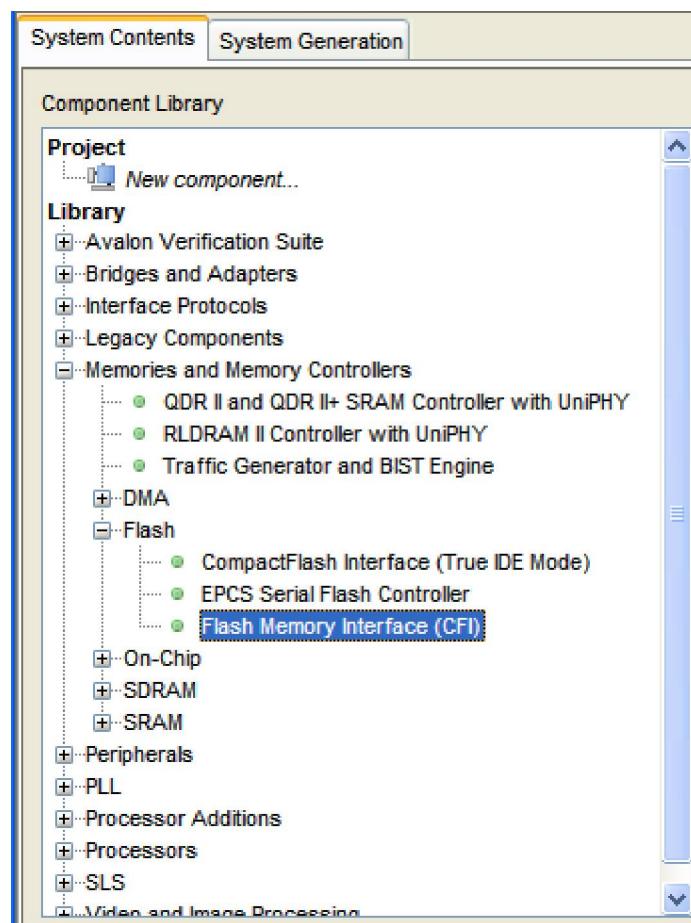


Figure 3-5 Flash Memory Interface (CFI)

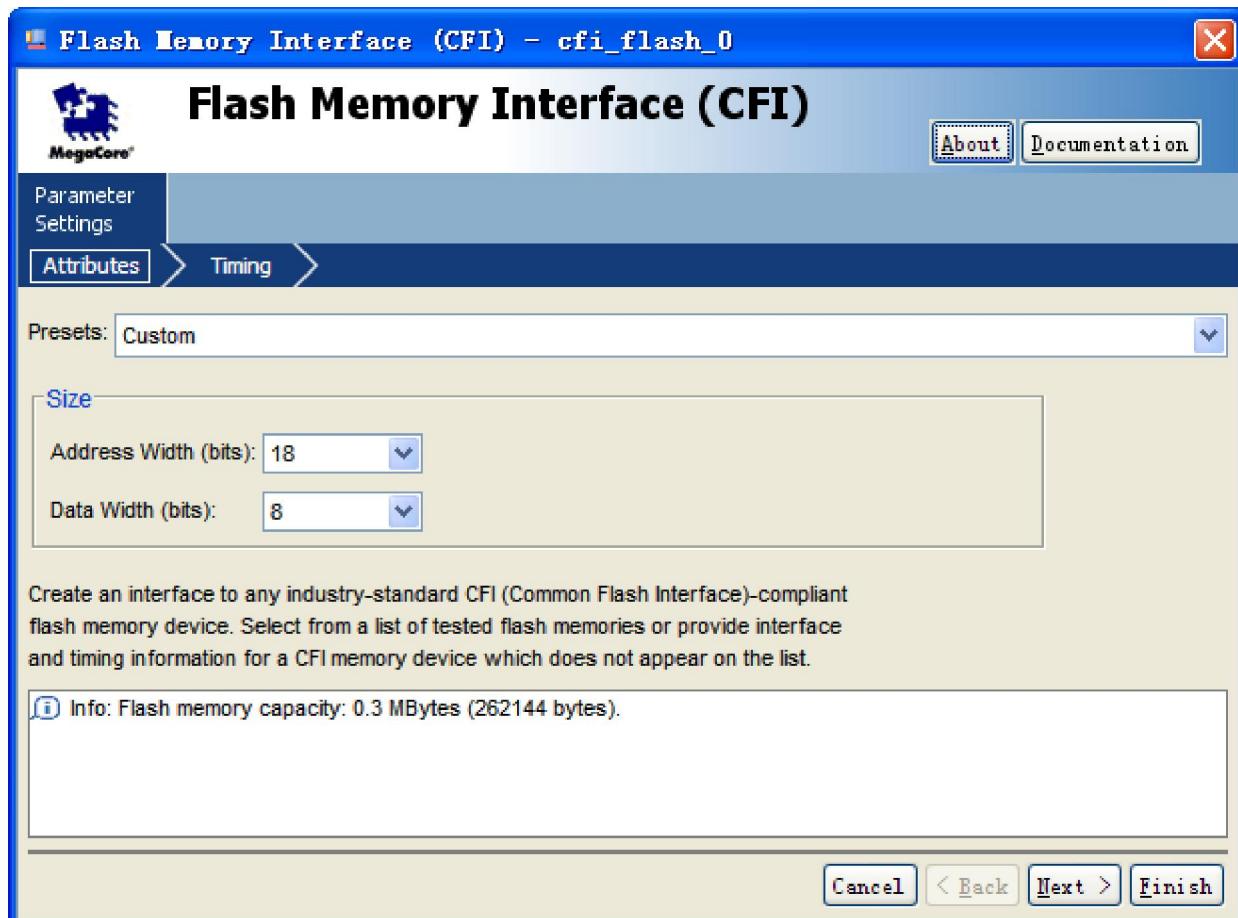


Figure 3-6 Flash Memory Interface (CFI) Box

5. Modify Address Width as shown in **Figure 3-7**.

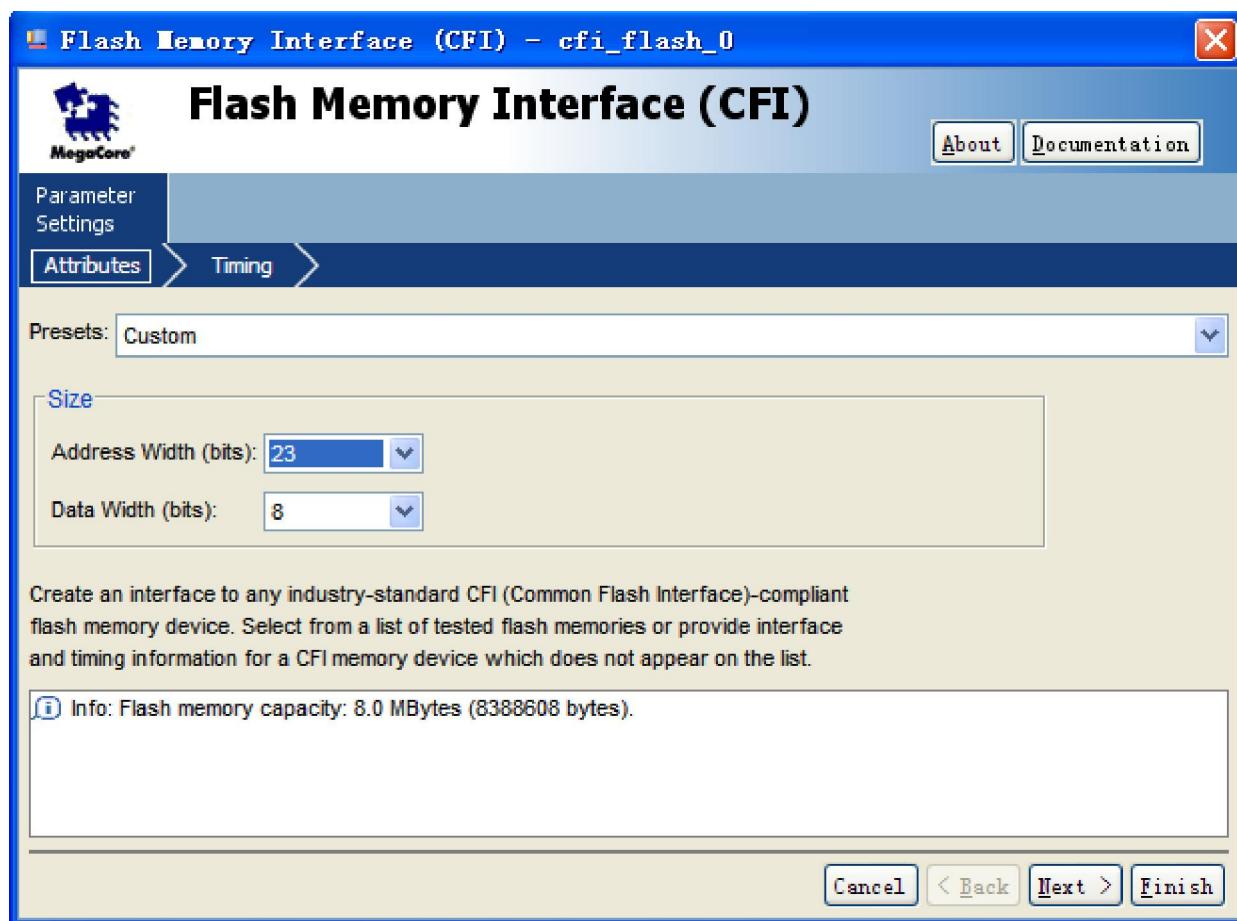


Figure 3-7 Modify Address Width in Flash Memory Interface (CFI)

6. Choose **Timing** tab, modify settings as shown in Figure 3-8.

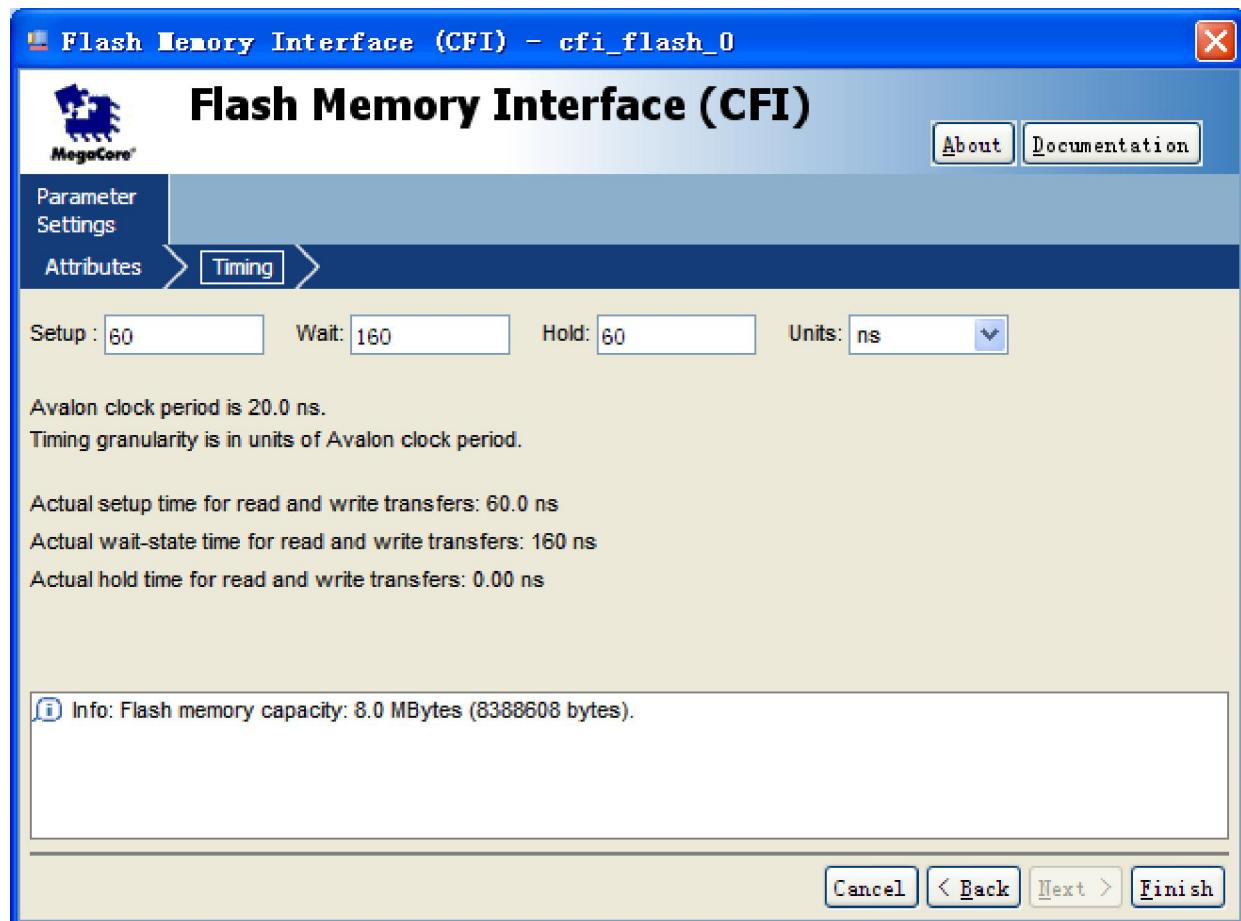


Figure 3-8 Timing Settings of CFI

7. Click **Finish** to close Flash Memory Interface (CFI) box, and return to the window as shown in

Figure 3-9.

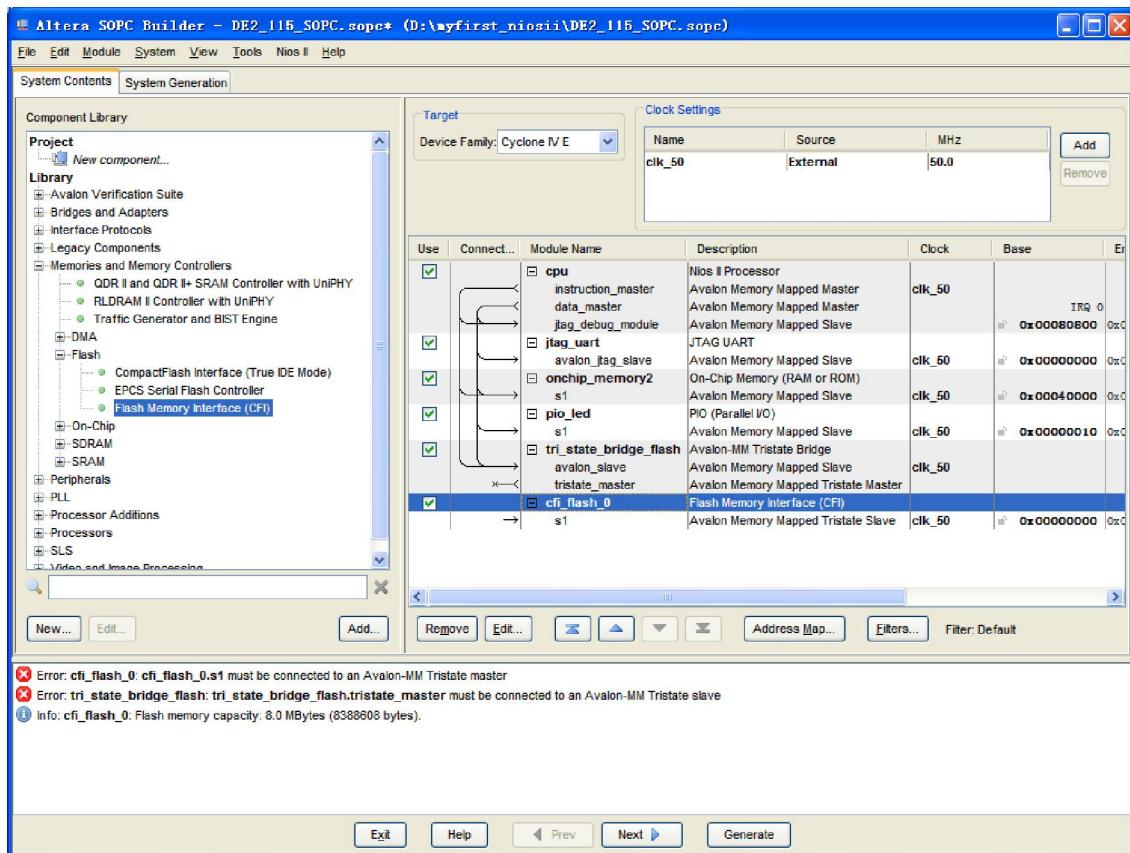


Figure 3-9 Add Flash Memory Interface (CFI) Component

8. Rename Flash Memory Interface (CFI) as shown in **Figure 3-10**.

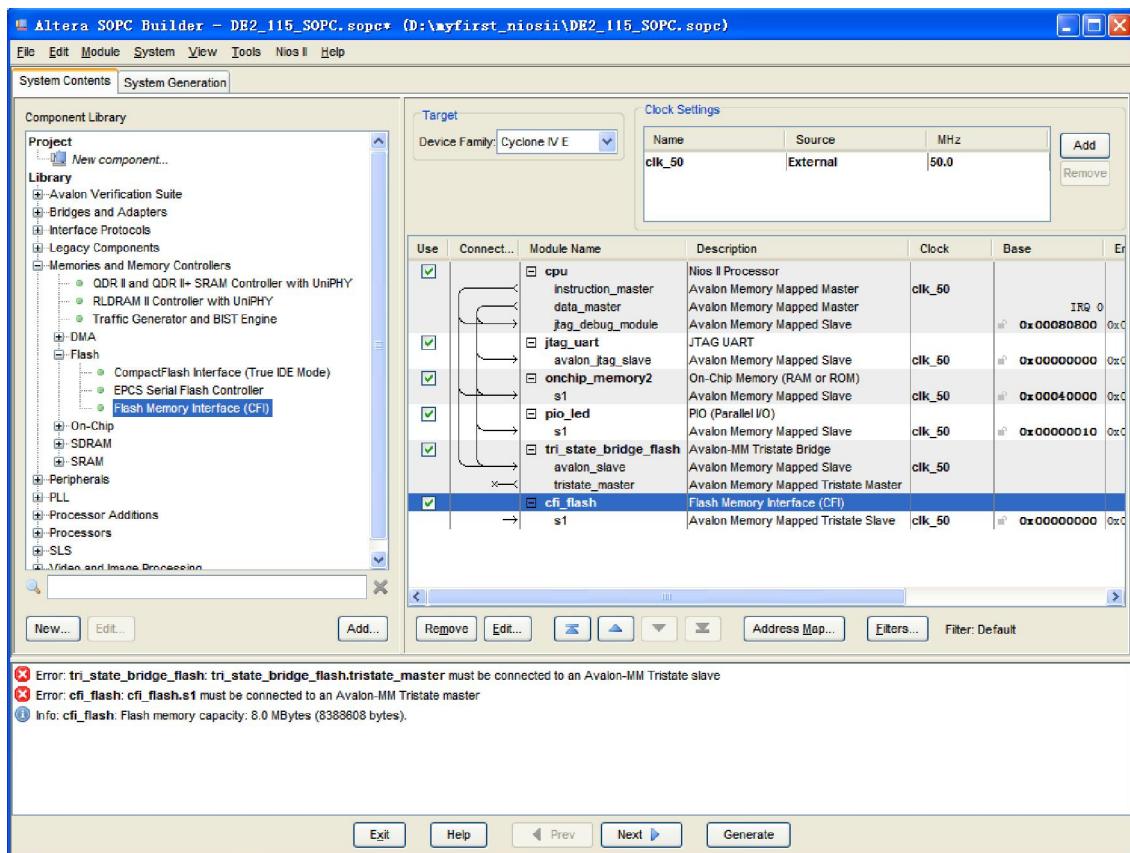


Figure 3-10 Rename Flash Memory Interface (CFI)

9. Link tri_state_bridge_flash to cfl_flash as shown in Figure 3-11.

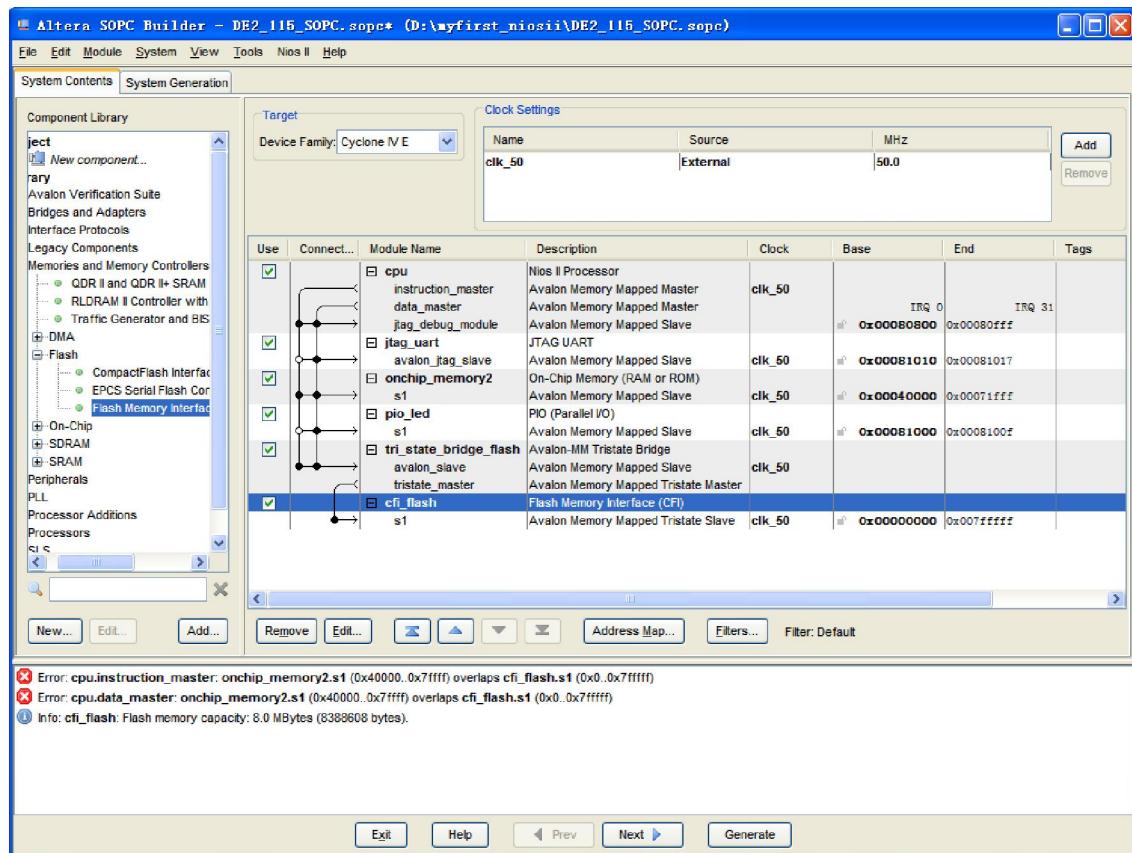


Figure 3-11 Link tri_state_bridge_flash to cfi_flash

10. Choose **tri_state_bridge_flash > Shared Signals tab**, and choose **data** box as shown in **Figure 3-12**.

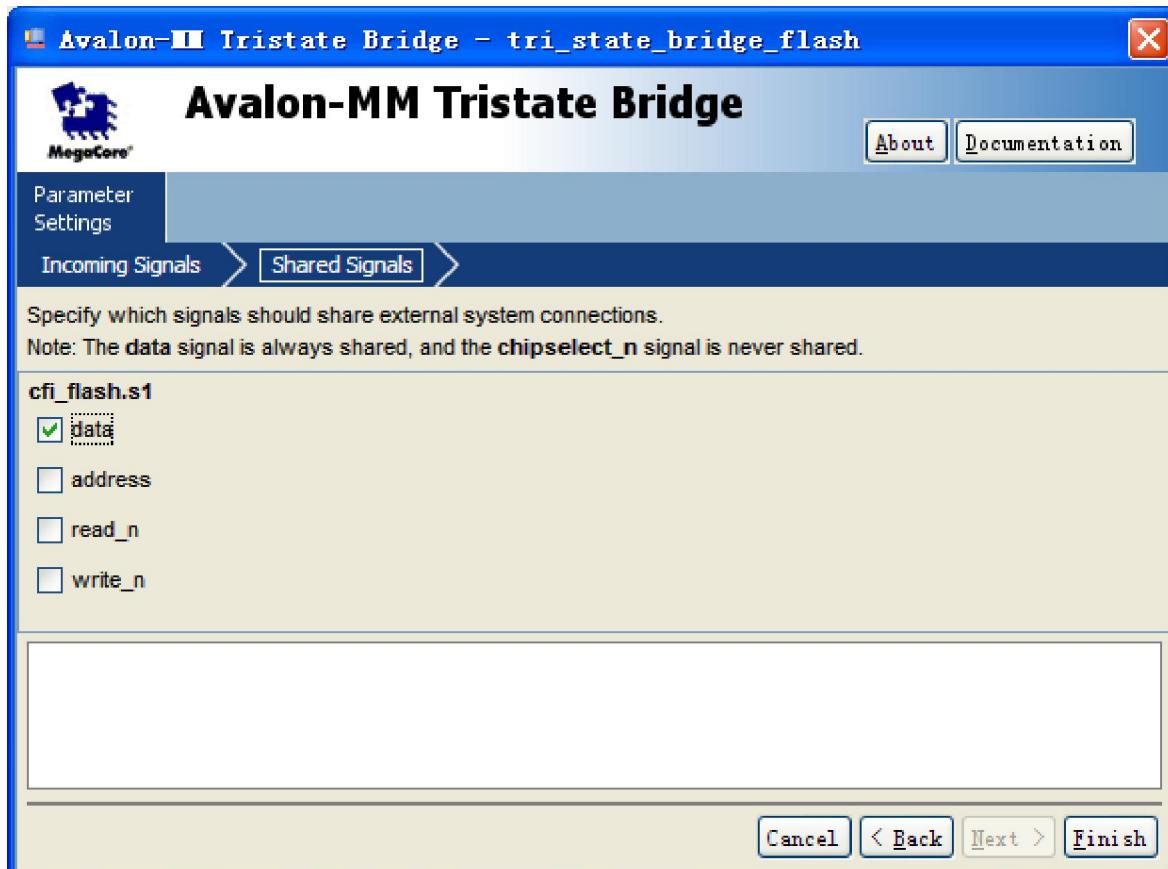


Figure 3-12 Choose data box

11. Click **Finish** to close the Avalon-MM Tristate Bridge Box. Choose **System > Auto-Assign Base Addresses**, then click **Generate** to generate the sopc.

3.2 Modify the myfirst_niosii.v

1. In the Quartus II, modify **myfirst_niosii.v** as below.

```
module myfirst_niosii
```

```
(
```

```
    CLOCK_50,
```

```
    LED,
```

```
    ////////////// Flash //////////////////
```

```
FL_ADDR,  
FL_CE_N,  
FL_DQ,  
FL_OE_N,  
FL_RESET_N,  
FL_RY,  
FL_WE_N,  
FL_WP_N,  
);  
  
input      CLOCK_50;  
  
output [7:0]  LED;  
  
////////// Flash //////////  
  
output [22:0]  FL_ADDR;  
  
output          FL_CE_N;  
  
inout [7:0]    FL_DQ;  
  
output          FL_OE_N;  
  
output          FL_RESET_N;  
  
input           FL_RY;  
  
output          FL_WE_N;  
  
output          FL_WP_N;  
  
DE2_115_SOPC DE2_115_SOPC_inst  
(
```

```
.clk_50          (CLOCK_50),  
.out_port_from_the_pio_led(LED),  
  
//flash  
.address_to_the_cfi_flash(FL_ADDR),  
.read_n_to_the_cfi_flash(FL_OE_N),  
.select_n_to_the_cfi_flash(FL_CE_N),  
.tri_state_bridge_flash_data(FL_DQ),  
.write_n_to_the_cfi_flash(FL_WE_N),  
  
.reset_n          (1'b1)  
// Flash Config  
assign FL_RESET_N = 1'b1;  
assign FL_WP_N = 1'b1;  
  
endmodule
```

2. Re-compilation myfirst_niosii project.

3.3 Re-assign pins

1. re-assign pins. The pins as shown in Table 3-1.

Node Name	Location
-----------	----------

CLOCK_50	PIN_Y2
FL_ADDR[0]	PIN_AG12
FL_ADDR[1]	PIN_AH7
FL_ADDR[2]	PIN_Y13
FL_ADDR[3]	PIN_Y14
FL_ADDR[4]	PIN_Y12
FL_ADDR[5]	PIN_AA13
FL_ADDR[6]	PIN_AA12
FL_ADDR[7]	PIN_AB13
FL_ADDR[8]	PIN_AB12
FL_ADDR[9]	PIN_AB10
FL_ADDR[10]	PIN_AE9
FL_ADDR[11]	PIN_AF9
FL_ADDR[12]	PIN_AA10
FL_ADDR[13]	PIN_AD8
FL_ADDR[14]	PIN_AC8
FL_ADDR[15]	PIN_Y10
FL_ADDR[16]	PIN_AA8
FL_ADDR[17]	PIN_AH12
FL_ADDR[18]	PIN_AC12
FL_ADDR[19]	PIN_AD12
FL_ADDR[20]	PIN_AE10
FL_ADDR[21]	PIN_AD10
FL_ADDR[22]	PIN_AD11
FL_CE_N	PIN_AG7
FL_DQ[0]	PIN_AH8
FL_DQ[1]	PIN_AF10
FL_DQ[2]	PIN_AG10
FL_DQ[3]	PIN_AH10
FL_DQ[4]	PIN_AF11
FL_DQ[5]	PIN_AG11
FL_DQ[6]	PIN_AH11
FL_DQ[7]	PIN_AF12
FL_OE_N	PIN_AG8
FL_RESET_N	PIN_AE11
FL_RY	PIN_Y1
FL_WE_N	PIN_AC10
FL_WP_N	PIN_AE12
LED[0]	PIN_E21
LED[1]	PIN_E22

LED[2]	PIN_E25
LED[3]	PIN_E24
LED[4]	PIN_H21
LED[5]	PIN_G20
LED[6]	PIN_G22
LED[7]	PIN_G21

Table 3-1 Pin assignment

2. Re-compilation myfirst_niosii project and re-download myfirst_niosii.sof to the development board.

3.4 Re-Configure System Library

1. In the Nios II IDE, right-click **hello_world_0** and choose **System Library Properties**. The **Properties for hello_world_0_syslib** dialog box opens.
2. Choose **cfi_flash** for **Program Memory** and **Read-only data memory**. See Figure 3-13.

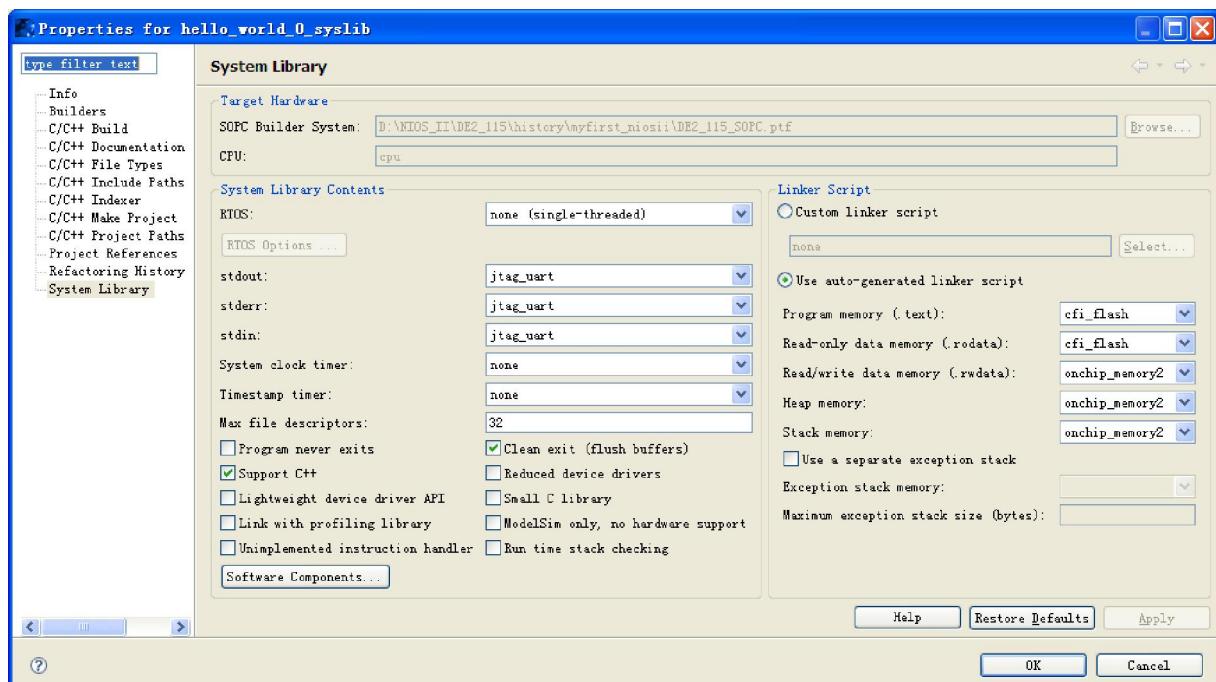


Figure 3-13 System Library

3. Click **OK** to close the **Properties for hello_world_0_syslib** box and re-build hello_world_0 project.

3.5 Programming the CFI Flash

1. Choose **Tools > Flash Programmer...** to open Flash Programmer box. See **Figure 3-14**.

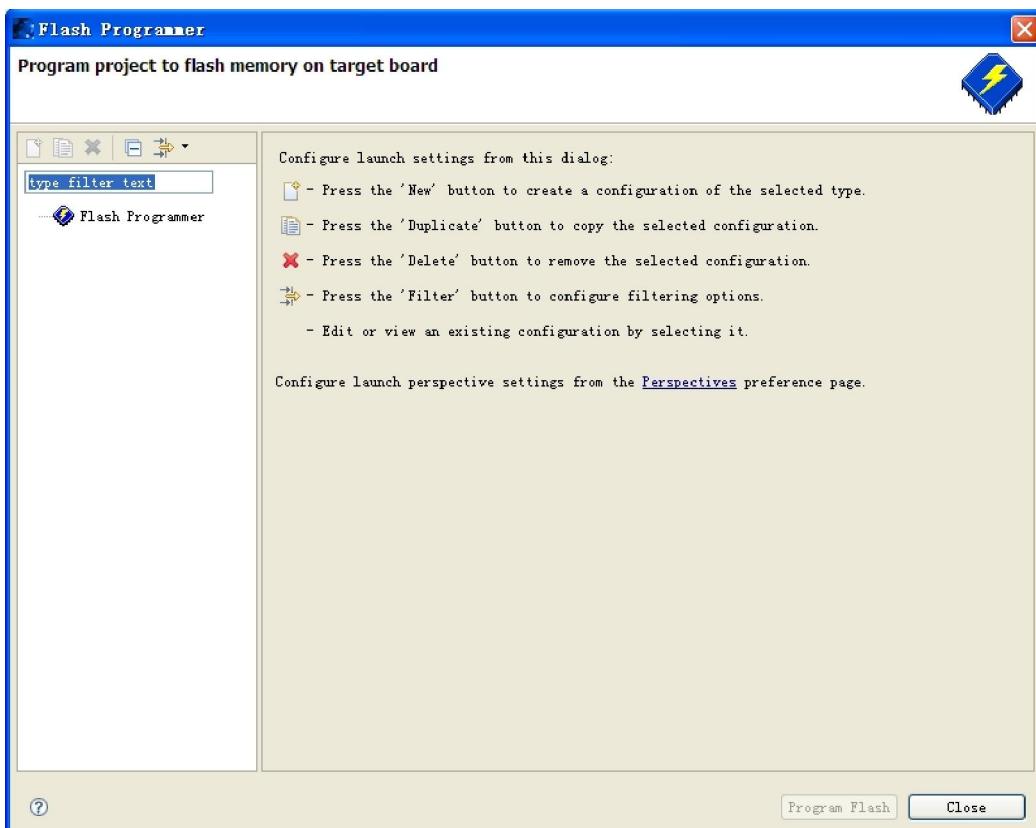


Figure 3-14 Flash Programmer Box

2. Double-click **Flash Programmer**, the flash programmer appears. Make sure your settings are the same as shown in **Figure 3-15**.

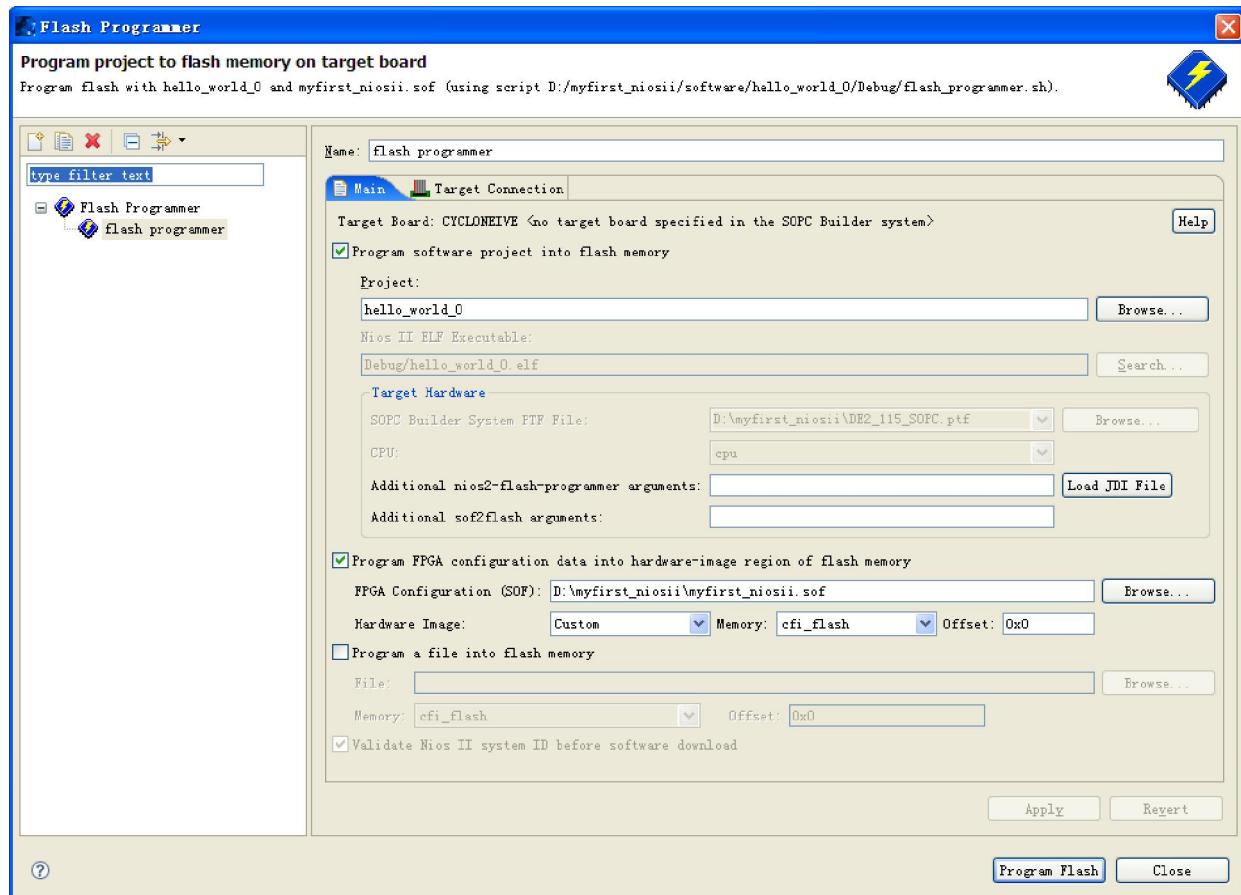


Figure 3-15 flash programmer settings

3. Click **Program Flash**. If the “Program Flash Now?” message box appears, click **Yes**.

See **Figure 3-16**.

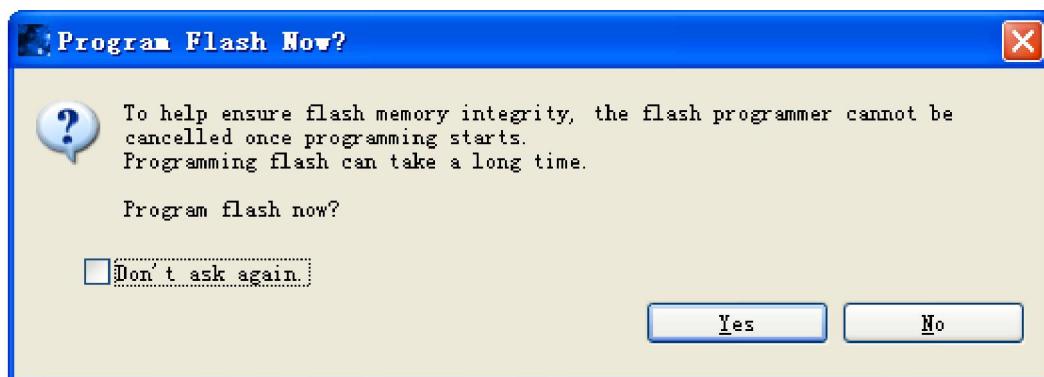


Figure 3-16 Program Flash Message Box

4. When program flash completely, the console tab displays as shown in **Figure 3-17**.

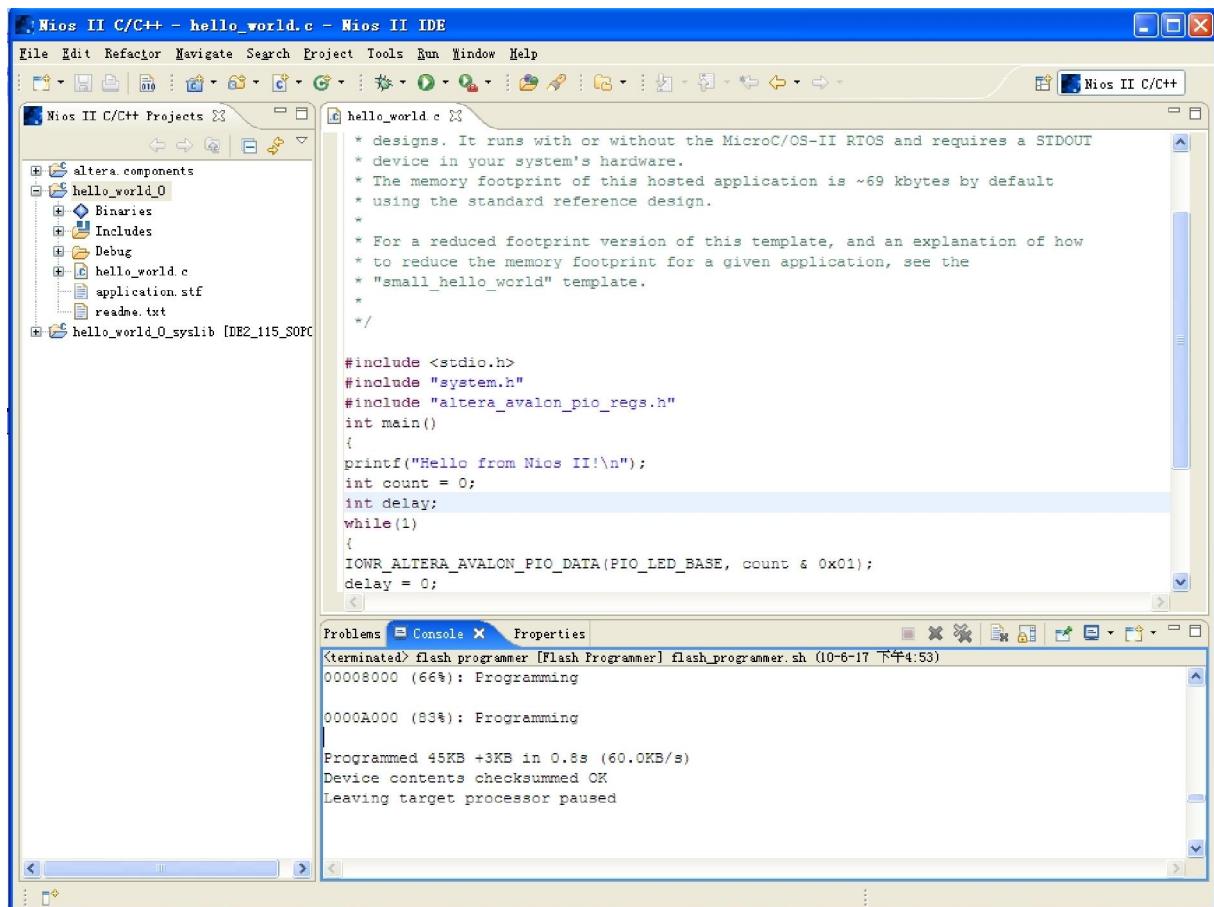


Figure 3-17 Program Flash completely

5. Restart power on the board. Download myfirst_niosii.sof of your project “myfirst_niosii”. You will see that the led blinks.