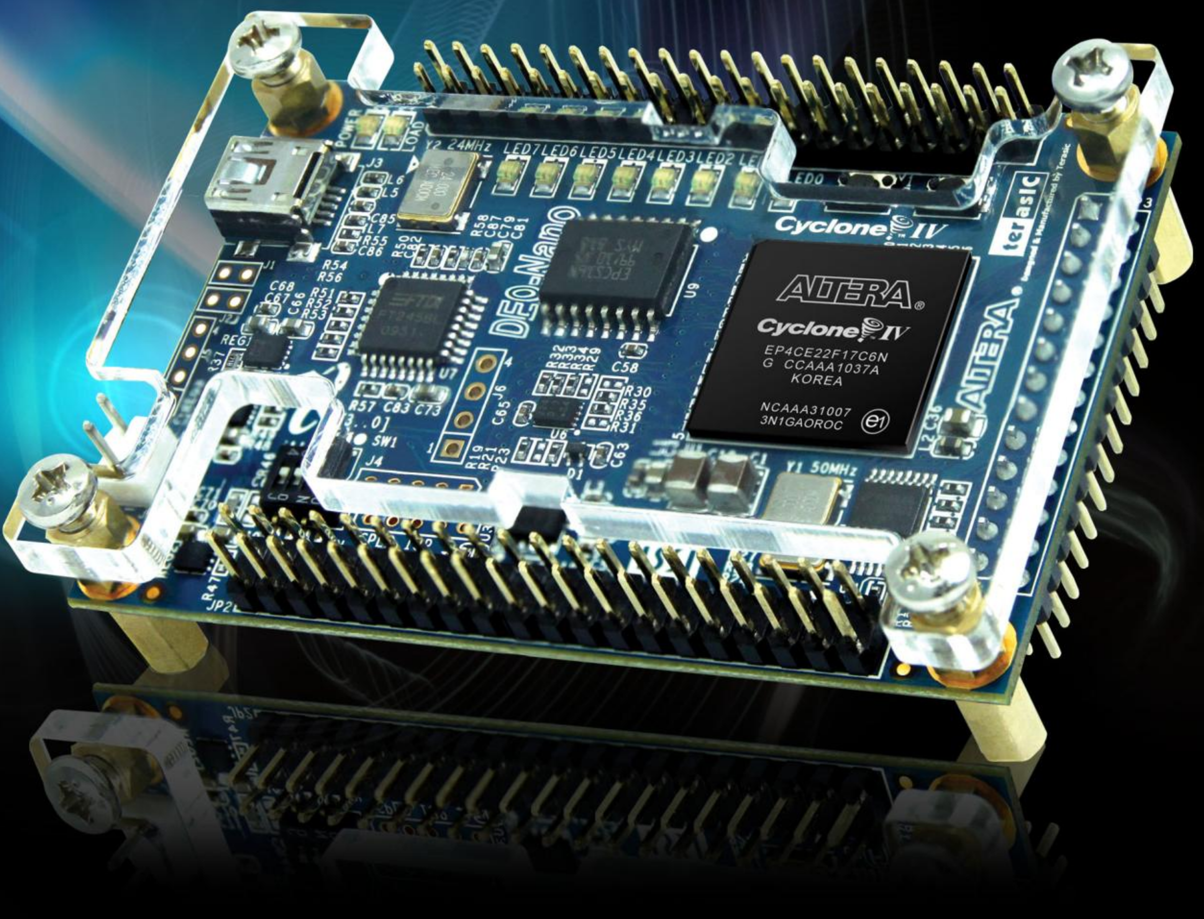
DE0-Nano

User Manual

World Leading FPGA Based Products and Design Services



www.terasic.com

CONTENTS

Chapter 1

Introduction of the DE0-Nano Board

The DE0-Nano board introduces a compact-sized FPGA development platform suited for prototyping circuit designs such as robots and mobile projects. The board is designed to be used in the simplest possible implementation targeting the Cyclone IV device up to 22,320 LEs.

The DE0-Nano has a collection of interfaces including two external GPIO headers to extend designs beyond the DE0-Nano board, on-board memory devices including SDRAM and EEPROM for larger data storage and frame buffering, as well as general user peripheral with LEDs and push-buttons.

The advantages of the DE0-Nano board include its size and weight, as well as its ability to be reconfigured without carrying superfluous hardware, setting itself apart from other general purpose development boards. In addition, for mobile designs where portable power is crucial, the DE0-Nano provides designers with three power scheme options including a USB mini-AB port, 2-pin external power header and two DC 5V pins on the GPIO headers.

1.1 Features

Figure 1-1 shows a photograph of the DE0-Nano Board.

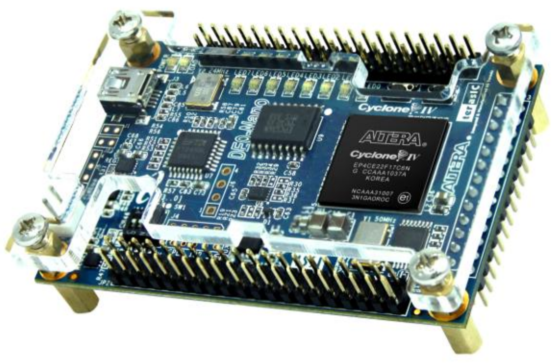


Figure 1-1 Layout of the DE0-Nano Board

The key features of the board are listed below:

- Featured device
  - Altera Cyclone® IV EP4CE22F17C6N FPGA
  - 153 maximum FPGA I/O pins

- Configuration status and set-up elements
  - On-board USB-Blaster circuit for programming
  - Altera serial configuration device – EPCS16

- Expansion header
  - Two 40-pin Headers (GPIOs) provides 72 I/O pins
  - Two 5V power pins, two 3.3V power pins and four ground pins

- Memory devices
  - 32MB SDRAM
  - 2Kb I2C EEPROM

- General user input/output
  - 8 green LEDs
  - 2 debounced push-buttons
  - 4-position DIP switch

- G-Sensor
  - ADI ADXL345, 3-axis accelerometer with high resolution (13-bit)

- A/D Converter
  - NS ADC128S022, 8-Channel, 12-bit A/D Converter
  - 50 Ksps to 200 Ksps

- Clock system
  - On-board 50MHz clock oscillator

- Power Supply
  - USB Type mini-AB port (5V)
  - DC 5V pin for each GPIO header (2 DC 5V pins)
  - 2-pin external power header (3.6-5.7V)

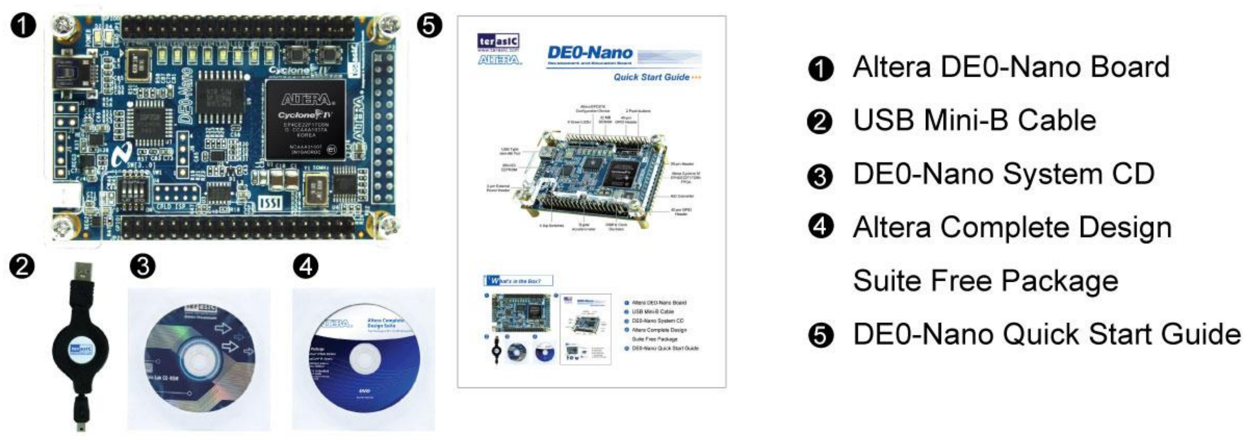
1.2 About the KIT

The kit will come with the following contents:

- DE0-Nano board
- System CD-ROM.

The system CD contains technical documents of the DE0-Nano board, which includes component datasheets, demonstrations, schematic, and user manual.

Figure 1-2 shows the photograph of the DE0-Nano kit contents.



❶ Altera DE0-Nano Board
❷ USB Mini-B Cable
❸ DE0-Nano System CD
❹ Altera Complete Design Suite Free Package
❺ DE0-Nano Quick Start Guide

Figure 1-2 DE0-Nano kit package contents

1.3 Getting Help

Here is information of how to get help if you encounter any problem:

- Terasic Technologies
- Tel: +886-3-550-8800
- Email: support@terasic.com

Chapter 2

DE0-Nano Board Architecture

This chapter describes the architecture of the DE0-Nano board including block diagram and components.

2.1 Layout and Components

The picture of the DE0-Nano board is shown in **Figure 2-1** and **Figure 2-2**. It depicts the layout of the board and indicates the locations of the connectors and key components.

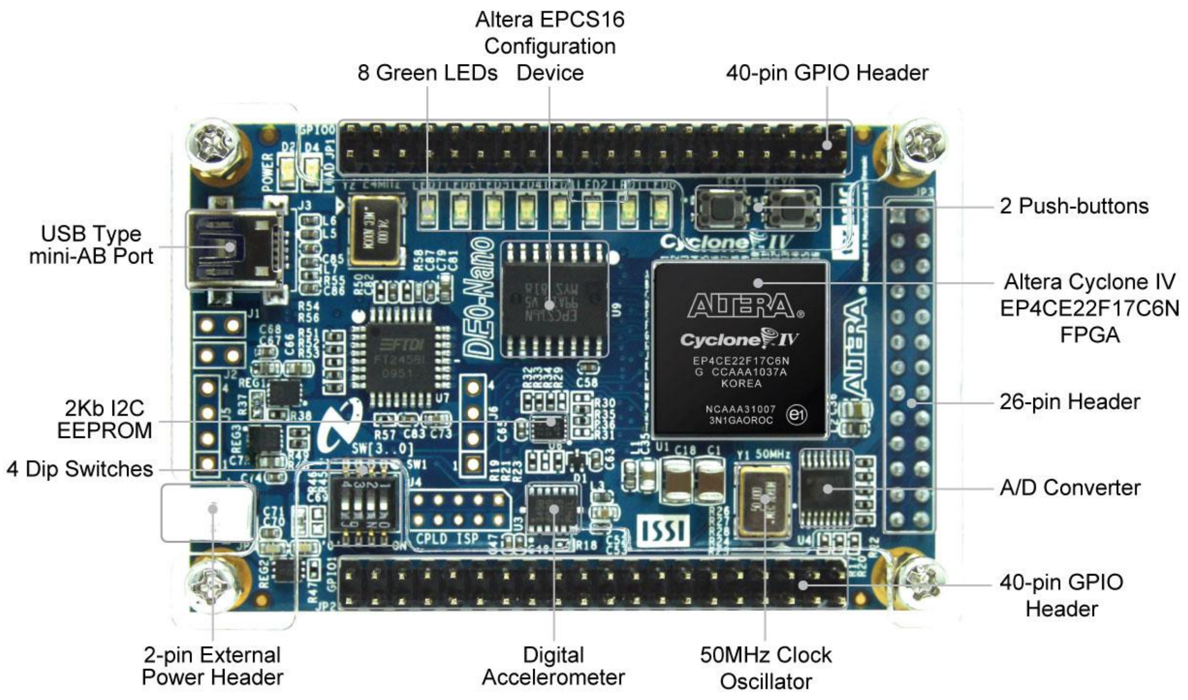


Figure 2-1 The DE0-Nano Board PCB and component diagram (top view)

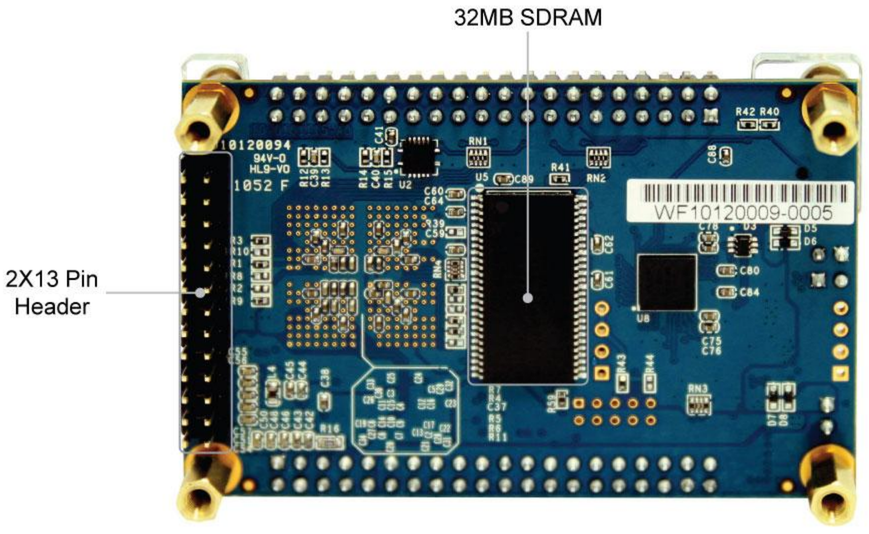


Figure 2-2 The DE0-Nano Board PCB and component diagram (bottom view)

2.2 Block Diagram of the DE0-Nano Board

Figure 2-3 shows the block diagram of the DE0-Nano board. To provide maximum flexibility for the user, all connections are made through the Cyclone IV FPGA device. Thus, the user can configure the FPGA to implement any system design.

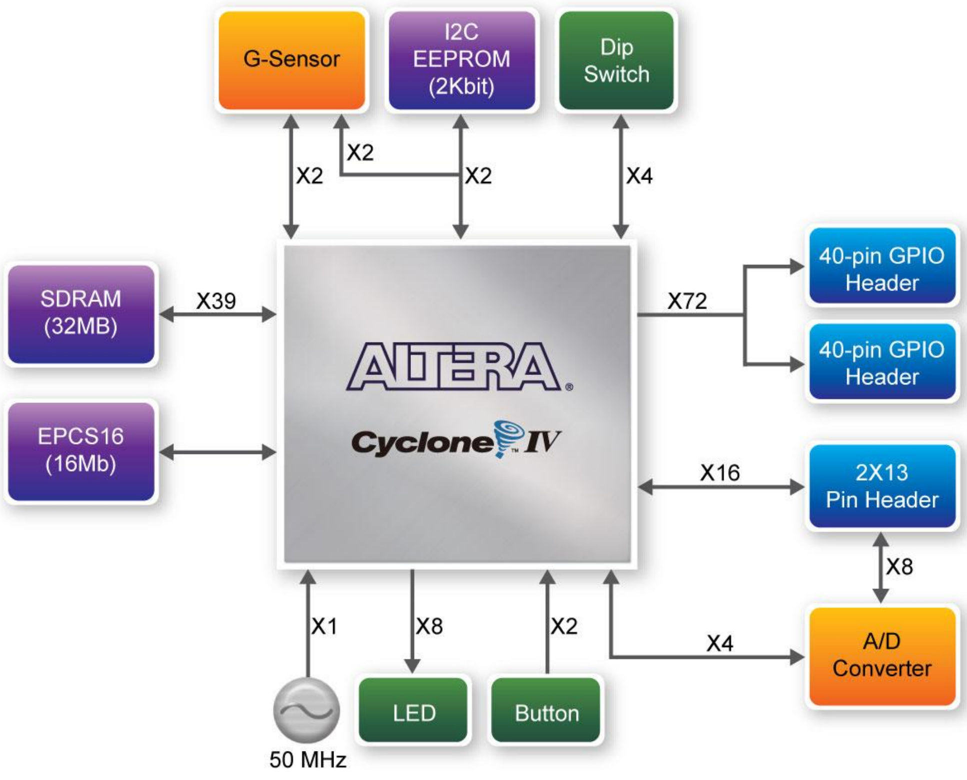


Figure 2-3 Block diagram of DE0-Nano Board

2.3 Power-up the DE0-Nano Board

The DE0-Nano board comes with a preloaded configuration bit stream to demonstrate the features of the board. This bit stream also allows users to see quickly if the board is working properly. To power-up the board two options are available which are described below:

1. Connect a USB Mini-B cable to a USB (Type A) host port and to the board.
2. Alternatively, users can power-up the DE0-Nano board by supplying 5V to the two DC +5 (VCC5) pins of the GPIO headers or supplying (3.6-5.7V) to the 2-pin header. For communication between the host and the DE0-Nano board, it is necessary to install the Altera USB Blaster driver software.

At this point you should observe the user LEDs flashing.

Chapter 3

Using the DE0-Nano Board

This section describes the detailed information of the components, connectors, and pin assignments of the DE0-Nano board.

3.1 Configuring the Cyclone IV FPGA

The DE0-Nano board contains a serial configuration device that stores configuration data for the Cyclone IV FPGA. This configuration data is automatically loaded from the configuration device into the FPGA every time while power is applied to the board. Using the Quartus II software, it is possible to reconfigure the FPGA at any time, and it is also possible to change the non-volatile data that is stored in the serial configuration device. Both types of programming methods are described below.

1. JTAG programming: In this method of programming, named after the IEEE standards Joint Test Action Group, the configuration bit stream is downloaded directly into the Cyclone IV FPGA. The FPGA will retain this configuration as long as power is applied to the board; the configuration information will be lost when the power is turned off.
2. Programming the serial configuration device: In this method, the configuration bit stream is downloaded into the Altera EPCS16 serial configuration device. It provides non-volatile storage of the bit stream, so that the information is retained even when the power supply to the DE0-Nano board is turned off. When the board's power is turned on, the configuration data in the EPCS16 device is automatically loaded into the Cyclone IV FPGA.

■ JTAG Chain on DE0-Nano Board

The USB-blaster is implemented on the DE0-Nano board to provide JTAG configuration through onboard USB-to-JTAG configuration logic using a USB Mini-B cable, a FTDI USB 2.0 Controller, and an Altera MAX II CPLD. Current configuration will be lost when power is turned off.

Figure 3-1 illustrates the JTAG configuration setup.

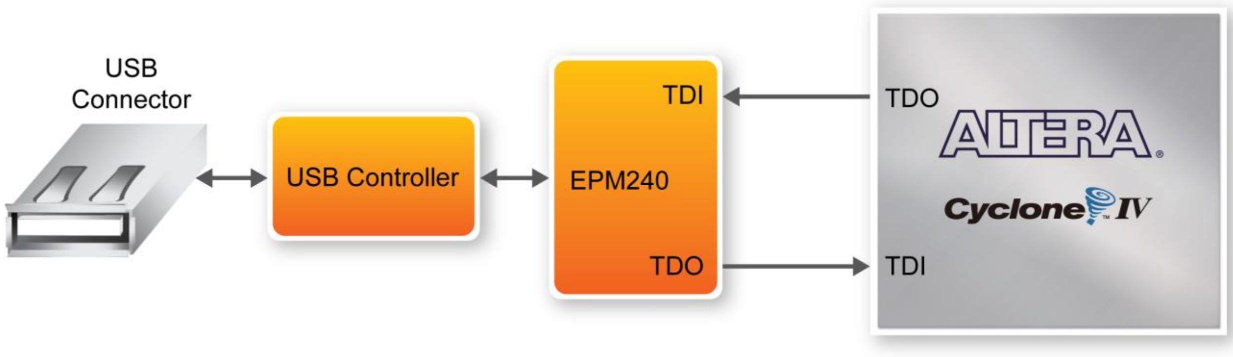


Figure 3-1 JTAG Chain

To download a configuration bit stream file into the Cyclone IV FPGA, perform the following steps:

- Connect a USB Mini-B cable to the Mini-USB Type-AB connector on the DE0-Nano.
- The FPGA can now be programmed through the Quartus II Programmer by selecting a configuration bit stream file with the .sof filename extension.

■ Configuring the EPCS16

The DE0-Nano board contains an EPCS16 serial configuration device that stores configuration data for the Cyclone IV device. The Cyclone IV device support in-system programming of a serial configuration device using the JTAG interface via the serial flash loader design. The serial flash loader is a bridge design for the Cyclone IV device that uses its JTAG interface to access the EPCS .jic file and then uses the AS interface to program the EPCS device. Figure 3-2 illustrates the programming method when adopting a serial flash loader solution.

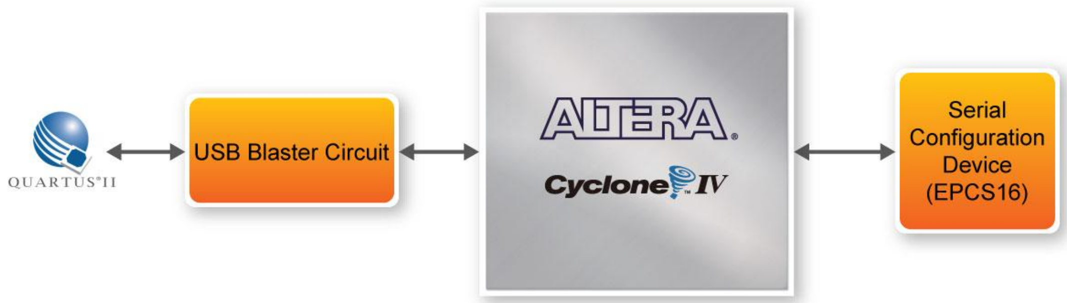


Figure 3-2 Programming a serial configuration device with serial flash loader solution

3.2 General User Input/Output

■ Push-buttons

The DE0-Nano board contains two push-buttons shown in **Figure 3-3** that allow you to interact with the Cyclone IV device. Each of these switches is debounced using a Schmitt Trigger circuit, as indicated in **Figure 3-4**. Each push-button provides a high logic level or a low logic level when it is not pressed or pressed, respectively.

A CPU reset push-button (KEY0) is an input to the Cyclone IV device. It is intended to be the master reset signal for FPGA designs loaded into the Cyclone IV device.

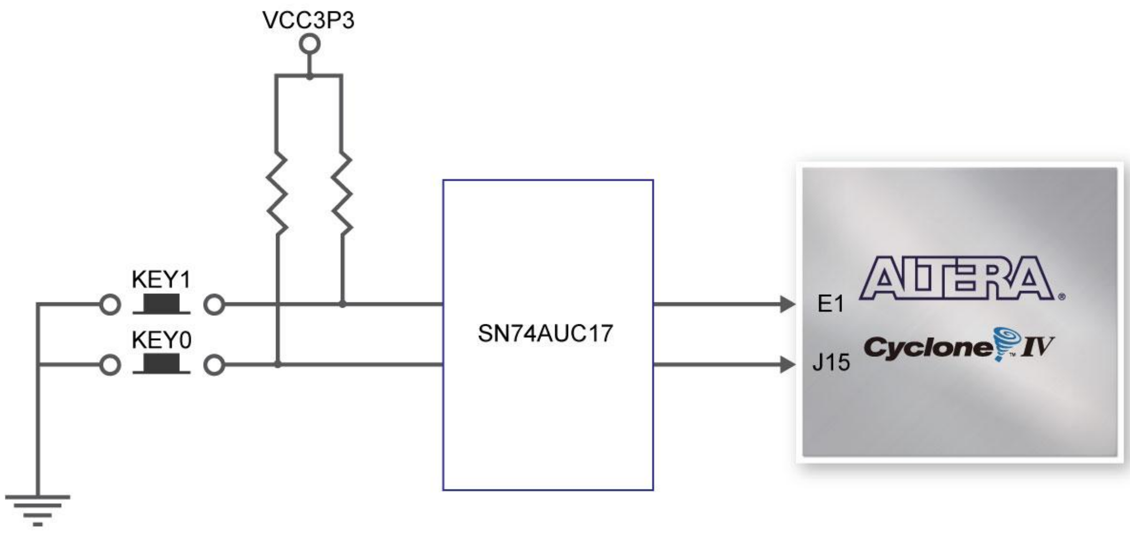


Figure 3-3 Connections between the push-buttons and Cyclone IV FPGA

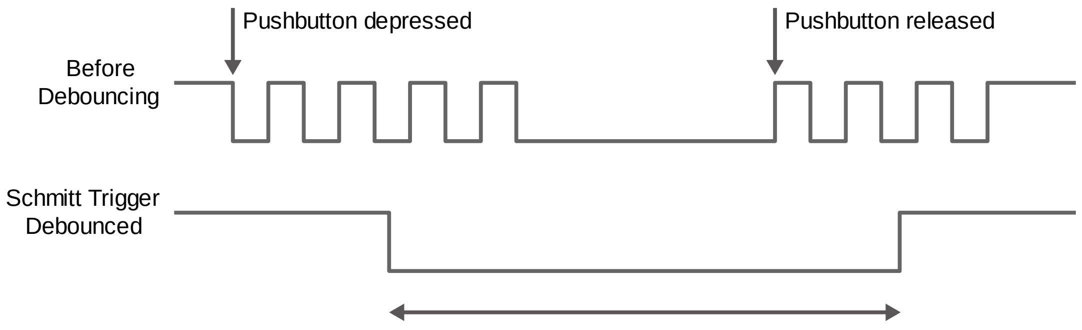


Figure 3-4 Push-buttons debouncing

■ LEDs

The DE0-Nano board consists of 8 green user-controllable LEDs presented in **Figure 3-5** to allow status and debugging signals to be driven to the LEDs from the designs loaded into the Cyclone IV device. Each LED is driven directly by the Cyclone IV FPGA. The LED is turn on and off when the associated pins are driven to a high and low logic level, respectively.

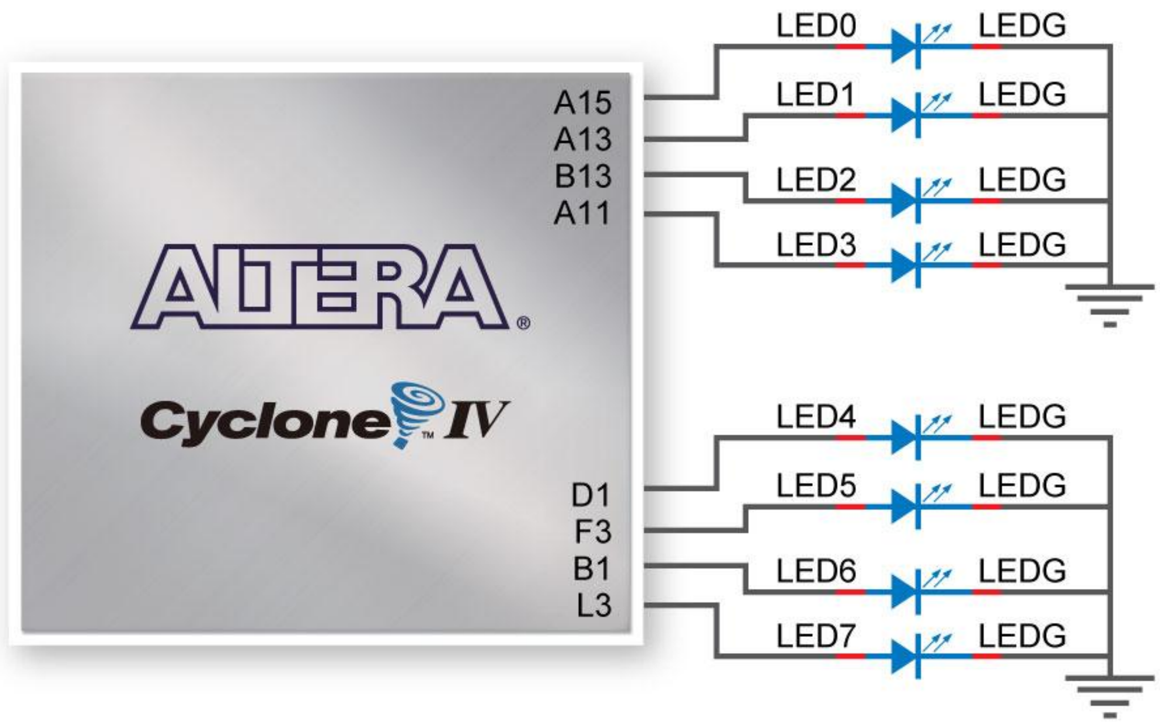


Figure 3-5 Connections between the LEDs and Cyclone IV FPGA

■ DIP Switch

The DE0-Nano board contains a 4 dip switches. When the switch is in the DOWN position or the UPPER position, it provides a high logic level or a low logic level to the FPGA.

3.3 SDRAM Memory

The board features a Synchronous Dynamic Random Access Memory (SDRAM) device providing 32MB with a 16-bit data lines connected to the FPGA. The chip uses 3.3V LVCMOS signaling standard. All signals are registered on the positive edge of the clock signal, CLK. **Figure 3-6** depicts its connection with the Cyclone IV FPGA.

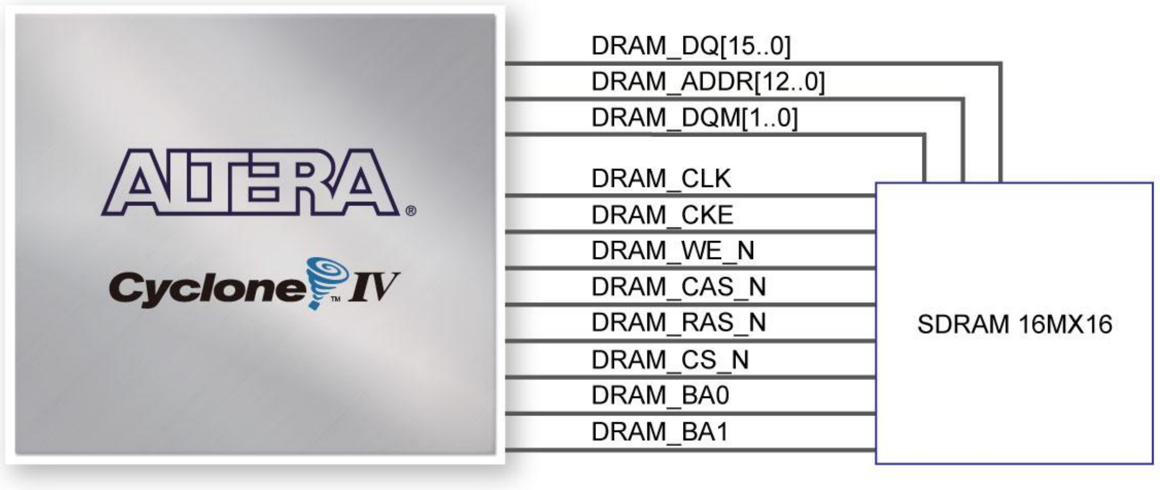


Figure 3-6 Connections between FPGA and SDRAM

3.4 I2C Serial EEPROM

A 2Kbit Electrically Erasable PROM (EEPROM) is equipped on the DE0-Nano which is configured through a 2-wire serial interface. The device is organized as one block of 256 x 8-bit memory. The I2C write and read address are 0xA0 and 0xA1, respectively. Figure 3-7 illustrates its connection with the Cyclone IV FPGA.

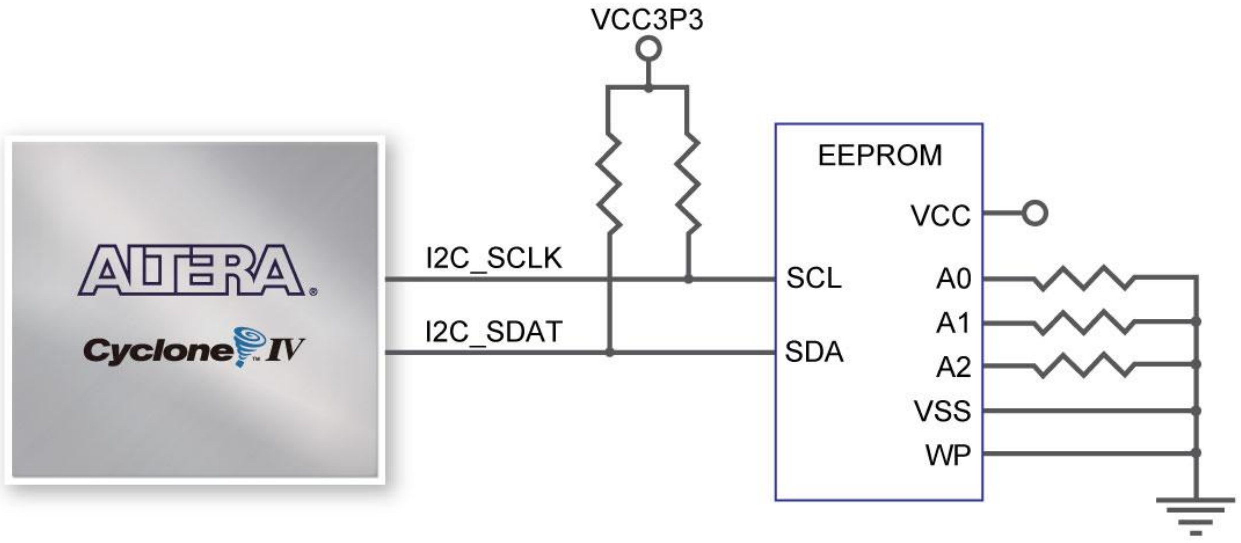


Figure 3-7 Connections between FPGA and EEPROM

3.5 Expansion Headers

The DE0-Nano board consists of two 40-pin expansion headers in **Figure 3-8** that offers additional connectivity and I/Os for general purpose applications. The headers connect directly to 72 pins of the Cyclone IV FPGA, and also provides two DC +5 (VCC_SYS), two DC +3.3V (VCC3P3), and four GND pins.

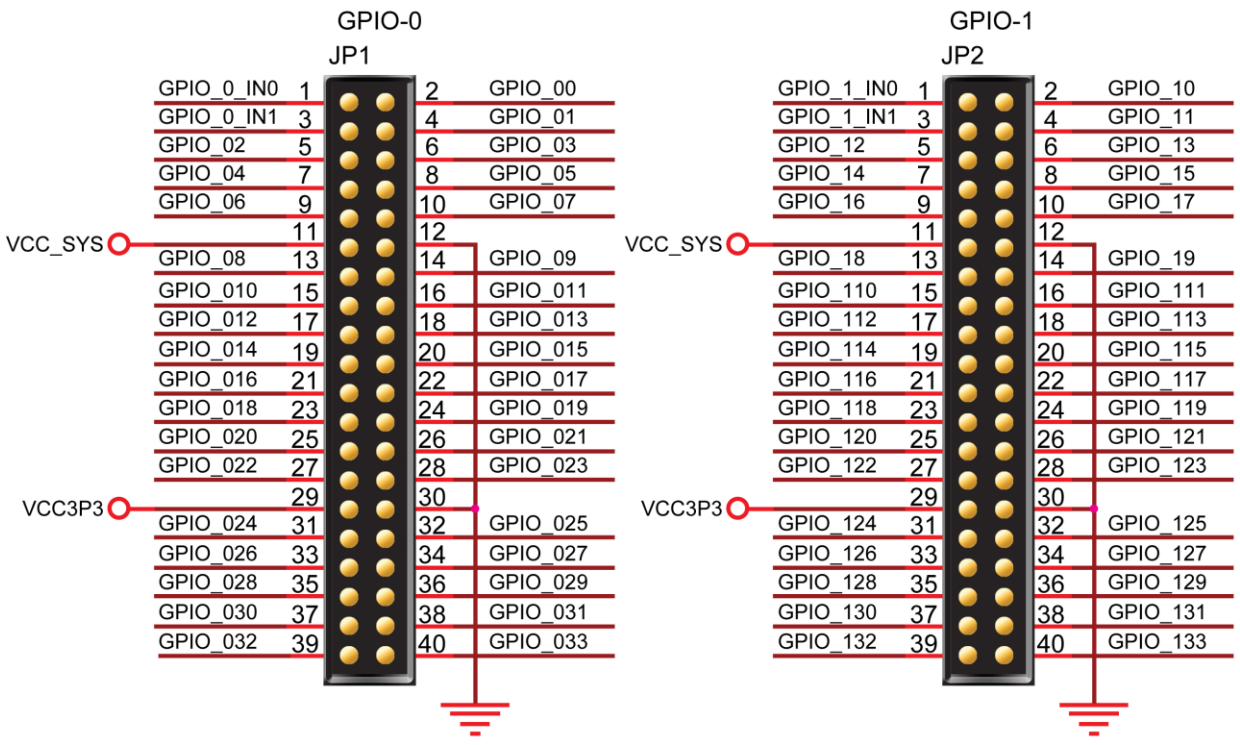


Figure 3-8 Pin distribution of the GPIO expansion headers

3.6 A/D Converter using 2x13 Header

The DE0-Nano contains an ADC128S022 lower power, eight-channel CMOS 12-bit analog-to-digital converter specified for conversion throughput rates of 50 ksps to 200 ksps. It can be configured to accept up to eight input signals at inputs IN0 through IN7 located at the 2x13 header shown in **Figure 3-9**. The I/Os of the 2x13 header are split into 2 banks, one being the analog input signals (IN0-IN7) and remaining I/Os are connected directly to the Cyclone IV device. For more detailed information on the A/D converter chip, please refer to its datasheet which is available on manufacturer's website or under the /datasheet folder of the system CD.

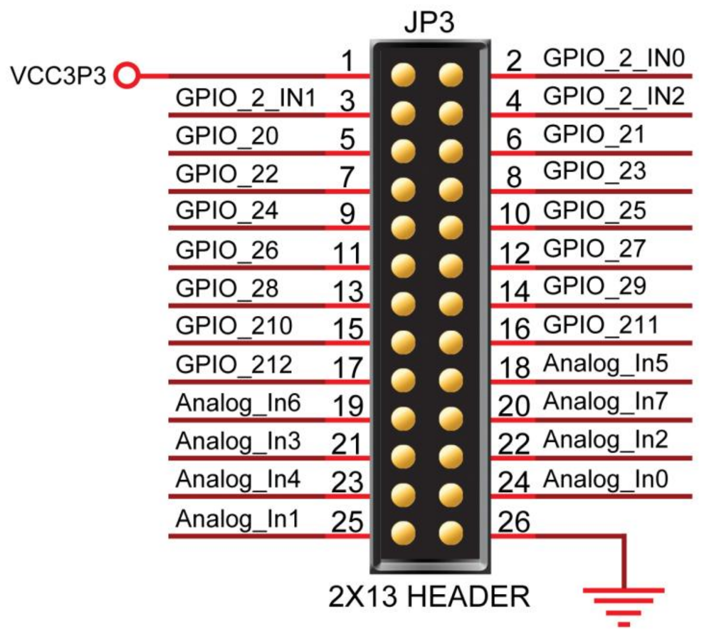


Figure 3-9　Pin distribution of the 2x13 Header

Figure 3-10 shows the connections on the 2x13 header, A/D converter and Cyclone IV device.

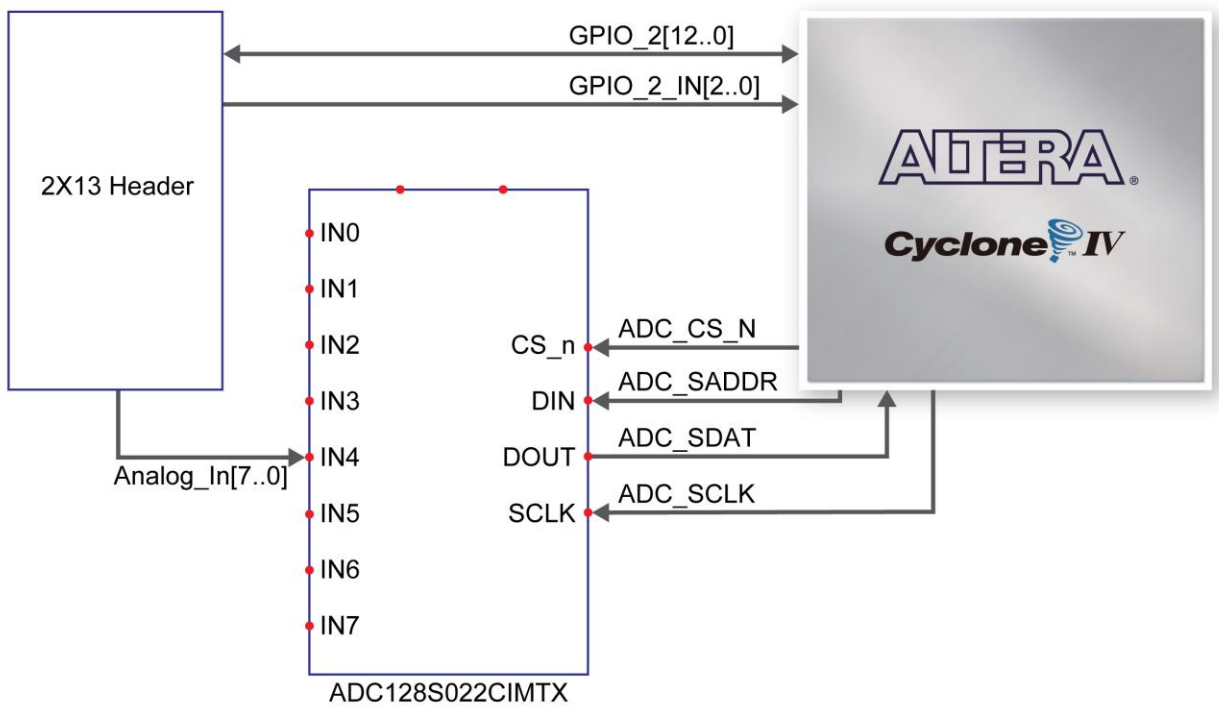


Figure 3-10　Wiring for 2x13 header and A/D converter

3.7 Digital Accelerometer

The ADXL345 is a small, thin, ultralow power consumption 3-axis accelerometer with high resolution measurement. The digital accelerometer is accessed through a SPI 3-wire digital interface. Digitalized output is formatted as 16-bit twos complement and can be accessed either using SPI interfac. This chip uses the 3.3V CMOS signaling standard. Main applications include medical instrumentation, industrial instrumentation, personal electronic aid and hard disk drive protection etc. Some of the key features of this device are listed below. For more detailed information of better using this chip, please refer to its datasheet which is available on manufacturer's website or under the /datasheet folder of the system CD.

- Up to 13-bit resolution at +/- 16g
- SPI (3- wire) digital interface
- Flexible interrupts modes

Figure 3-11 shows the connections between ADXL345 and Cyclone IV device.

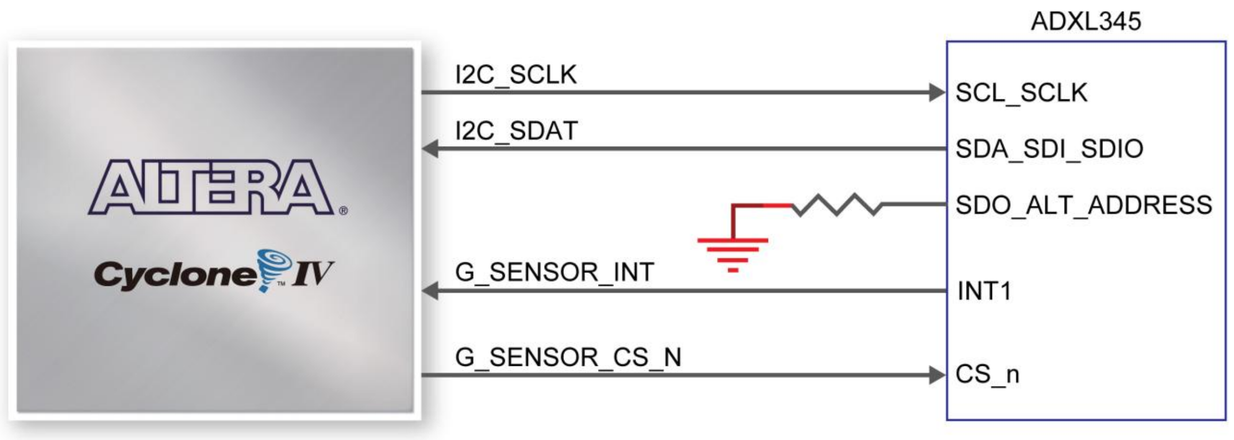


Figure 3-11 Wiring between the ADXL345 and Cyclone IV device

3.8 Clock Circuitry

The DE0-Nano board consists of a 50 MHz oscillator clock connected directly to the dedicated clock input pin of the Cyclone IV FPGA. The 50MHz clock input can be use as a source clock to drive the PLL circuit. The clock distribution on the DE0-Nano board is shown in Figure 3-12.

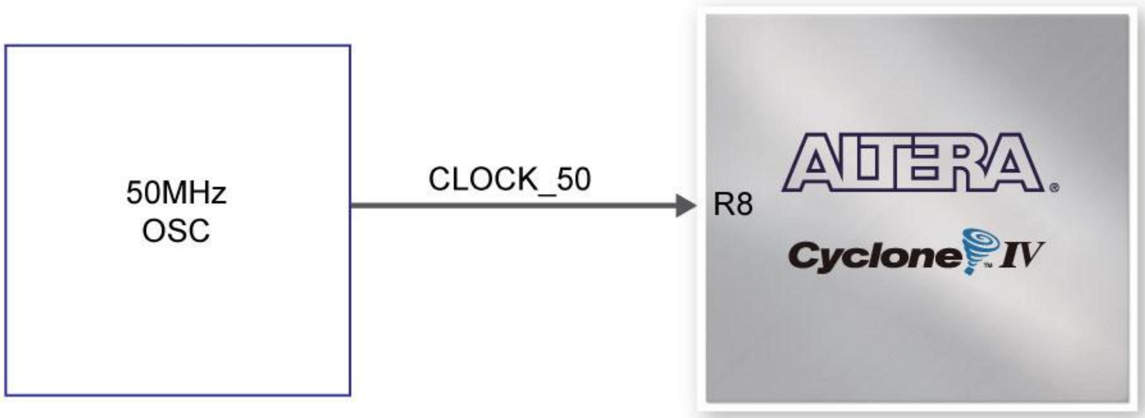


Figure 3-12 Block diagram of the clock distribution

3.9 Power Supply

The DE0-Nano board's power is provided through the USB 5V power or the 5V VCC pins on the two 40-pin headers or the 2-pin power header. The DC voltage is then stepped down to various regulators. For portable project applications, connect a battery power supply (3.6~5.7V) to the 2-pin external power header shown in **Figure 3-13**.

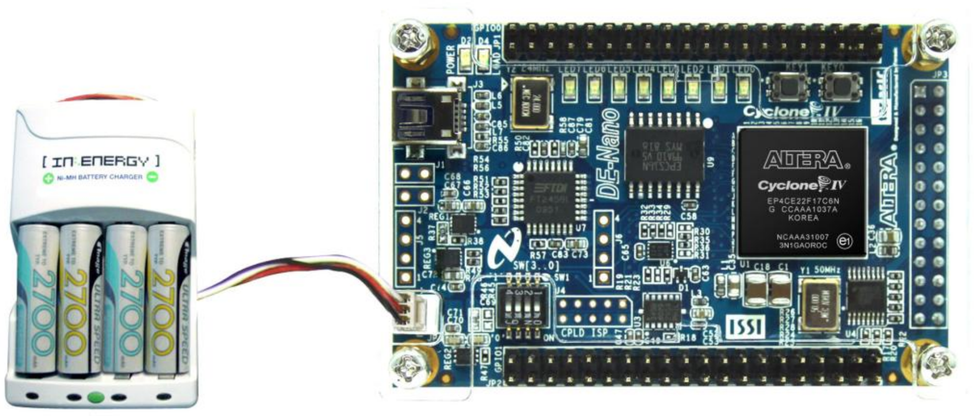


Figure 3-13 Portable Battery Connection

■ Power Distribution System

Figure 3-14 shows the power distribution system on the DE0-Nano board.

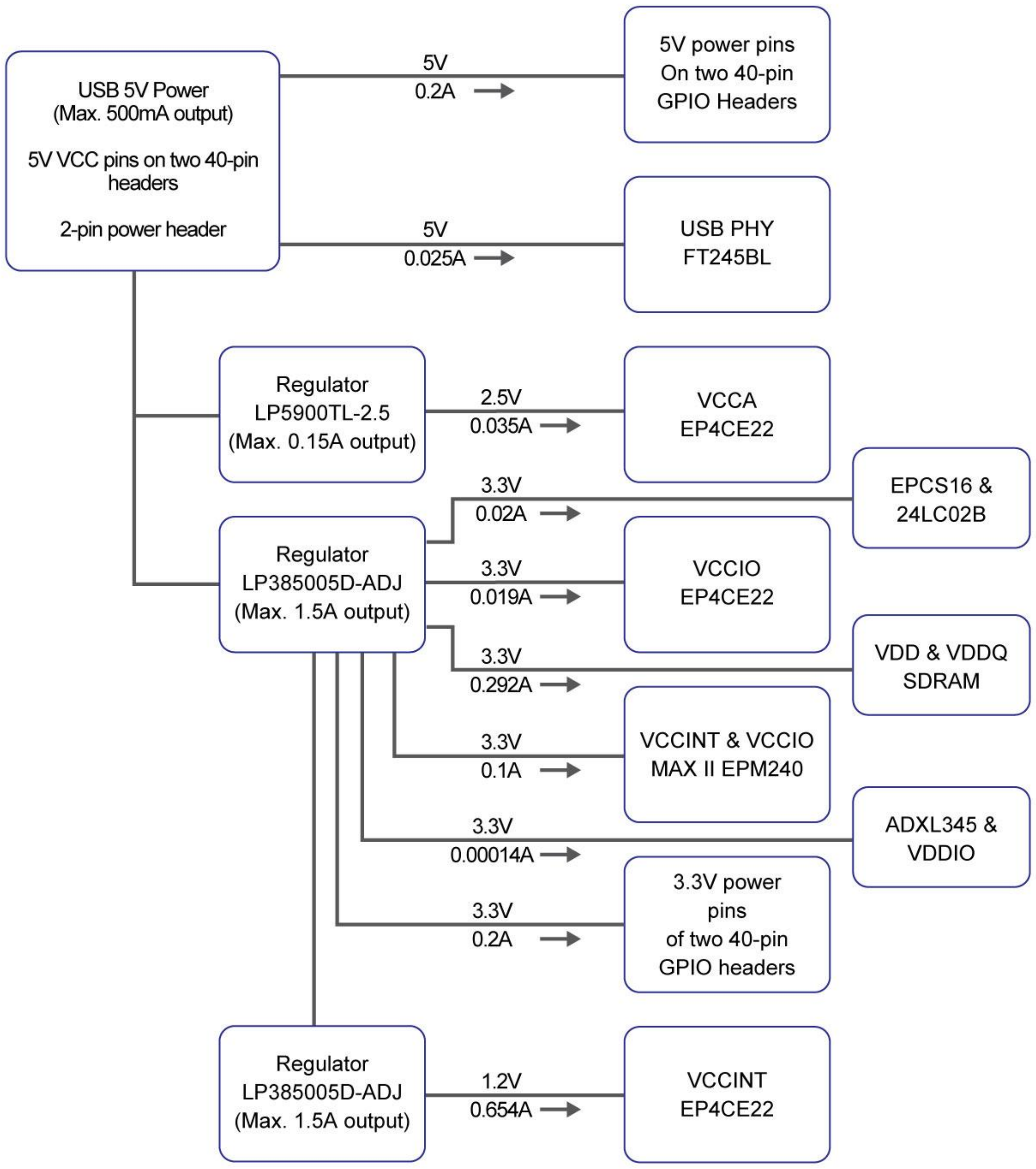

USB 5V Power
(Max. 500mA output)
5V VCC pins on two 40-pin headers
2-pin power header
5V
0.2A
5V power pins On two 40-pin GPIO Headers
5V
0.025A
USB PHY FT245BL
Regulator LP5900TL-2.5 (Max. 0.15A output)
2.5V
0.035A
VCCA EP4CE22
3.3V
0.02A
EPCS16 & 24LC02B
Regulator LP385005D-ADJ (Max. 1.5A output)
3.3V
0.019A
VCCIO EP4CE22
3.3V
0.292A
VDD & VDDQ SDRAM
3.3V
0.1A
VCCINT & VCCIO MAX II EPM240
3.3V
0.00014A
ADXL345 & VDDIO
3.3V
0.2A
3.3V power pins of two 40-pin GPIO headers
Regulator LP385005D-ADJ (Max. 1.5A output)
1.2V
0.654A
VCCINT EP4CE22

Figure 3-14 DE0-Nano Power Distribution System

Chapter 4

DE0-Nano Control Panel

The DE0-Nano board comes with a Control Panel facility that allows users to access various components on the board from a host computer. The host computer communicates with the board through a USB connection. The facility can be used to verify the functionality of components on the board or be used as a debug tool while developing RTL code.

This chapter first presents some basic functions of the Control Panel, then describes its structure in block diagram form, and finally describes its capabilities.

4.1 Control Panel Setup

The Control Panel Software Utility is located in the directory *"tools/DE0_NANO_ControlPanel*" in the **DE0-Nano System CD**. It's free of installation, just copy the whole folder to your host computer and launch the control panel by executing the "DE0_NANO_ControlPanel.exe".

Specific control circuit should be downloaded to your FPGA board before the control panel can request it to perform required tasks. The program will call Quartus II tools to download the control circuit to the FPGA board through USB-Blaster[USB-0] connection.

To activate the Control Panel, perform the following steps:

1. Make sure Quartus II 10.0 or later version is installed successfully on your PC.
2. Connect a USB A to Mini-B cable to a USB (Type A) host port and to the board.
3. Start the executable DE0_NANO_ControlPanel.exe on the host computer. The Control Panel user interface shown in **Figure 4-1** will appear.
5. The DE0_NANO_ControlPanel.sof bit stream is loaded automatically as soon as the DE0_NANO_ControlPanel.exe is launched.
6. In case the connection is disconnected, click on CONNECT where the .sof will be re-loaded onto the board.
7. *Note, the Control Panel will occupy the USB port until you close that port; you cannot use Quartus II to download a configuration file into the FPGA until the USB port is closed.*

8. The Control Panel is now ready for use; experience it by setting the ON/OFF status for some LEDs and observing the result on the DE0-Nano board.

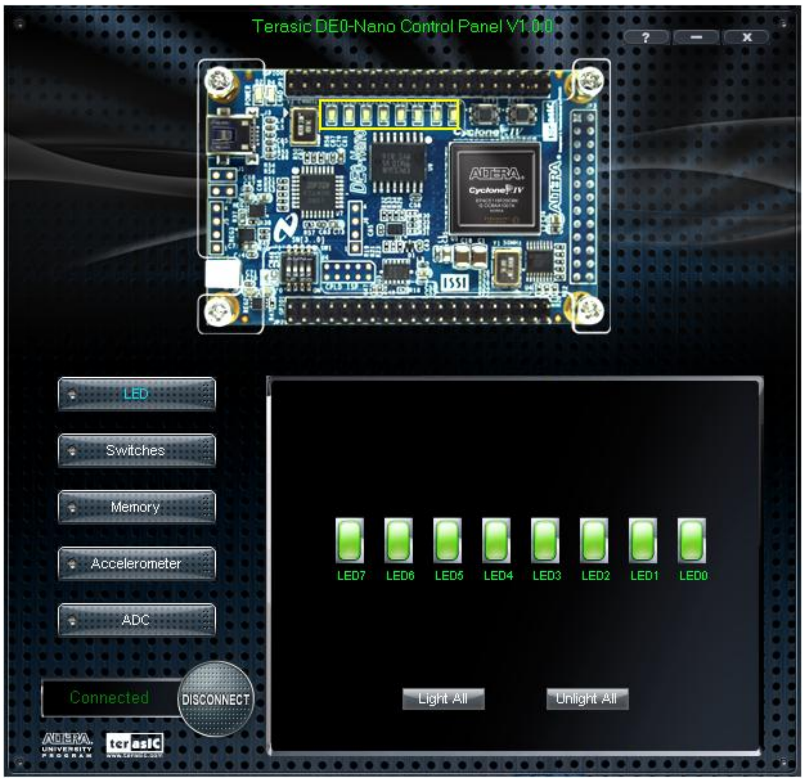


Figure 4-1 The DE0-Nano Control Panel

The concept of the DE0-Nano Control Panel is illustrated in **Figure 4-2**. The “Control Circuit” that performs the control functions is implemented in the FPGA board. It communicates with the Control Panel window, which is active on the host computer, via the USB Blaster link. The graphical interface is used to issue commands to the control circuit. It handles all requests and performs data transfers between the computer and the DE0-Nano board.

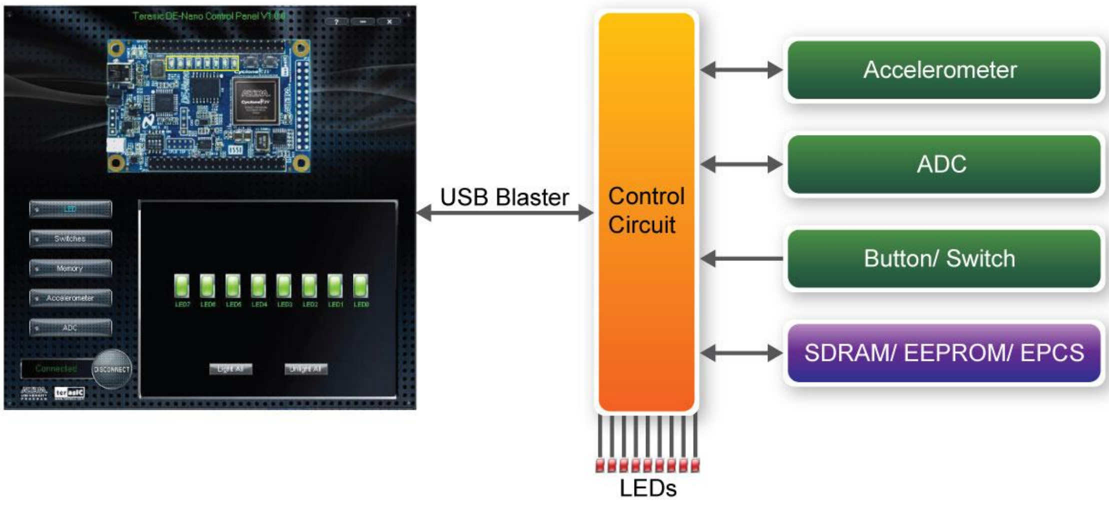


Figure 4-2 The DE0-Nano Control Panel concept

The DE0-Nano Control Panel can be used to light up LEDs, change the buttons/switches status, read/write to SDRAM Memory, read ADC channels, and display the Accelerometer information.

4.2 Controlling the LEDs

A simple function of the Control Panel is to allow setting the values displayed on LEDs. Choosing the **LED** tab leads to the window in Figure 4-3. Here, you can directly turn the LEDs on or off individually or by clicking "Light All" or "Unlight All".

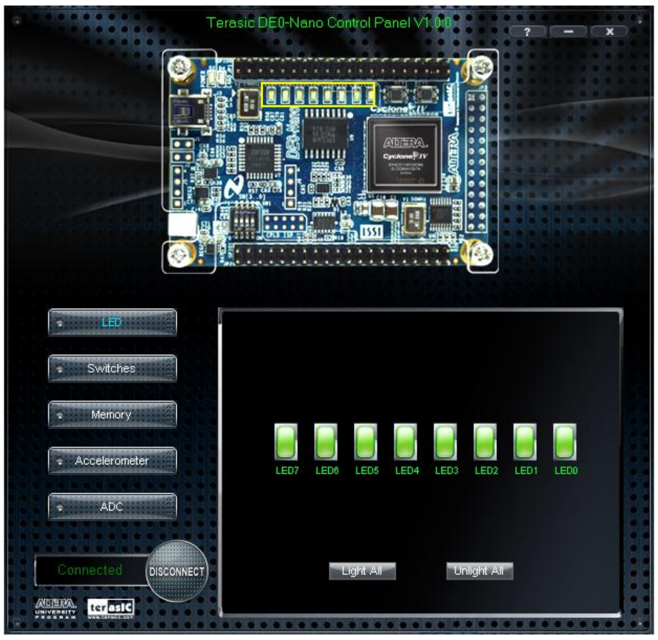


Figure 4-3 Controlling LEDs

4.3 Switches and Push-buttons

Choosing the **Switches** tab leads to the window in Figure 4-4. The function is designed to monitor the status of slide switches and push-buttons in real time and show the status in a graphical user interface. It can be used to verify the functionality of the slide switches and push-buttons.

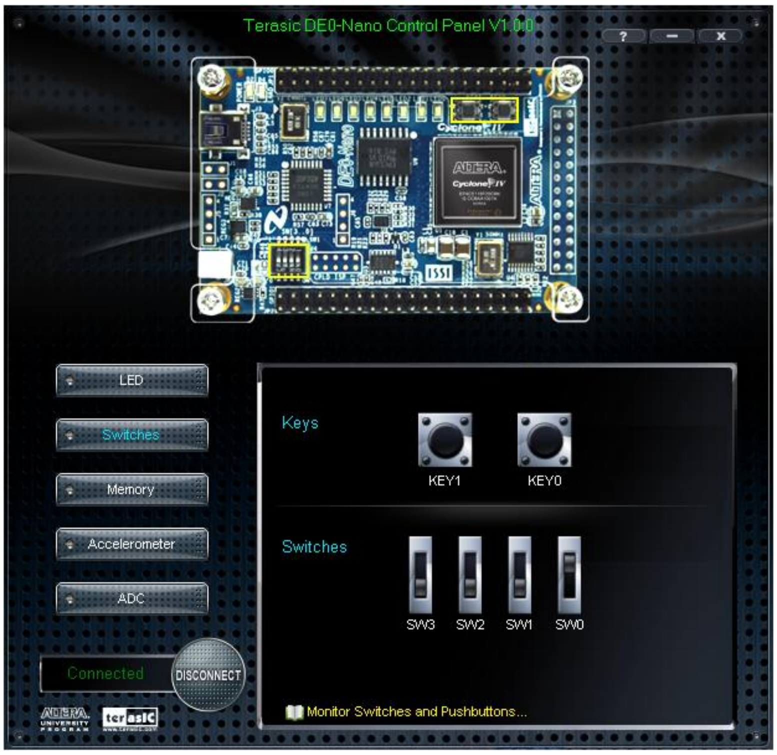
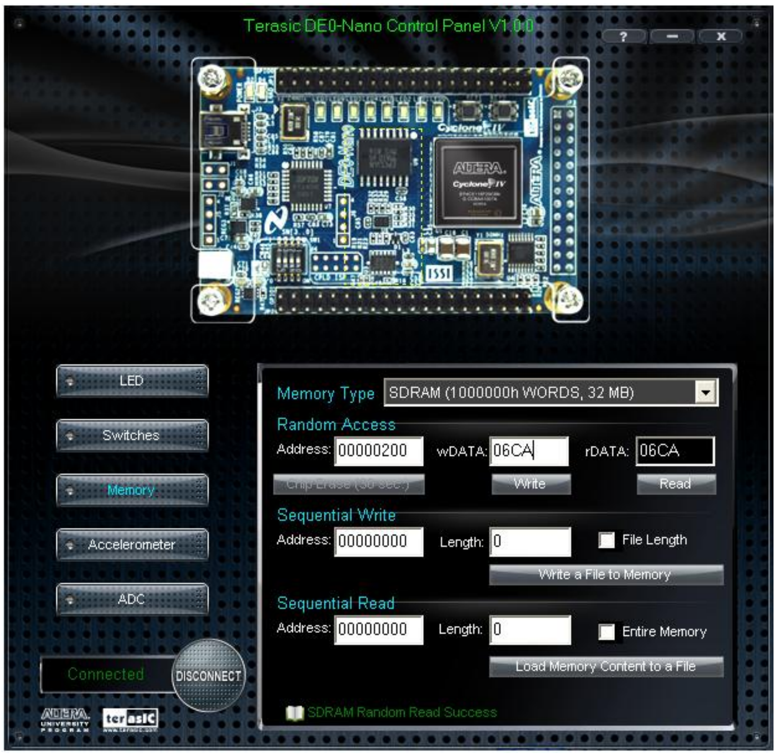


Figure 4-4 Monitoring switches and buttons

The ability to check the status of push-button and slide switch is not needed in typical design activities. However, it provides users a simple mechanism for verifying if the buttons and switches are functioning correctly. Thus, it can be used for troubleshooting purposes.

4.4 Memory Controller

The Control Panel can be used to write/read data to/from the SDRAM/EEPROM/EPCS on the DE0-Nano board. As an example, we will describe how the SDRAM may be accessed; the same approach is used to access the EEPROM and EPCS. Click on the Memory tab and select “SDRAM” to reach the window in Figure 4-6.



Terasic DE0-Nano Control Panel V1.0.0

Memory Type: SDRAM (1000000h WORDS, 32 MB)

Random Access
Address: 00000200 wDATA: 06CA rDATA: 06CA

Sequential Write
Address: 00000000 Length: 0 File Length

Sequential Read
Address: 00000000 Length: 0 Entire Memory

Connected

SDRAM Random Read Success

Figure 4-5 Accessing the SDRAM

A 16-bit word can be written into the SDRAM by entering the address of the desired location, specifying the data to be written, and pressing the Write button. Contents of the location can be read by pressing the Read button. Figure 4-5 depicts the result of writing the hexadecimal value 06CA into offset address 200, followed by reading the same location.

The Sequential Write function of the Control Panel is used to write the contents of a file into the SDRAM as follows:

1. Specify the starting address in the Address box.
2. Specify the number of bytes to be written in the Length box. If the entire file is to be loaded, then a checkmark may be placed in the File Length box instead of giving the number of bytes.
3. To initiate the writing process, click on the Write a File to Memory button.
4. When the Control Panel responds with the standard Windows dialog box asking for the source file, specify the desired file in the usual manner.

The Control Panel also supports loading files with a .hex extension. Files with a .hex extension are ASCII text files that specify memory values using ASCII characters to represent hexadecimal values. For example, a file containing the line

0123456789ABCDEF

Defines eight 8-bit values: 01, 23, 45, 67, 89, AB, CD, EF. These values will be loaded consecutively into the memory.

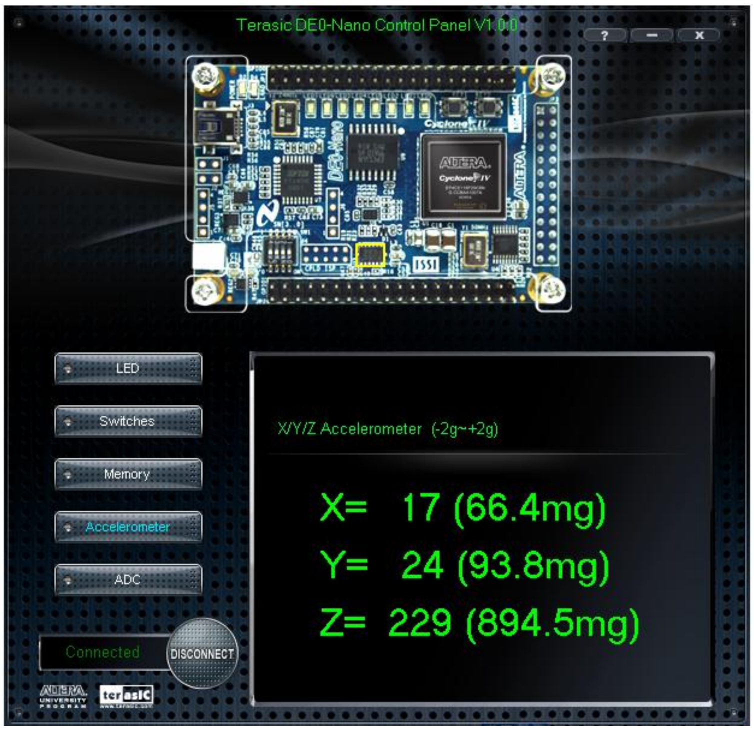
The Sequential Read function is used to read the contents of the SDRAM and fill them into a file as follows:

1. Specify the starting address in the Address box.
2. Specify the number of bytes to be copied into the file in the Length box. If the entire contents of the SDRAM are to be copied (which involves all 32 Mbytes), then place a checkmark in the Entire Memory box.
3. Press Load Memory Content to a File button.
4. When the Control Panel responds with the standard Windows dialog box asking for the destination file, specify the desired file in the usual manner.

Users can use the similar way to access the SDRAM, EEPROM and EPCS. Please note that users need to erase the EPCS before writing data to it.

4.5 Digital Accelerometer

The Control Panel can be used to display the status of the Digital Accelerometer where it measures the output of its 3-axis (X, Y, Z). The measurement range is set to ±2g. Figure 4-6 shows the current digital accelerometer status of the DE0-Nano when **Accelerometer** tab is clicked.



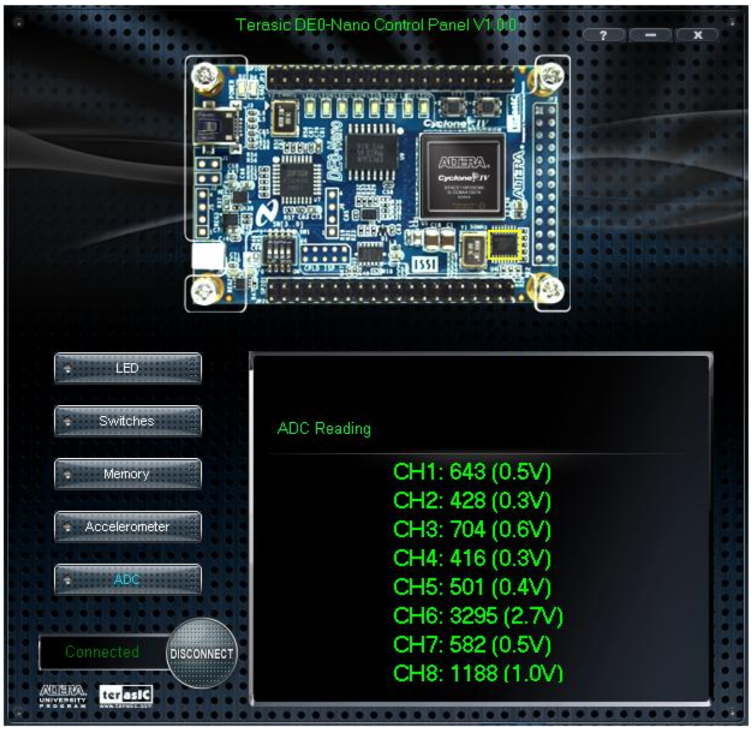
X/Y/Z Accelerometer (-2g~+2g)

X= 17 (66.4mg)
Y= 24 (93.8mg)
Z= 229 (894.5mg)

Figure 4-6 Digital Accelerometer status

4.6 ADC

From the Control Panel, users are able to read the eight-channel 12-bit analog-to-digital converter. Figure 4-7 shows the ADC readings when the **ADC** tab is chosen.



ADC Reading

CH1: 643 (0.5V)
CH2: 428 (0.3V)
CH3: 704 (0.6V)
CH4: 416 (0.3V)
CH5: 501 (0.4V)
CH6: 3295 (2.7V)
CH7: 582 (0.5V)
CH8: 1188 (1.0V)

Figure 4-7 ADC Readings

4.7 Overall Structure of the DE0-Nano Control Panel

The DE0-Nano Control Panel is based on a Nios II SOPC system instantiated in the Cyclone IV E FPGA with software running on the on-chip memory. The software part is implemented in C code; the hardware part is implemented in Verilog HDL code with SOPC builder. The source code is not available on the DE0-Nano System CD.

To run the Control Panel, users should make the configuration according to Section 3.1. Figure 4-8 depicts the structure of the Control Panel. Each input/output device is controlled by the Nios II Processor instantiated in the FPGA chip. The communication with the PC is done via the USB Blaster link. The Nios II interprets the commands sent from the PC and performs the corresponding actions.

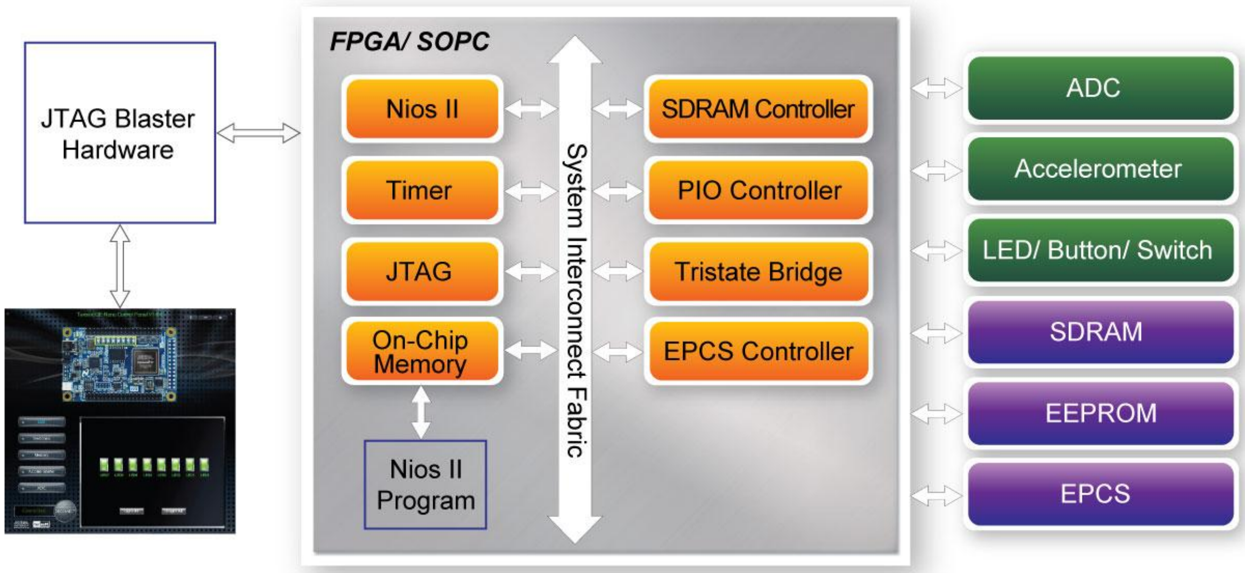


Figure 4-8 The block diagram of the DE0-Nano Control Panel

Chapter 5

DE0-Nano System Builder

This chapter describes how users can create a custom design project on the DE0-Nano board by using DE0-Nano Tool – DE0-Nano System Builder.

5.1 Introduction

The DE0-Nano System Builder is a Windows based software utility, designed to assist users to create a Quartus II project for the DE0-Nano board within minutes. The generated Quartus II project files include:

- Quartus II Project File (.qpf)
- Quartus II Setting File (.qsf)
- Top-Level Design File (.v)
- Synopsys Design Constraints file (.sdc)
- Pin Assignment Document (.htm)

By providing the above files, DE0-Nano System Builder prevents occurrence of situations that are prone to errors when users manually edit the top-level design file or place pin assignments. The common mistakes that users encounter are the following:

1. Board damaged for wrong pin/bank voltage assignments.
2. Board malfunction caused by wrong device connections or missing pin counts for connected ends.
3. Performance degeneration because of improper pin assignments.

5.2 General Design Flow

This section will introduce the general design flow to build a project for the DE0-Nano board via the DE0-Nano System Builder. The general design flow is illustrated in **Figure 5-1**.

Users should launch DE0-Nano System Builder and create a new project according to their design requirements. When users complete the settings, the DE0-Nano System Builder will generate two major files which include top-level design file (.v) and Quartus II setting file (.qsf).

The top-level design file contains top-level verilog HDL wrapper for users to add their own design/logic. The Quartus II setting file contains information such as FPGA device type, top-level pin assignment, and I/O standard for each user-defined I/O pin.

Finally, Quartus II programmer must be used to download SOF file to DE0-Nano board using JTAG interface.

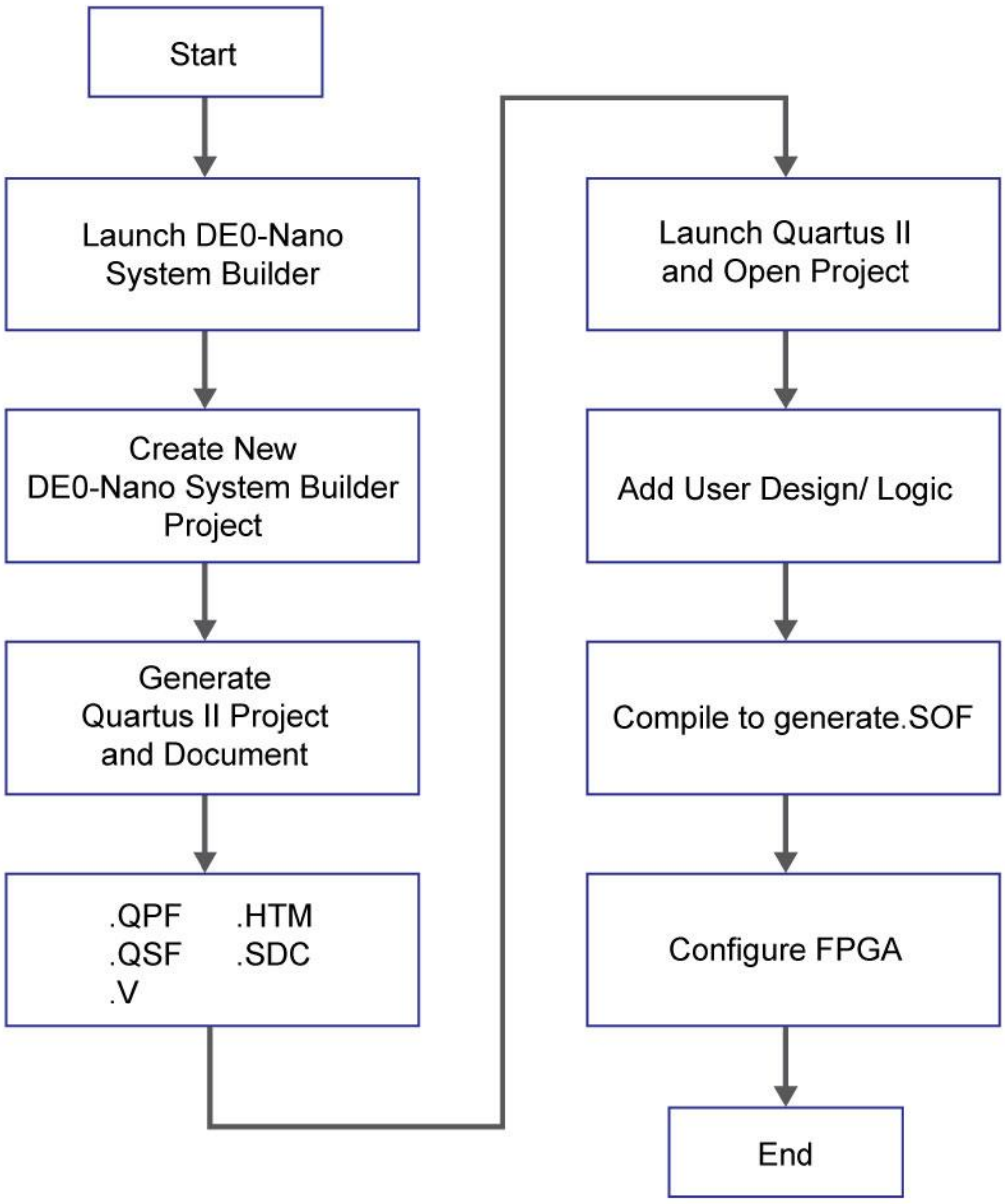

Start
Launch DE0-Nano System Builder
Create New DE0-Nano System Builder Project
Generate Quartus II Project and Document
.QPF .HTM
.QSF .SDC
.V
Launch Quartus II and Open Project
Add User Design/ Logic
Compile to generate.SOF
Configure FPGA
End

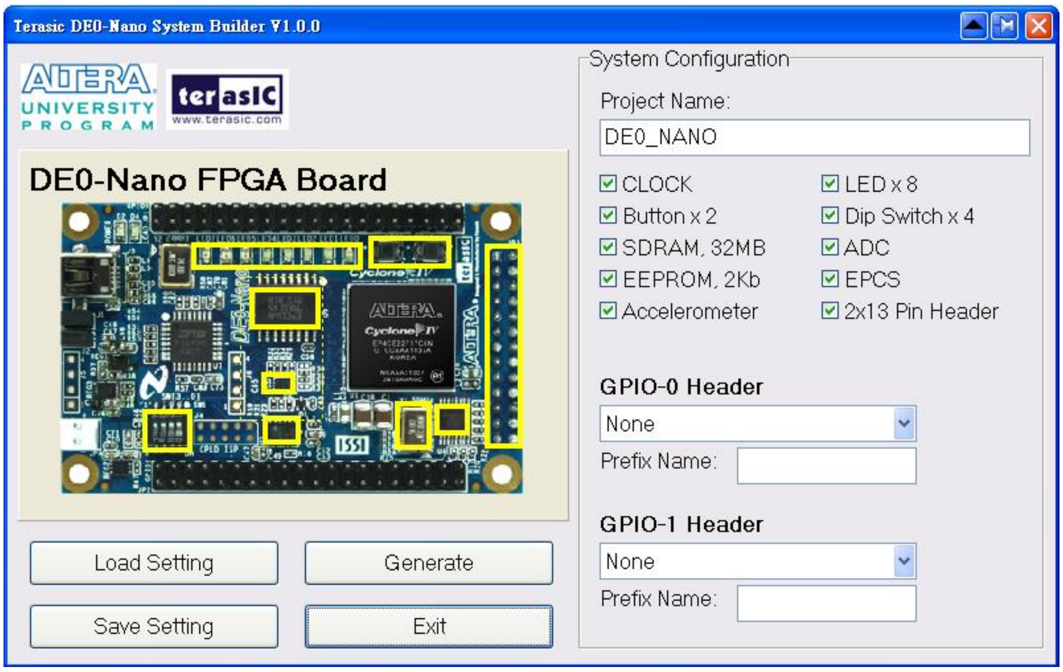
Figure 5-1 The general design flow of building a design

5.3 Using DE0-Nano System Builder

This section provides the detailed procedures on how the DE0-Nano System Builder is used.

■ Install and launch the DE0-Nano System Builder

The DE0-Nano System Builder is located in the directory: *"Tools\DE0_NANO_SystemBuilder"* on the DE0-Nano System CD. Users can copy the whole folder to a host computer without installing the utility. Launch the DE0-Nano System Builder by executing the DE0_NANO_SystemBuilder.exe on the host computer and the GUI window will appear as shown in **Figure 5-2**.



System Configuration

Project Name: DE0_NANO

☑ CLOCK
☑ LED x 8
☑ Button x 2
☑ Dip Switch x 4
☑ SDRAM, 32MB
☑ ADC
☑ EEPROM, 2Kb
☑ EPCS
☑ Accelerometer
☑ 2x13 Pin Header

GPIO-0 Header: None
Prefix Name:

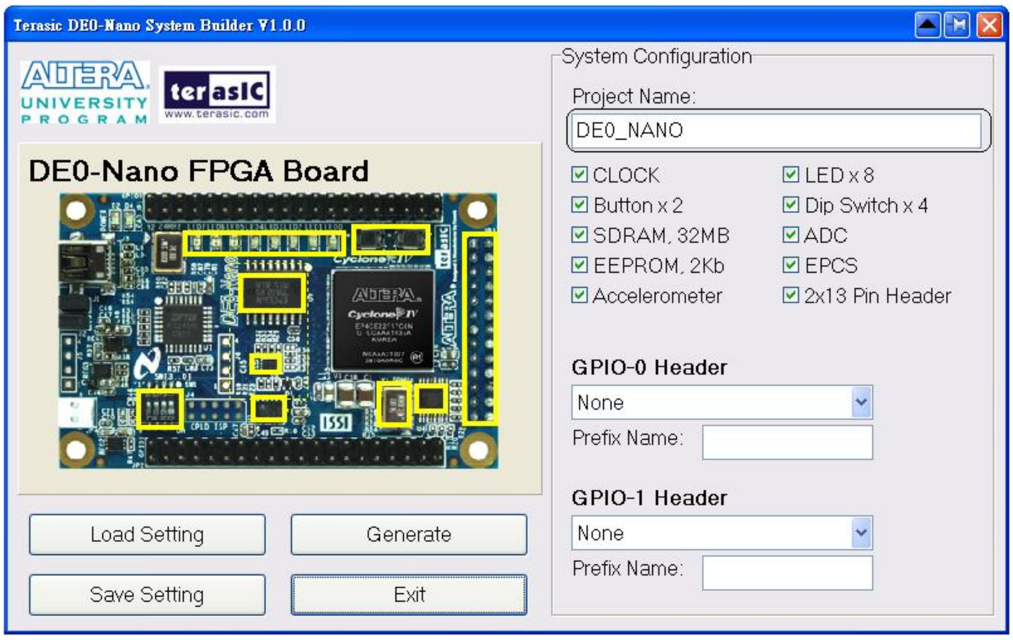
GPIO-1 Header: None
Prefix Name:

Figure 5-2 The DE0-Nano System Builder window

■ Input Project Name

Input project name as show in **Figure 5-3**.

Project Name: Type in an appropriate name here, it will automatically be assigned as the name of your top-level design entity.


Terasic DE0-Nano System Builder V1.0.0

DE0-Nano FPGA Board

System Configuration

Project Name: DE0_NANO

- ☑ CLOCK
- ☑ LED x 8
- ☑ Button x 2
- ☑ Dip Switch x 4
- ☑ SDRAM, 32MB
- ☑ ADC
- ☑ EEPROM, 2Kb
- ☑ EPCS
- ☑ Accelerometer
- ☑ 2x13 Pin Header

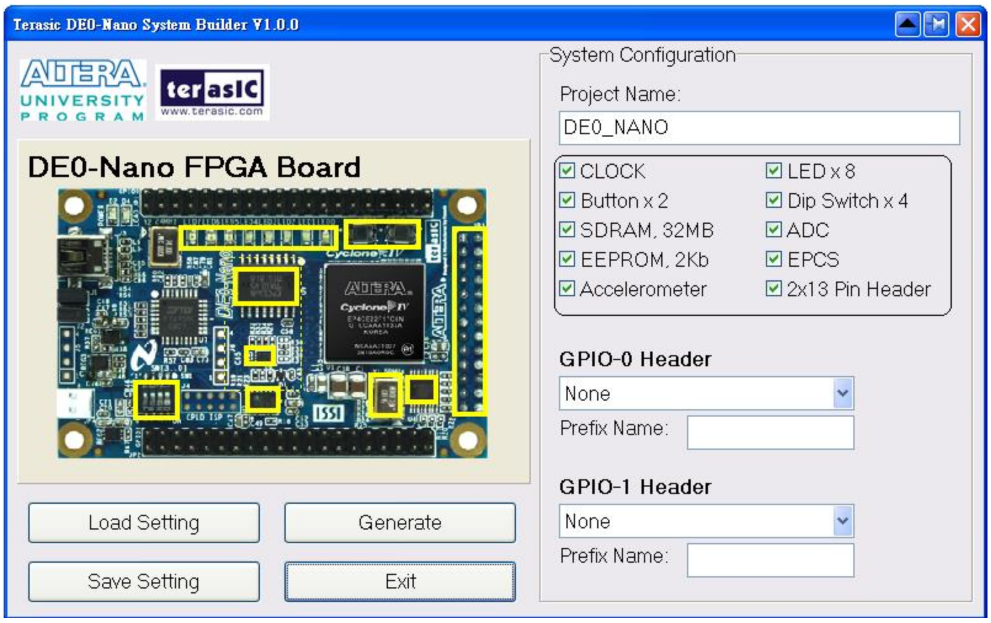
GPIO-0 Header: None
Prefix Name:

GPIO-1 Header: None
Prefix Name:

Figure 5-3 The DE0-Nano Board Type and Project Name

■ System Configuration

Under System Configuration users are given the flexibility of enabling their choice of included components on the DE0-Nano as shown in **Figure 5-4**. Each component of the DE0-Nano is listed where users can enable or disable a component according to their design by simply marking a check or removing the check in the field provided. If the component is enabled, the DE0-Nano System Builder will automatically generate the associated pin assignments including the pin name, pin location, pin direction, and I/O standard.


Terasic DE0-Nano System Builder V1.0.0

DE0-Nano FPGA Board

System Configuration

Project Name: DE0_NANO

- ☑ CLOCK
- ☑ LED x 8
- ☑ Button x 2
- ☑ Dip Switch x 4
- ☑ SDRAM, 32MB
- ☑ ADC
- ☑ EEPROM, 2Kb
- ☑ EPCS
- ☑ Accelerometer
- ☑ 2x13 Pin Header

GPIO-0 Header: None
Prefix Name:

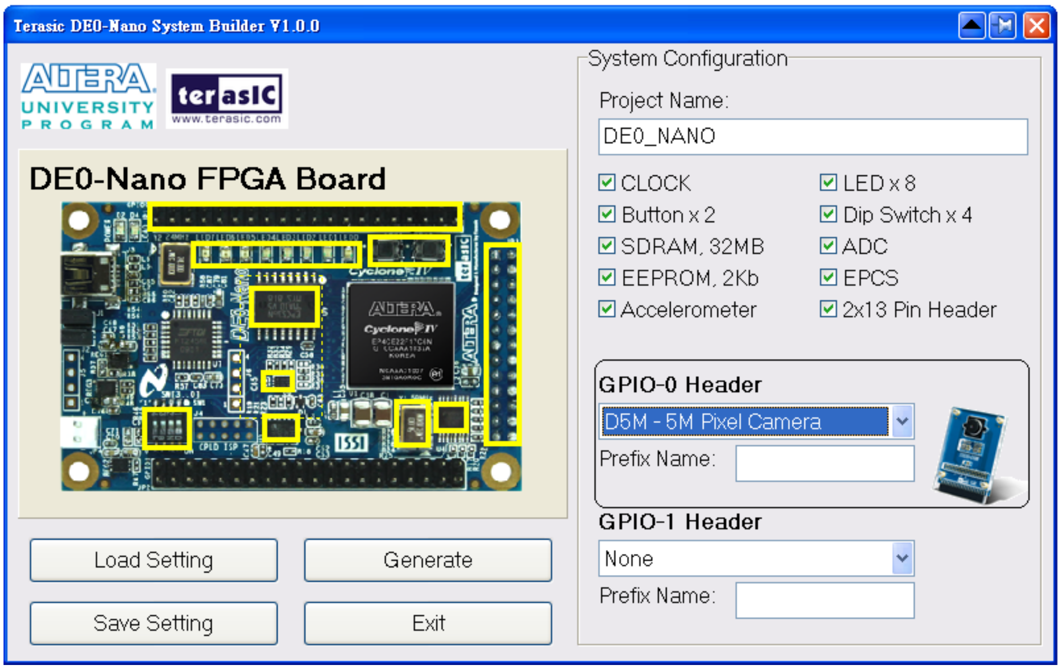
GPIO-1 Header: None
Prefix Name:

Figure 5-4 System Configuration Group

■ GPIO Expansion

Users can connect GPIO expansion card onto GPIO header located on the DE0-Nano board as shown in **Figure 5-5**. Select the appropriate daughter card you wish to include in your design from the drop-down menu. The system builder will automatically generate the associated pin assignments including the pin name, pin location, pin direction, and IO standard.

If a customized daughter board is used, users can select “GPIO Default” followed by changing the pin name and pin direction according to the specification of the customized daughter board.



System Configuration

Project Name: DE0_NANO

☑ CLOCK ☑ LED x 8
☑ Button x 2 ☑ Dip Switch x 4
☑ SDRAM, 32MB ☑ ADC
☑ EEPROM, 2Kb ☑ EPCS
☑ Accelerometer ☑ 2x13 Pin Header

GPIO-0 Header: D5M - 5M Pixel Camera
Prefix Name:

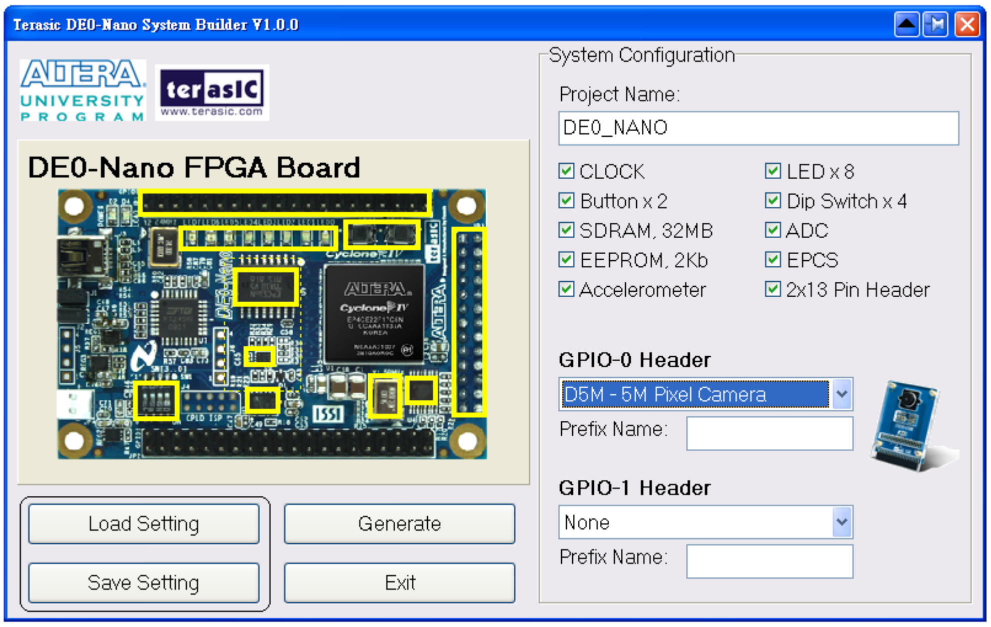
GPIO-1 Header: None
Prefix Name:

Figure 5-5 GPIO Expansion Group

The “Prefix Name” is an optional feature which denotes the prefix pin name of the daughter card assigned in your design. Users may leave this field empty.

■ Project Setting Management

The DE0-Nano System Builder also provides functions to restore default setting, loading a setting, and saving users’ board configuration file shown in **Figure 5-6**. Users can save the current board configuration information into a .cfg file and load it to the DE0-Nano System Builder.


Terasic DE0-Nano System Builder V1.0.0

DE0-Nano FPGA Board

System Configuration

Project Name: DE0_NANO

☑ CLOCK ☑ LED x 8
☑ Button x 2 ☑ Dip Switch x 4
☑ SDRAM, 32MB ☑ ADC
☑ EEPROM, 2Kb ☑ EPCS
☑ Accelerometer ☑ 2x13 Pin Header

GPIO-0 Header: D5M - 5M Pixel Camera
Prefix Name:

GPIO-1 Header: None
Prefix Name:

Figure 5-6 Project Settings

■ Project Generation

When users press the Generate button, the DE0-Nano System Builder will generate the corresponding Quartus II files and documents as listed in the Table 5-1:

Table 5-1 The files generated by DE0-Nano System Builder

No.	***Filename***	***Description***
1	**<Project name>.v**	**Top level verilog HDL file for Quartus II**
2	**<Project name>.qpf**	**Quartus II Project File**
3	**<Project name>.qsf**	**Quartus II Setting File**
4	**<Project name>.sdc**	**Synopsys Design Constraints file for Quartus II**
5	**<Project name>.htm**	**Pin Assignment Document**

Users can use Quartus II software to add custom logic into the project and compile the project to generate the SRAM Object File (.sof).

Chapter 6

My First FPGA Project Tutorial

This tutorial provides comprehensive information that will help you understand how to create a FPGA design and run it on you DE0-Nano development board. The following sections provide a quick overview of the design flow, explain what you need to get started, and describe what you will learn.

6.1 Design Flow

Figure 6-1shows the FPGA design flow block diagram.

The standard FPGA design flow starts with design entry using schematics or a hardware description language (HDL), such as Verilog HDL or VHDL. In this step, you can create a digital circuit that is implemented inside the FPGA. The flow then proceeds through compilation, simulation, programming, and verification in the FPGA hardware.

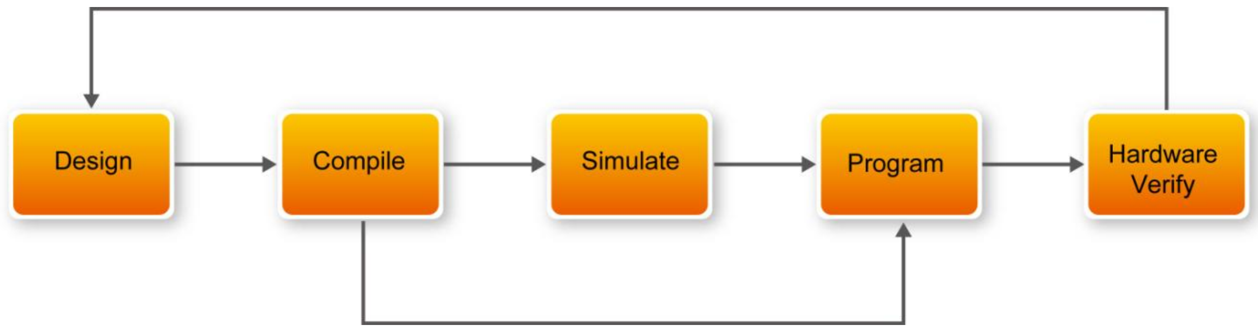


Figure 6-1 Design Flow

This tutorial guides you through all of the steps except for simulation. Although it is not covered in this document, simulation is very important to learn, and there are entire applications devoted to simulating hardware designs. There are two types of simulation, Functional and Timing Functional simulation allows you to verify that your code is manipulating the inputs and outputs appropriately. Timing (or post place-and-route) simulation verifies that the design meets timing and functions appropriately in the device.

6.2 Before You Begin

This tutorial assumes the following prerequisites

■ You generally know what a FPGA is. This tutorial does not explain the basic concepts of programmable logic.

■ You are somewhat familiar with digital circuit design and electronic design automation (EDA) tools.

■ You have installed the Altera Quartus II 10.1 software on your computer. If you do not have the Quartus II software, you can download it from the Altera web site at www.altera.com/download.

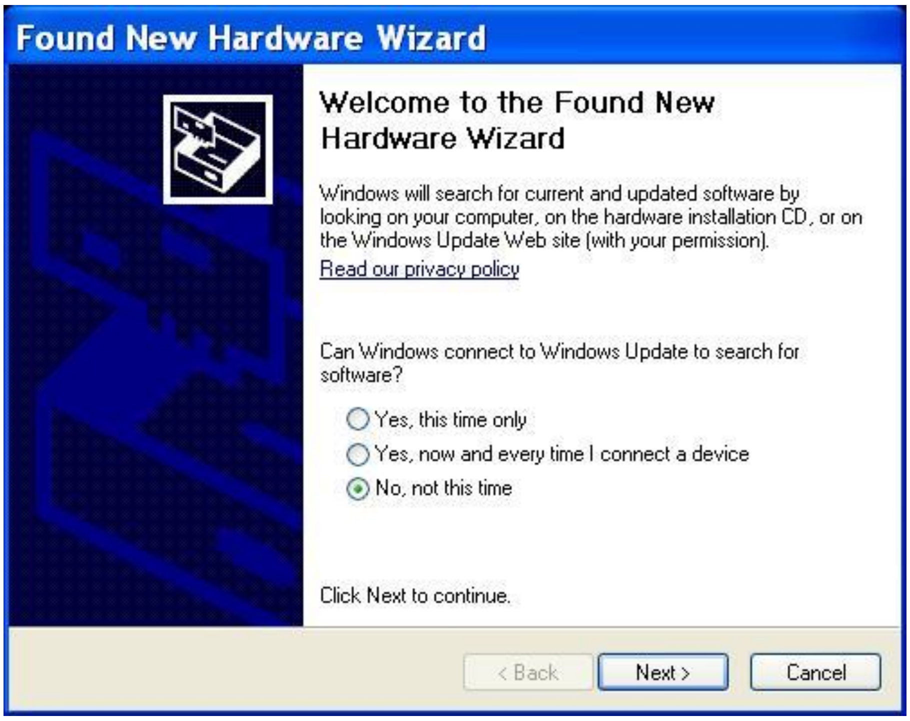
■ You have a DE0-Nano Development Board on which you will test your project. Using a development board helps you to verify whether your design is really working.

■ You have gone through the quick start guide and/or the getting started user guide for your development kit. These documents ensure that you have:

- Installed the required software.
- Determined that the development board functions properly and is connected to your computer.

Next step you should installed the USB-Blaster driver. Use the USB cable to connect the leftmost USB connector on the DE0-Nano board to a USB port on a computer that runs the Quartus II software.

The computer will recognize the new hardware connected to its USB port, but it will be unable to proceed if it does not have the required driver already installed. The DE0-Nano board is programmed by using Altera USB-Blaster mechanism. If the USB-Blaster driver is not already installed, the New Hardware Wizard in **Figure 6-2** will appear.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

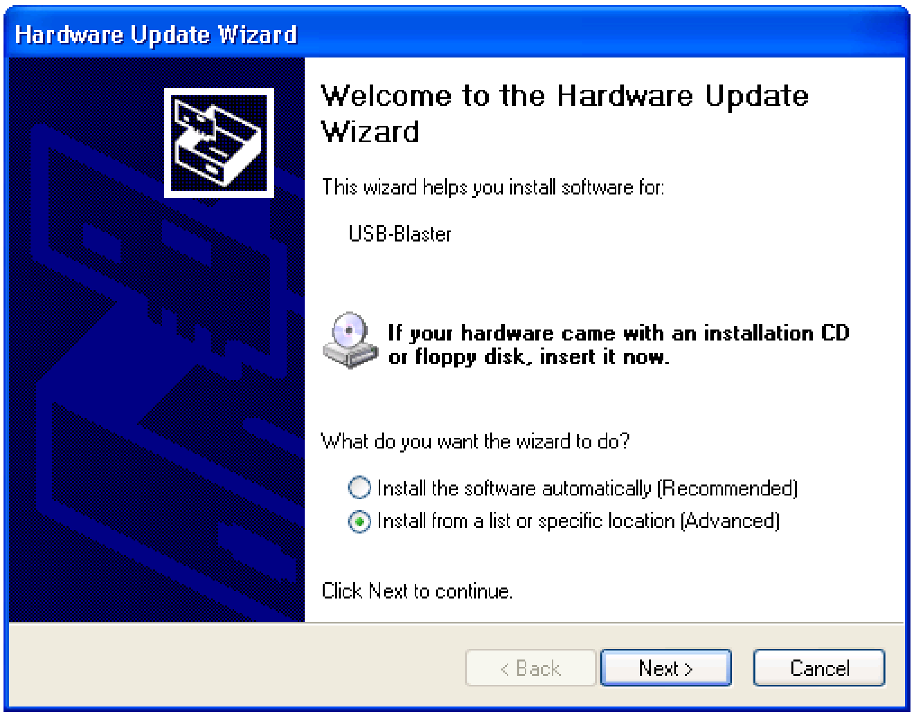
Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

Figure 6-2 Found New Hardware Wizard

Since the desired driver is not available on the Windows Update Web site, select "No, not this time" in response to the question asked and click Next. This leads to the window in **Figure 6-3**.


Hardware Update Wizard

Welcome to the Hardware Update Wizard

This wizard helps you install software for:

USB-Blaster

If your hardware came with an installation CD or floppy disk, insert it now.

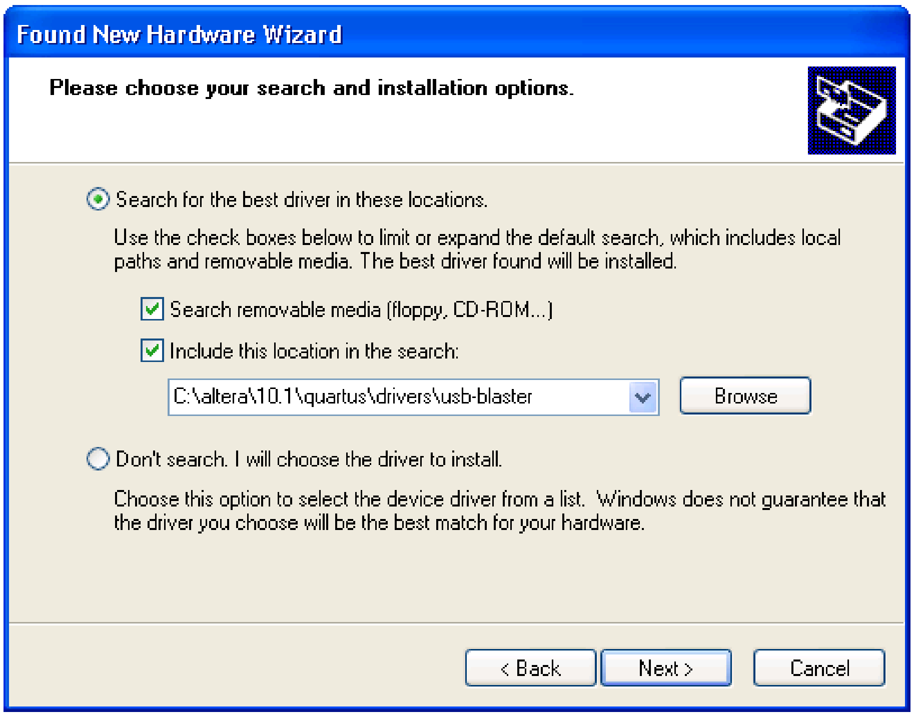
What do you want the wizard to do?

- Install the software automatically (Recommended)
- [✓ Install from a list or specific location (Advanced)]

Click Next to continue.

Figure 6-3 The driver is found in a specific location

The driver is available within the Quartus II software. Hence, select Install from a specific location and click Next to get to **Figure 6-4**.


Found New Hardware Wizard

Please choose your search and installation options.

[✓ Search for the best driver in these locations.]

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

[✓ Search removable media (floppy, CD-ROM...)]

[✓ Include this location in the search:]

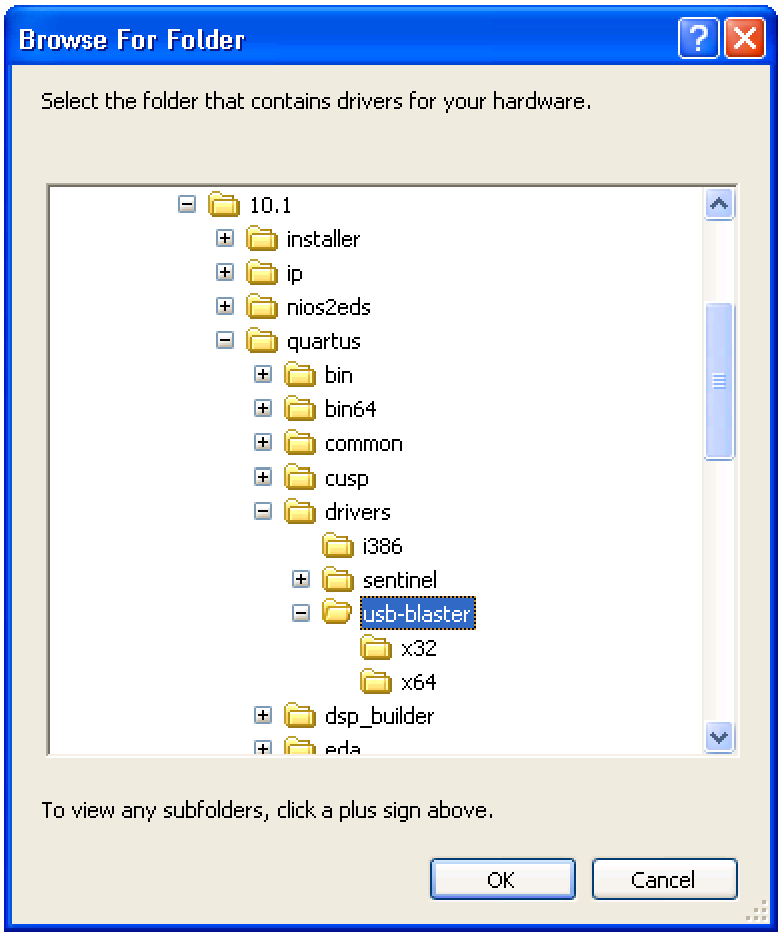
C:\altera\10.1\quartus\drivers\usb-blaster

Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Figure 6-4 Specify the location of the driver

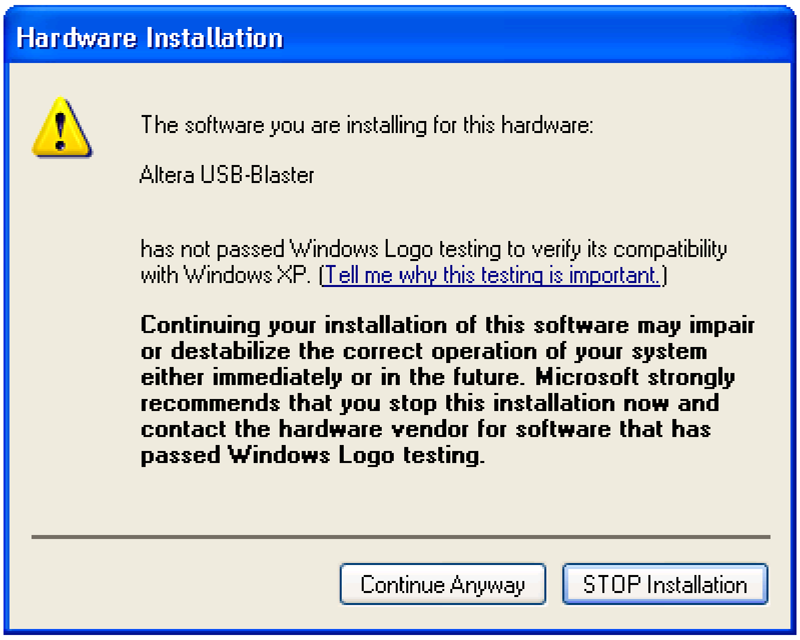
Now, choose Search for the best driver in these locations and click Browse to get to the pop-up box in **Figure 6-5** Find the desired driver, which is at location C:\altera\10.1\quartus\drivers\usb-blaster. Click OK and then upon returning to **Figure 6-4** click Next. At this point the installation will commence, but a dialog box in **Figure 6-6** will appear indicating that the driver has not passed the Windows Logo testing. Click Continue Anyway.


Browse For Folder

Select the folder that contains drivers for your hardware.

To view any subfolders, click a plus sign above.

Figure 6-5 Browse to find the location


Hardware Installation

The software you are installing for this hardware:

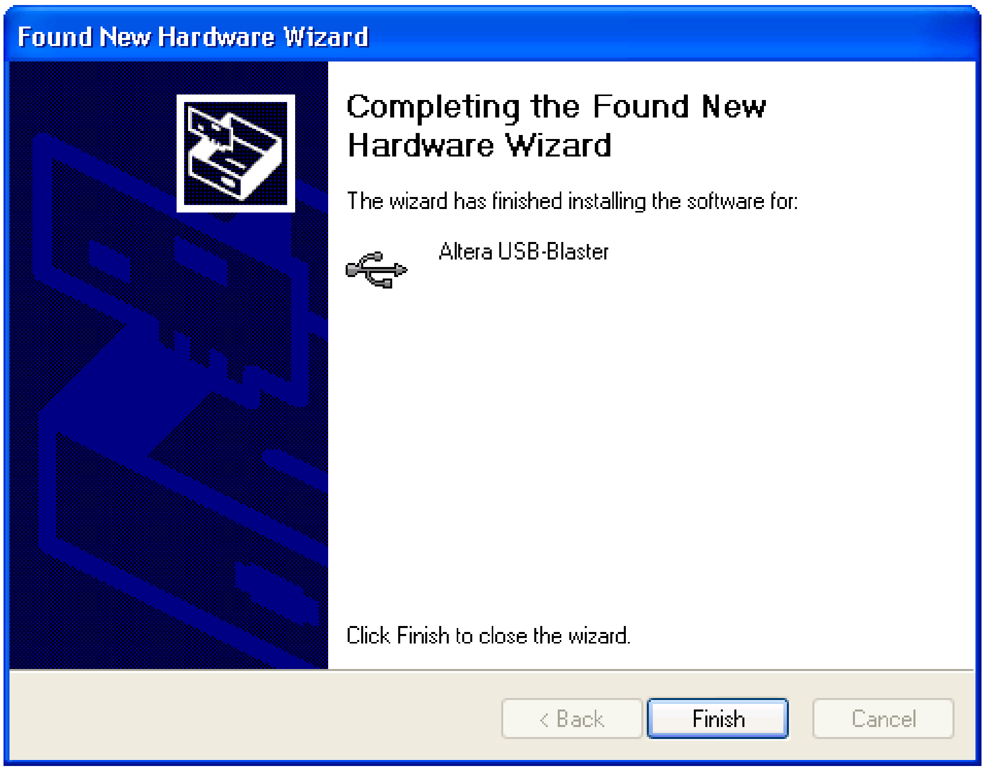
Altera USB-Blaster

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

Figure 6-6 There is no need to test the driver

The driver will now be installed as indicated in **Figure 6-7** Click Finish and you can start using the DE0-Nano board.



Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Altera USB-Blaster

Click Finish to close the wizard.

Figure 6-7 The driver is installed

6.3 What You Will Learn

In this tutorial you will perform the following tasks:

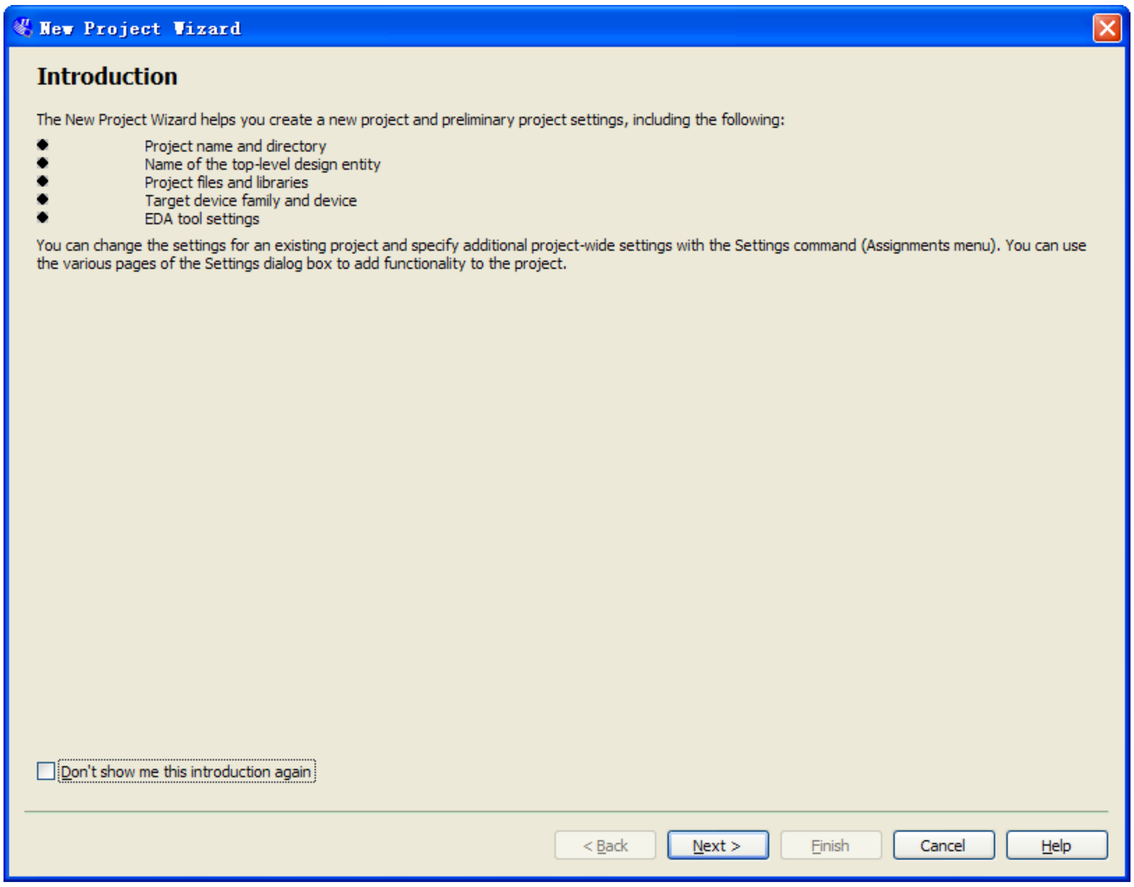
Create a design that causes LEDs on the development board to blink at a speed that is controlled by an input key—This design is easy to create and gives you visual feedback that the design works. Of course, you can use your DE0-Nano board to run other designs as well. For the LED design, you will write Verilog HDL code for a simple 32-bit counter, add a phase-locked loop (PLL) megafunction as the clock source, and add a 2-input multiplexer megafunction. When the design is running on the board, you can press an input switch to multiplex the counter bits that drive the output LEDs.

Develop a foundation to learn more about FPGAs—For example, you can create and download digital signal processing (DSP) functions onto a single chip, or build a multi-processor system, or create anything else you can imagine all on the same chip. You don't have to scour data books to find the perfect logic device or create your own ASIC. All you need is your computer, your imagination, and an Altera DE0-Nano FPGA development board.

6.4 Assign The Device

You begin this tutorial by creating a new Quartus II project. A project is a set of files that maintain information about your FPGA design. The Quartus II Settings File (.qsf) and Quartus II Project File (.qpf) files are the primary files in a Quartus II project. To compile a design or make pin assignments, you must first create a project.

In the Quartus II software, select File > New Project Wizard. The Introduction page opens. See Figure 6-8



Introduction

The New Project Wizard helps you create a new project and preliminary project settings, including the following:

- Project name and directory
- Name of the top-level design entity
- Project files and libraries
- Target device family and device
- EDA tool settings

You can change the settings for an existing project and specify additional project-wide settings with the Settings command (Assignments menu). You can use the various pages of the Settings dialog box to add functionality to the project.

☐ Don't show me this introduction again

Figure 6-8 New Project Wizard introduction

2. Click Next.

3. Enter the following information about your project:

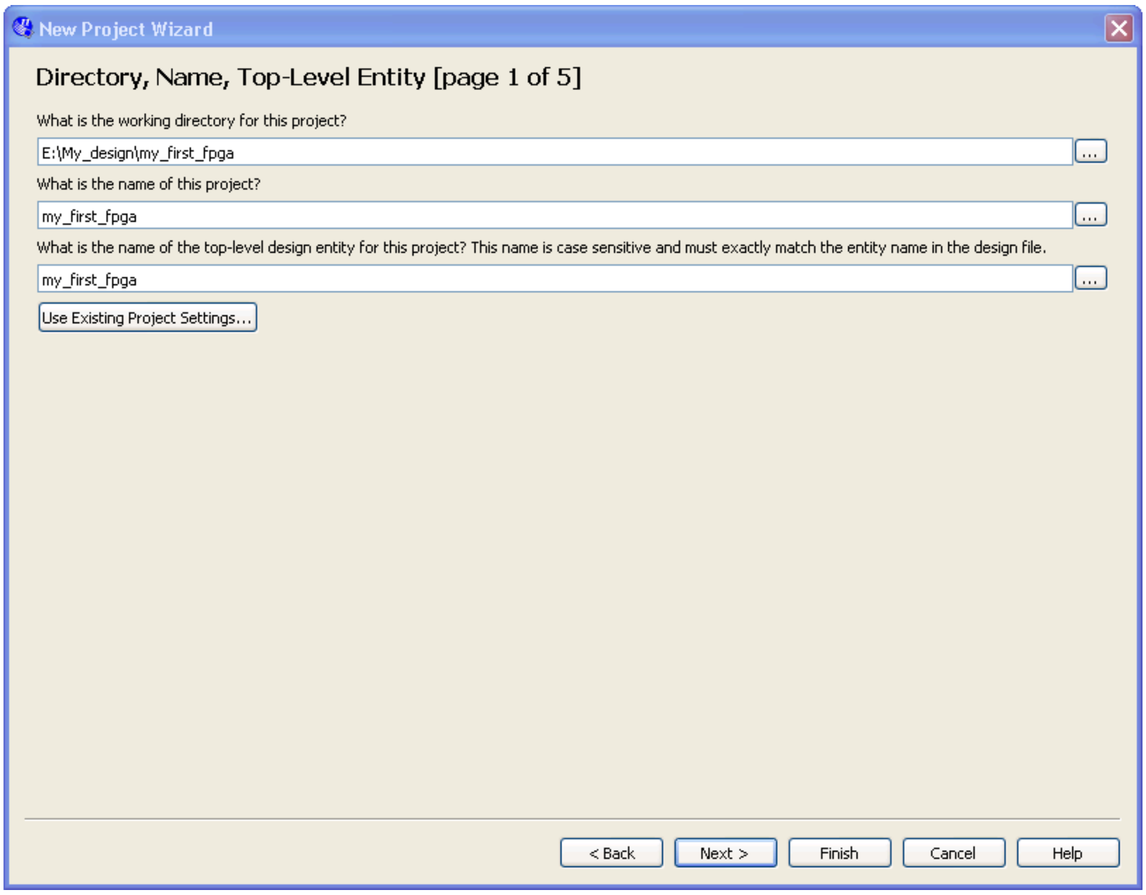
a. What is the working directory for this project? Enter a directory in which you will store your Quartus II project files for this design.

b. For example, E:\My_design\my_first_fpga.

c. File names, project names, and directories in the Quartus II software cannot contain spaces.

d. What is the name of this project? Type my_first_fpga.

e. What is the name of the top-level design entity for this project? Type my_first_fpga. See **Figure 6-9**.



Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

E:\My_design\my_first_fpga

What is the name of this project?

my_first_fpga

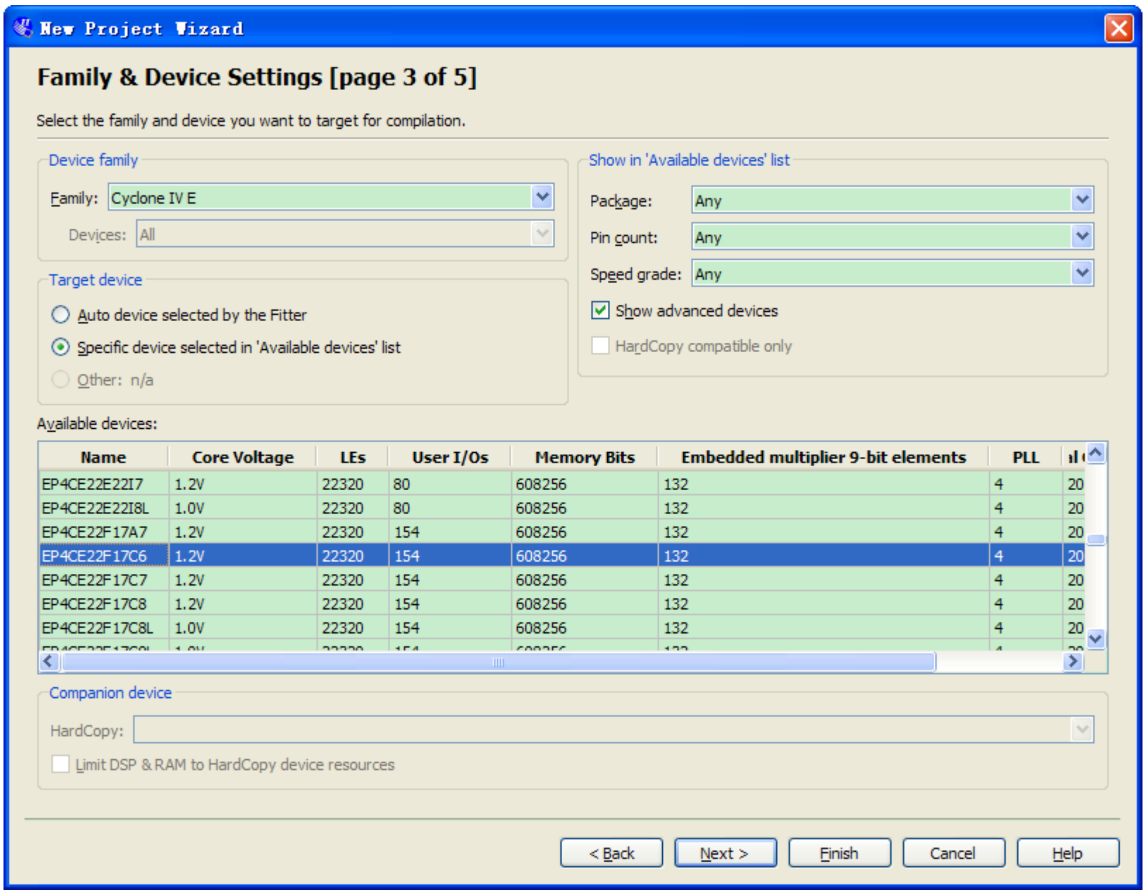
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

my_first_fpga

Figure 6-9 Project information

f. Click Next.

g. You will assign a specific FPGA device to the design and make pin assignments. See **Figure 6-10**.


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 IV E

Devices: All

Target device

Auto device selected by the Fitter [✓ Specific device selected in 'Available devices' list] Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

[✓] Show advanced devices

HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL
EP4CE22E22I7	1.2V	22320	80	608256	132	4
EP4CE22E22I8L	1.0V	22320	80	608256	132	4
EP4CE22F17A7	1.2V	22320	154	608256	132	4
EP4CE22F17C6	1.2V	22320	154	608256	132	4
EP4CE22F17C7	1.2V	22320	154	608256	132	4
EP4CE22F17C8	1.2V	22320	154	608256	132	4
EP4CE22F17C8L	1.0V	22320	154	608256	132	4

Companion device

HardCopy:

Limit DSP & RAM to HardCopy device resources

Figure 6-10 Specify the Device Example

h. Click Finish.

4. When prompted, choose Yes to create the my_first_fpga project directory. You just created your first Quartus II FPGA project. See **Figure 6-11**.

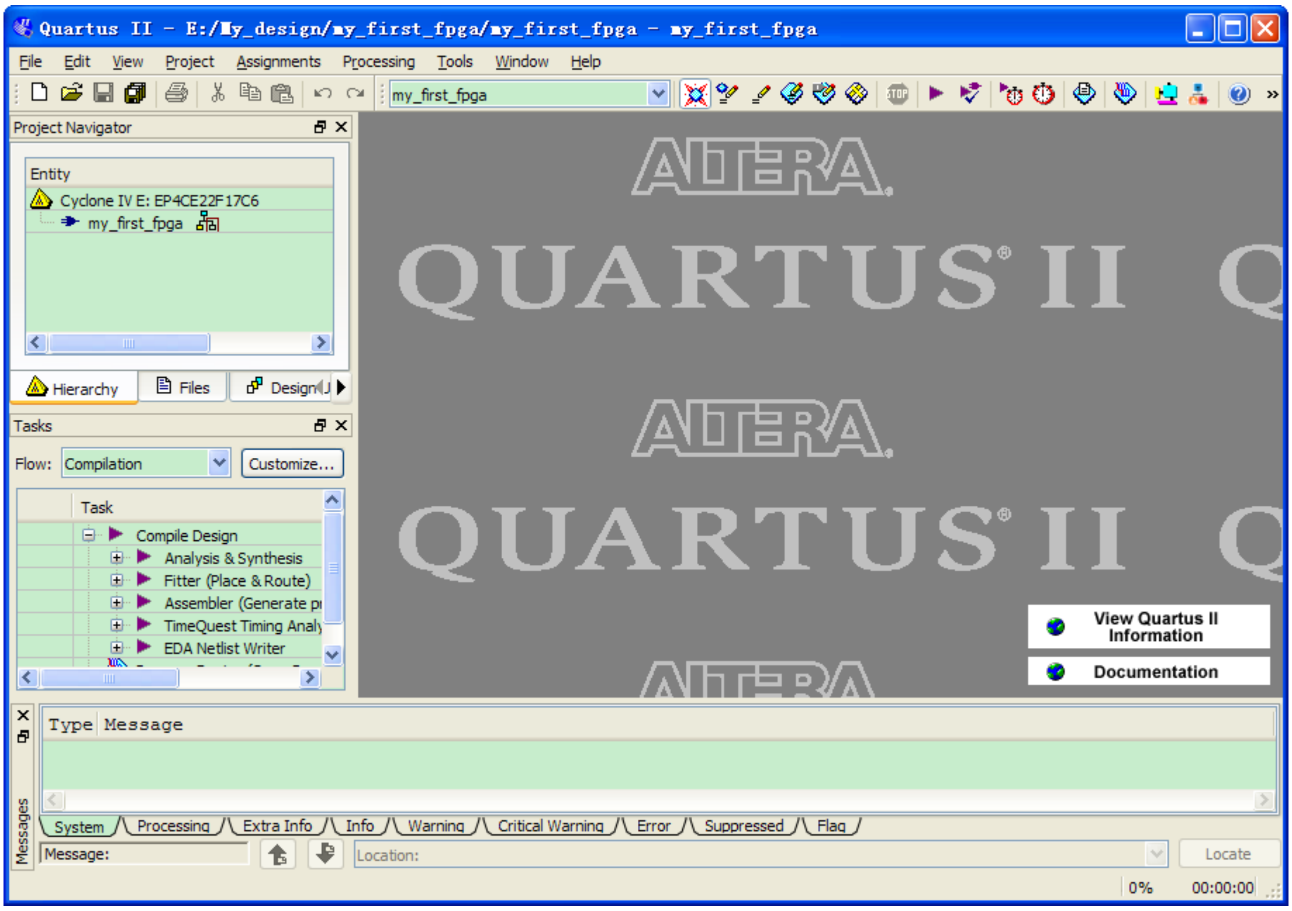


Figure 6-11 my_first_fpga project

6.5 Add a PLL Megafunction

This section describes How to Add a PLL Megafunction

In the design entry step you create a schematic or Block Design File (.bdf) that is the top-level design. You will add library of parameterized modules (LPM) functions and use Verilog HDL code to add a logic block. When creating your own designs, you can choose any of these methods or a combination of them.

1. Choose File > New > Block Diagram/Schematic File (see **Figure 6-12** to create a new file, Block1.bdf, which you will save as the top-level design.

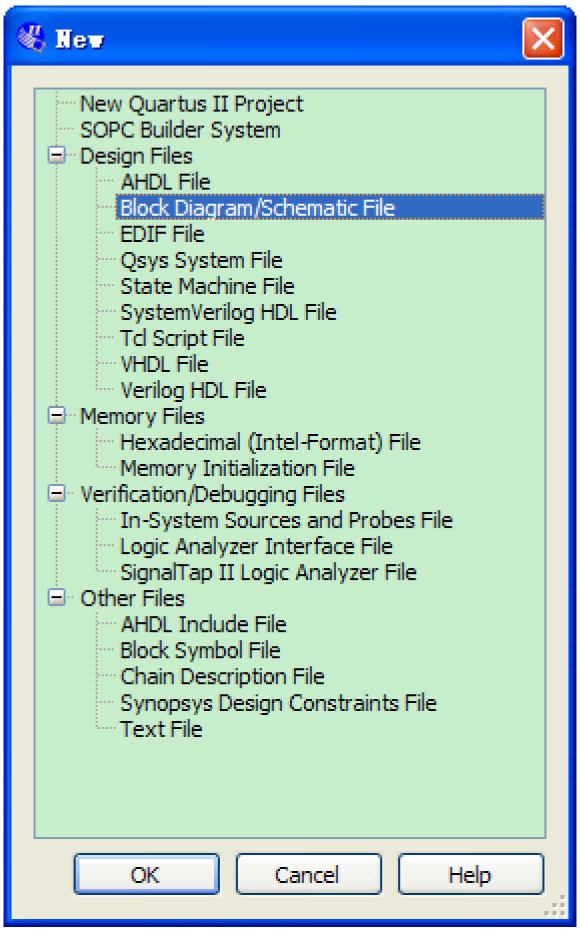


Figure 6-12 New BDF

2. Click OK.
3. Choose File > Save As and enter the following information.

   - File name: my_first_fpga
   - Save as type: Block Diagram/Schematic File (*.bdf)

4. Click Save. The new design file appears in the Block Editor (see Figure 6-13).

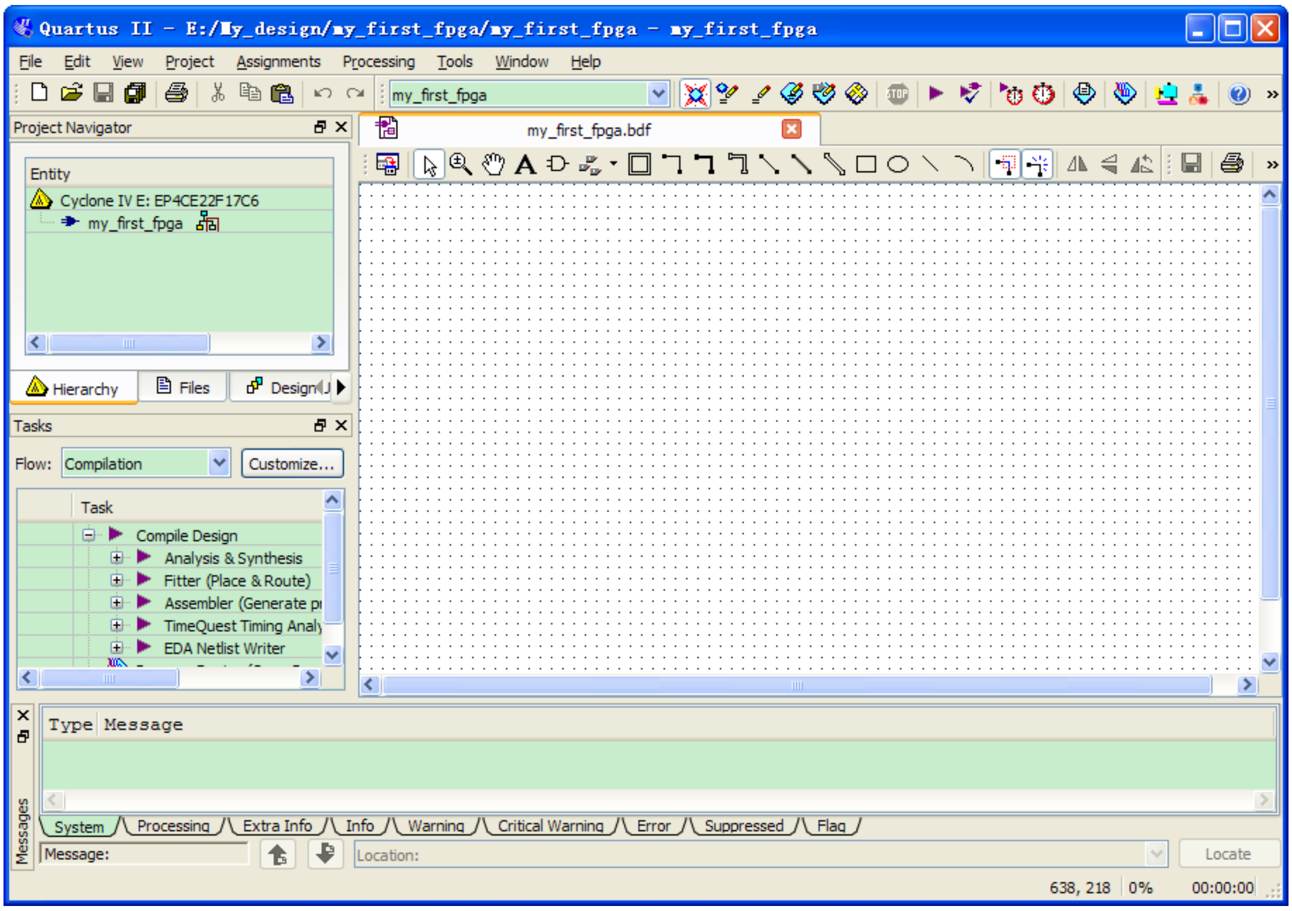


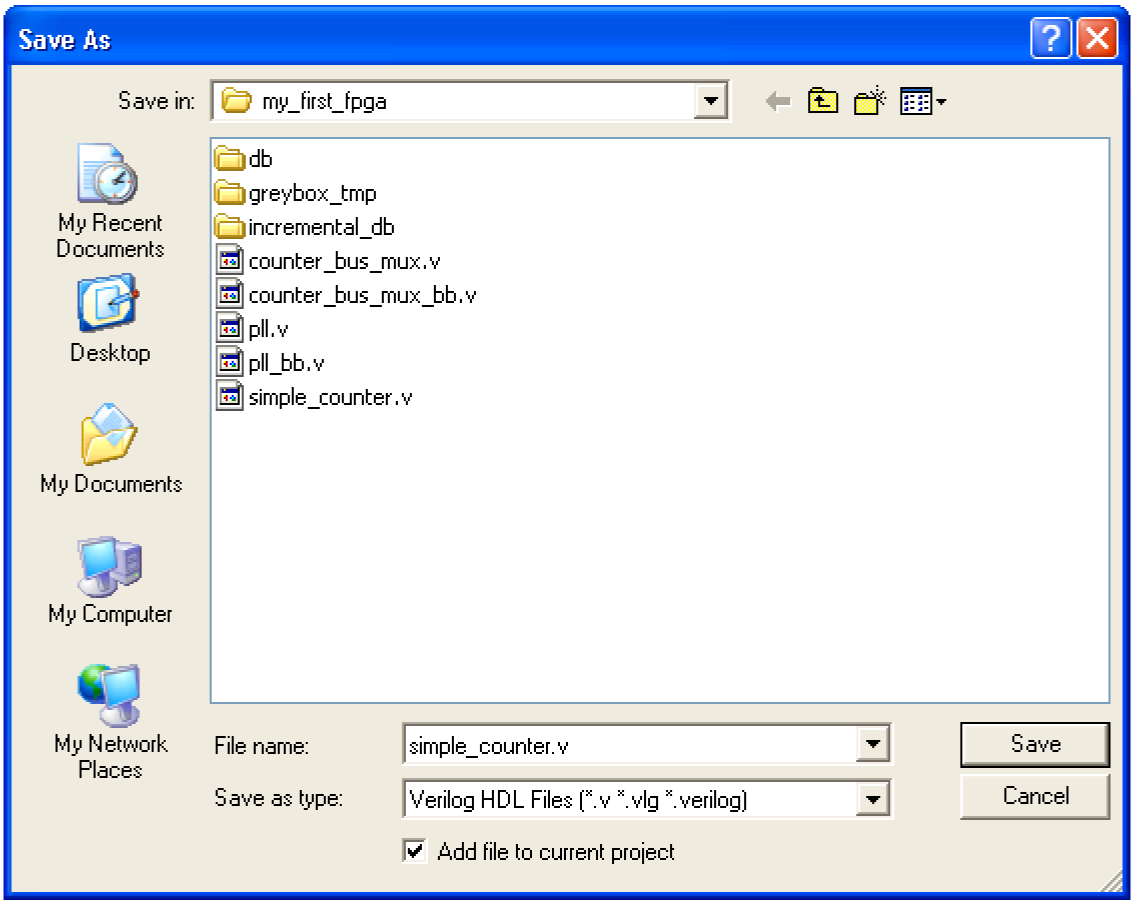
Figure 6-13 Bank BDF

5. Add HDL code to the blank block diagram by choosing File > New > Verilog HDL File.

6. Click OK to create a new file Verilog1.v, which you will save as simple_counter.v.

7. Select File > Save As and enter the following information (see Figure 6-14).

- File name: simple_counter.v
- Save as type: Verilog HDL File (*.v, *.vlg, *.verilog)



db
greybox_tmp
incremental_db
counter_bus_mux.v
counter_bus_mux_bb.v
pll.v
pll_bb.v
simple_counter.v

File name: simple_counter.v
Save as type: Verilog HDL Files (*.v *.vlg *.verilog)
☑ Add file to current project

Figure 6-14 Saving the Verilog HDL file

The resulting empty file is ready for you to enter the Verilog HDL code.

8. Type the following Verilog HDL code into the blank simple_counter.v file (see **Figure 6-15** The Verilog File of simple_counter.v).

```
//It has a single clock input and a 32-bit output port

module simple_counter (

                         CLOCK_50,

                         counter_out

                       );

input     CLOCK_50 ;

output [31:0] counter_out;

reg       [31:0] counter_out;
```

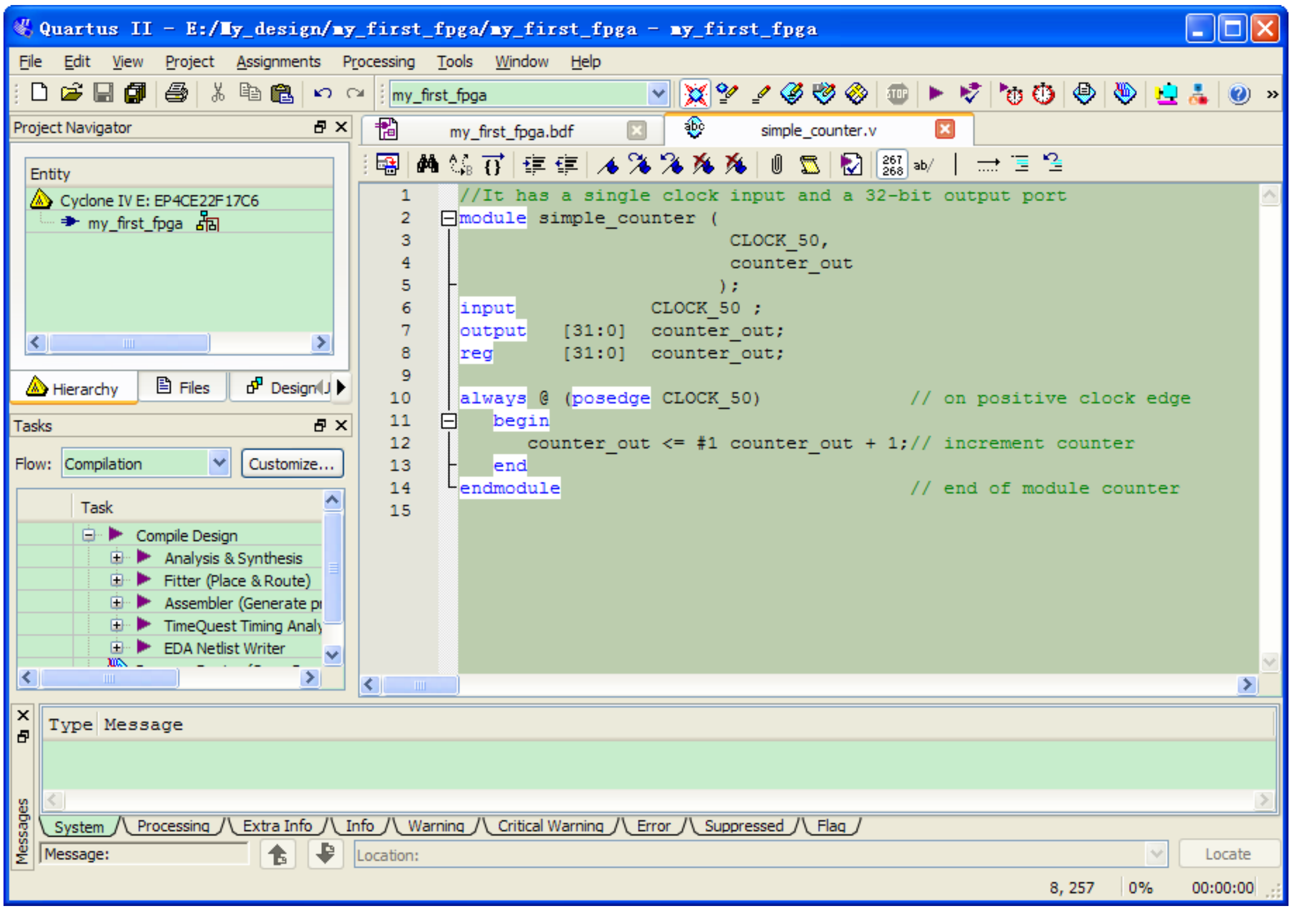
always @ (posedge CLOCK_50) // on positive clock edge

begin

counter_out <= #1 counter_out + 1;// increment counter

end

endmodule // end of module counter



```
//It has a single clock input and a 32-bit output port
module simple_counter (
                        CLOCK_50,
                        counter_out
                       );
input            CLOCK_50 ;
output    [31:0] counter_out;
reg       [31:0] counter_out;

always @ (posedge CLOCK_50)            // on positive clock edge
    begin
        counter_out <= #1 counter_out + 1;// increment counter
    end
endmodule                              // end of module counter
```

Figure 6-15 The Verilog File of simple_counter.v

9. Save the file by choosing File > Save, pressing Ctrl + S, or by clicking the floppy disk icon.
10. Choose File > Create/Update > Create Symbol Files for Current File to convert the simple_counter.v file to a Symbol File (.sym).You use this Symbol File to add the HDL code to your BDF schematic.

The Quartus II software creates a Symbol File and displays a message (see Figure 6-16).

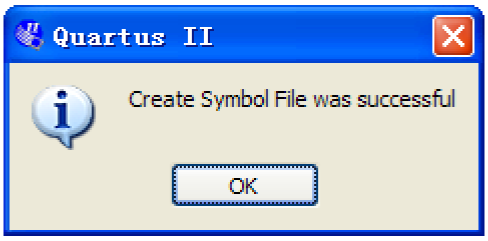


Figure 6-16 Create Symbol File was Successful

11. Click OK.
12. To add the simple_counter.v symbol to the top-level design, click the my_first_fpga.bdf tab.
13. Choose Edit > Insert Symbol.
14. Double-click the Project directory to expand it.
15. Select the newly created simple_counter symbol by clicking it's icon.

You can also double-click in a blank area of the BDF to open the Symbol dialog box (If your Quartus II version is lower than 10.0).

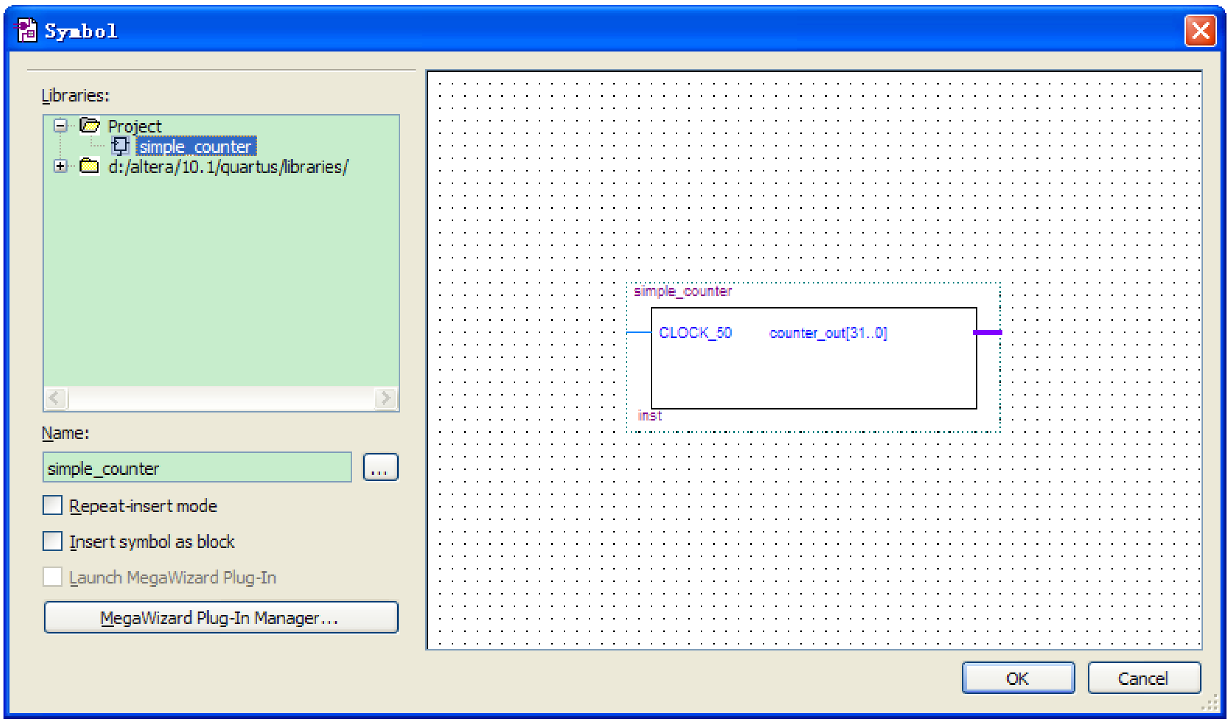


Figure 6-17 Adding the Symbol to the BDF

16. Click OK.
17. Move the cursor to the BDF grid; the symbol image moves with the cursor. Click to place the simple_counter symbol onto the BDF. You can move the block after placing it by simply clicking and dragging it to where you want it and releasing the mouse button to place it. See **Figure 6-18**.

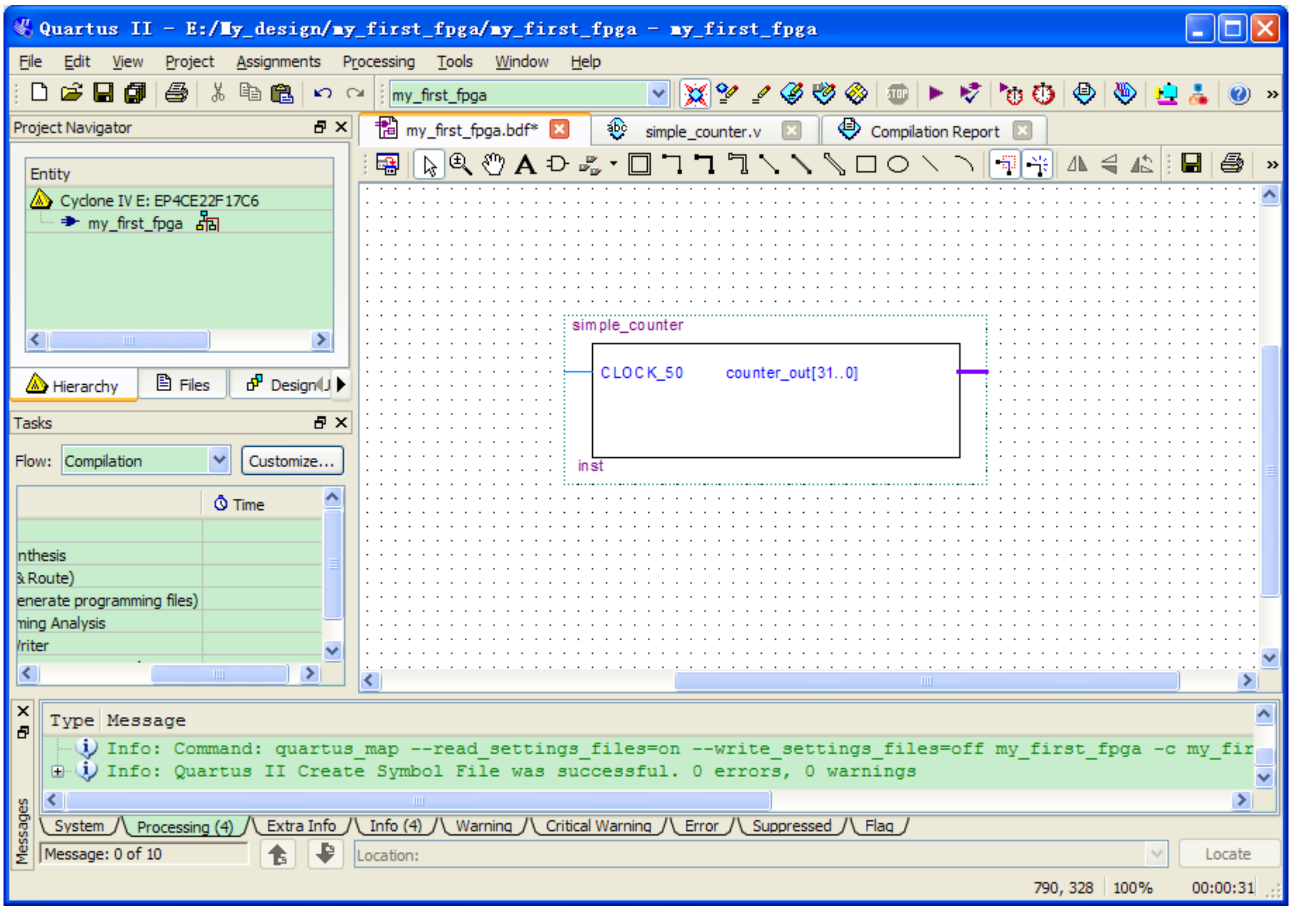


Figure 6-18 Placing the simple_counter symbol

18. Press the Esc key or click an empty place on the schematic grid to cancel placing further instances of this symbol.

19. Save your project regularly.

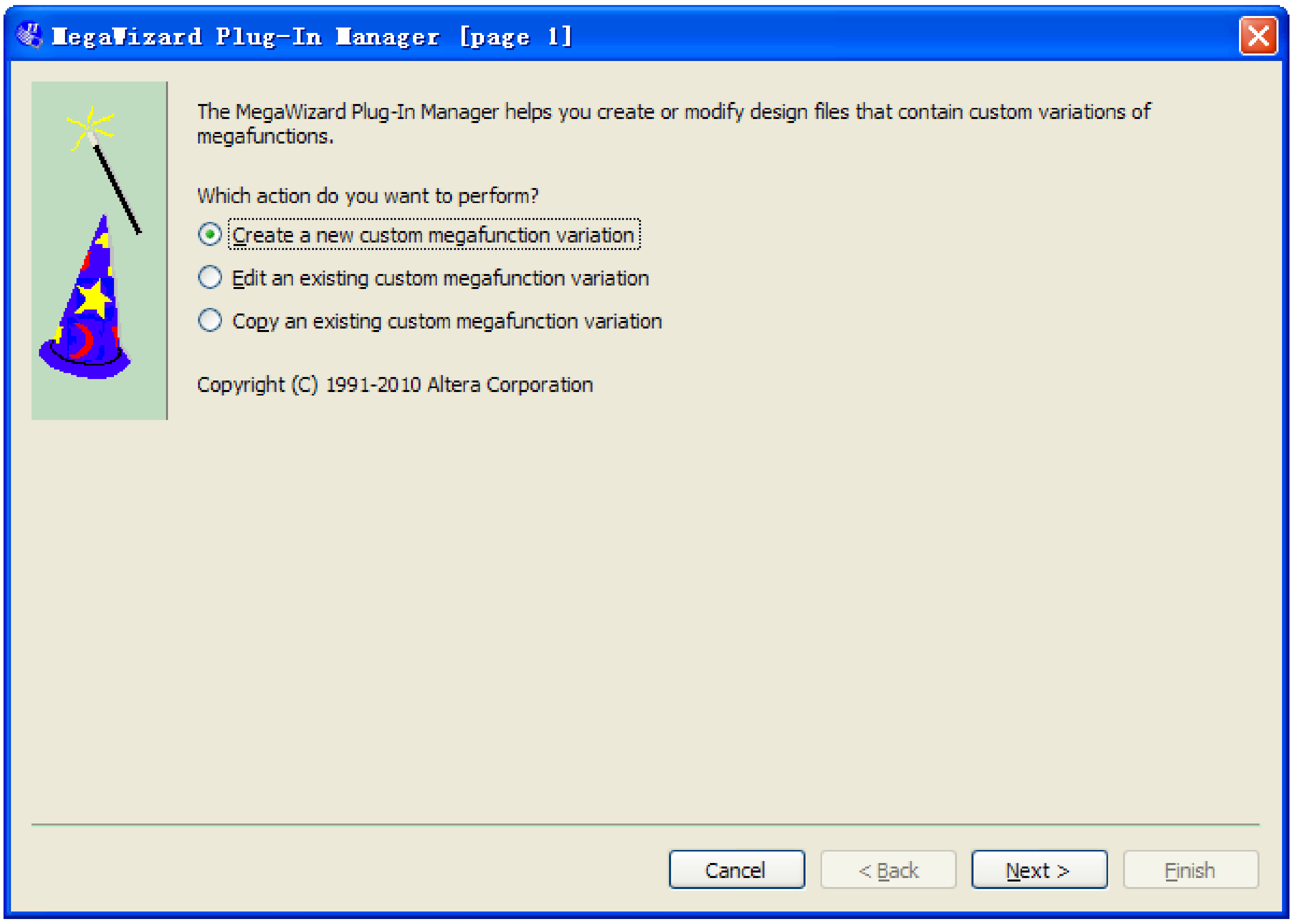
Using Quartus Add a PLL Megafunction

Megafunctions, such as the ones available in the LPM, are pre-designed modules that you can use in FPGA designs. These Altera-provided megafunctions are optimized for speed, area, and device family. You can increase

Efficiency by using a megafunction instead of writing the function yourself. Altera also provides more complex functions, called MegaCore functions, which you can evaluate for free but require a license file for use in production designs. This tutorial design uses a PLL clock source to drive a simple counter. A PLL uses the on-board oscillator (DE0-Nano Board is 50 MHz) to create a constant clock frequency as the input to the counter. To create the clock source, you will add a pre-built LPM megafunction named ALTPLL.

1. Choose Edit > Insert Symbol or click Add Symbol on the toolbar

2. Click Megawizard Plug-in Manager. The MegaWizard® Plug-In Manager appears (see Figure 6-19).


MegaWizard Plug-In Manager [page 1]

The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.

Which action do you want to perform?

[✓ Create a new custom megafunction variation]

Edit an existing custom megafunction variation

Copy an existing custom megafunction variation

Copyright (C) 1991-2010 Altera Corporation

Figure 6-19 Mega Wizard Plug-In Manager

3. Click Next.

4. In MegaWizard Plug-In Manager [page 2a], specify the following selections (see **Figure 6-20**):

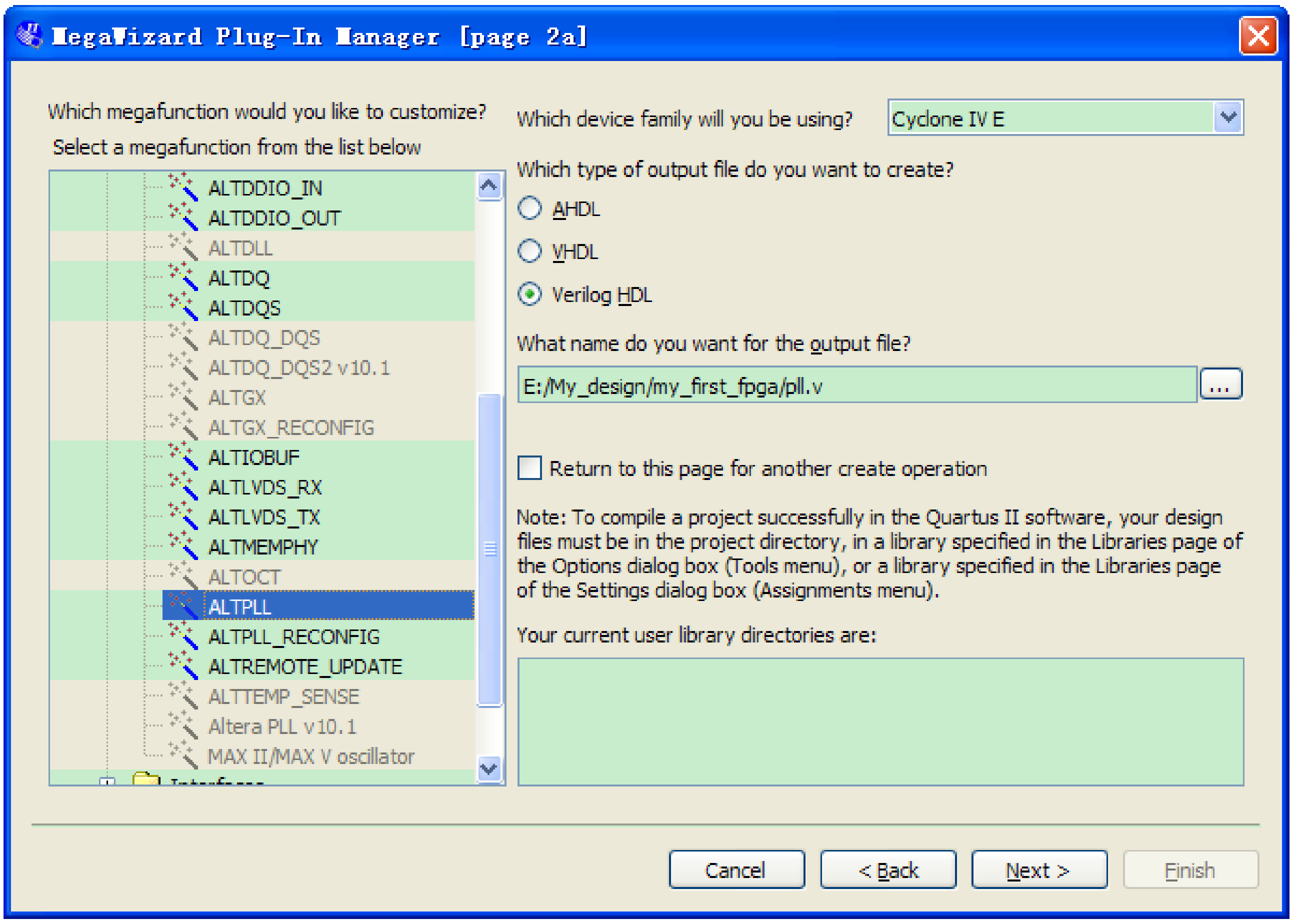
a. Choose I/O > ALTPLL.

b. Under Which device family will you be using? Choose the Cyclone IV E for DE0-Nano development board.

c. Under Which type of output file do you want to create? Choose Verilog HDL.

d. Under What name do you want for the output file? Type pll at the end of the already created directory name.

e. Click Next.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

ALTDDIO_IN
ALTDDIO_OUT
ALTDLL
ALTDQ
ALTDQS
ALTDQ_DQS
ALTDQ_DQS2 v10.1
ALTGX
ALTGX_RECONFIG
ALTIOBUF
ALTLVDS_RX
ALTLVDS_TX
ALTMEMPHY
ALTOCT
ALTPLL
ALTPLL_RECONFIG
ALTREMOTE_UPDATE
ALTTEMP_SENSE
Altera PLL v10.1
MAX II/MAX V oscillator

Which device family will you be using? Cyclone IV E

Which type of output file do you want to create?
AHDL
VHDL
[✓ Verilog HDL]

What name do you want for the output file?
E:/My_design/my_first_fpga/pll.v

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

Figure 6-20 MegaWizard Plug-In Manager [page 2a] Selections

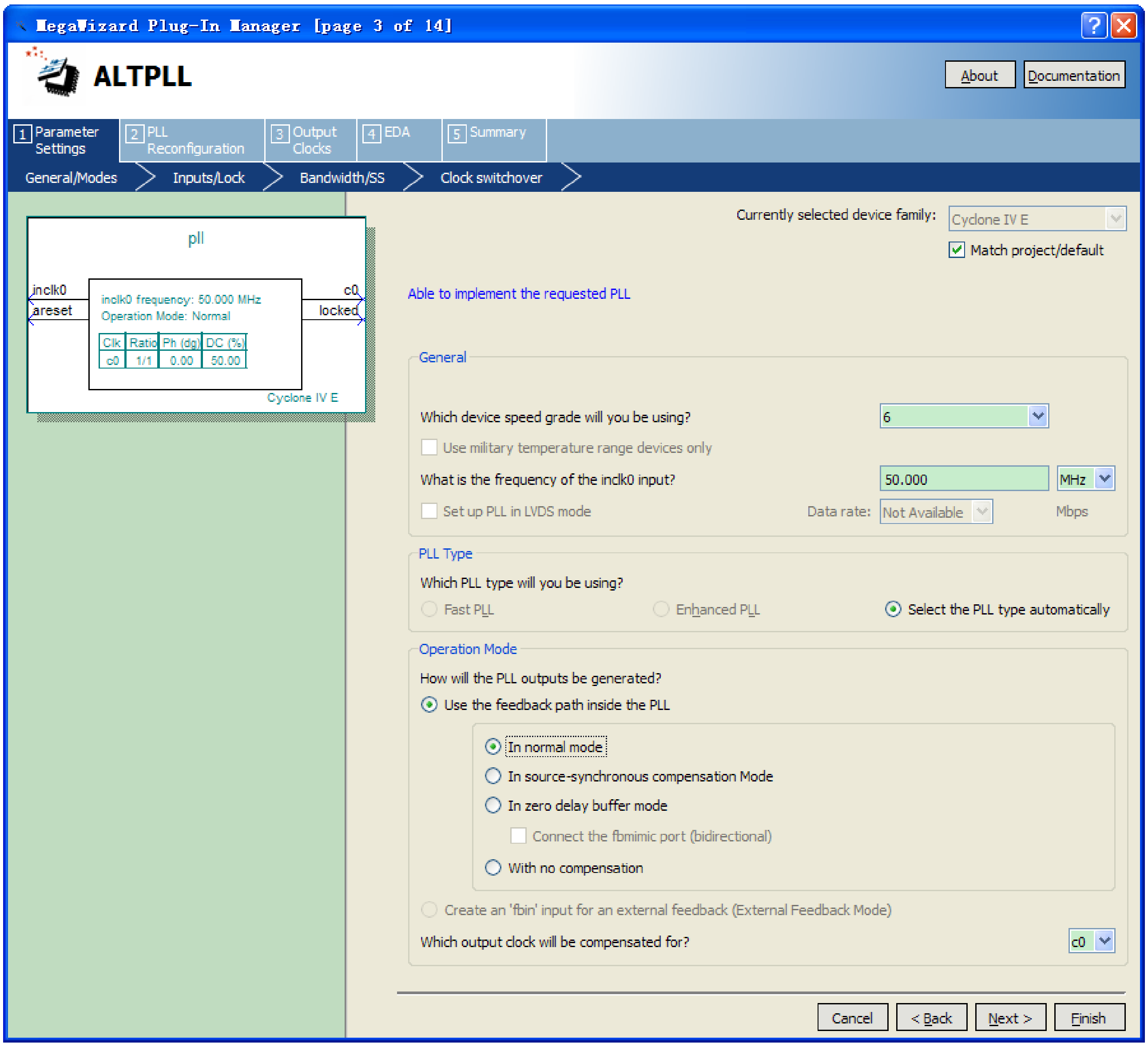
5. In the MegaWizard Plug-In Manager [page 3 of 14] window, make the following selections (see **Figure 6-21**).

a. Confirm that the Currently selected device family option shows the device family that corresponds to the development board you are using.

b. The device speed grade choose 6 for DE0-Nano.

c. Set the frequency of the inclock0 input 50 MHz.

d. Click Next.


MegaWizard Plug-In Manager [page 3 of 14]

ALTPLL

About | Documentation

1 Parameter Settings | 2 PLL Reconfiguration | 3 Output Clocks | 4 EDA | 5 Summary

General/Modes > Inputs/Lock > Bandwidth/SS > Clock switchover >

pll

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00

Currently selected device family: Cyclone IV E
[✓] Match project/default

Able to implement the requested PLL

General

Which device speed grade will you be using? 6
[ ] Use military temperature range devices only
What is the frequency of the inclk0 input? 50.000 MHz
[ ] Set up PLL in LVDS mode   Data rate: Not Available Mbps

PLL Type

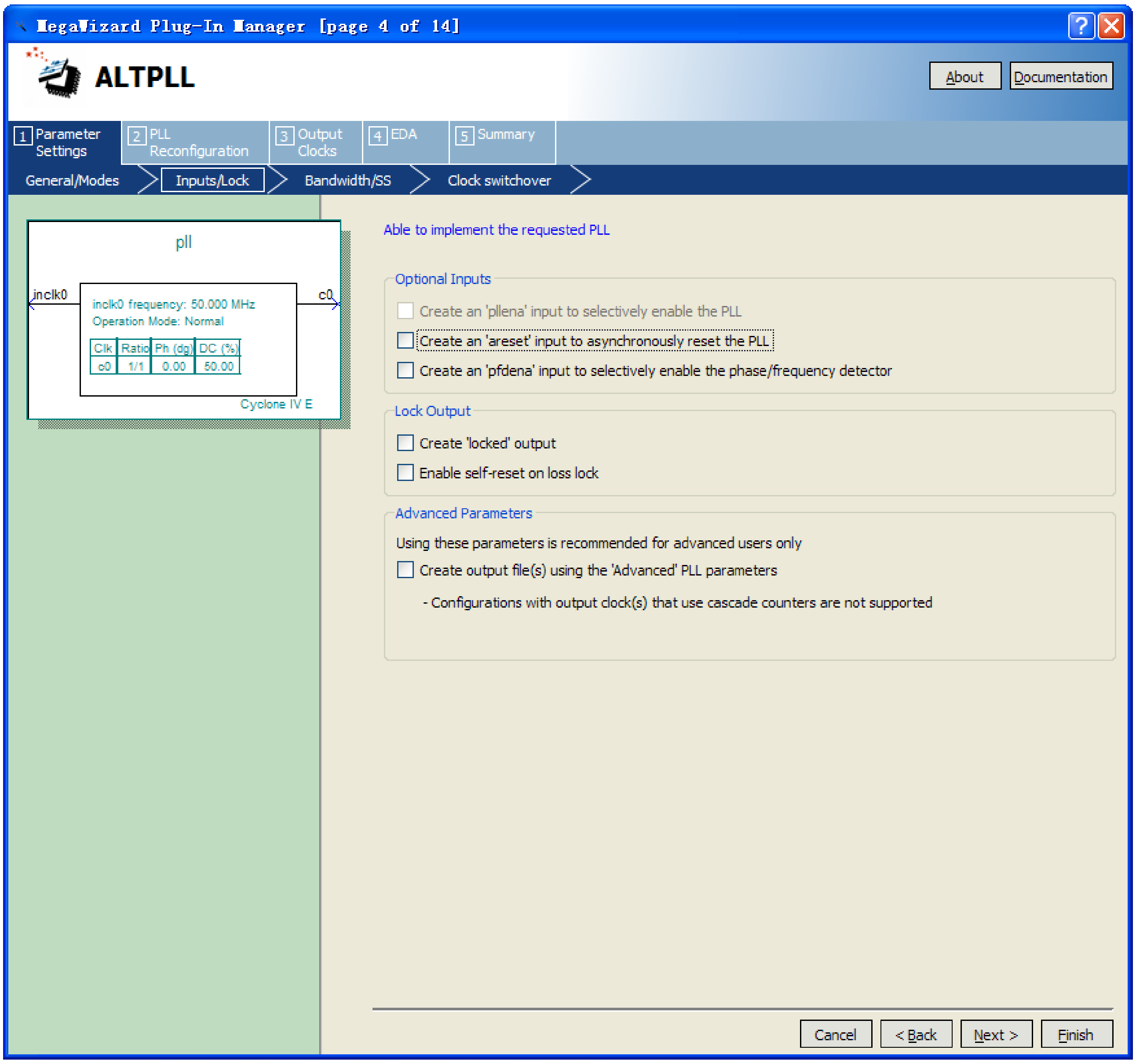
Which PLL type will you be using?
Fast PLL   Enhanced PLL   [✓ Select the PLL type automatically]

Operation Mode

How will the PLL outputs be generated?
[✓ Use the feedback path inside the PLL]
[✓ In normal mode]
In source-synchronous compensation Mode
In zero delay buffer mode
[ ] Connect the fbmimic port (bidirectional)
With no compensation
Create an 'fbin' input for an external feedback (External Feedback Mode)
Which output clock will be compensated for? c0

Figure 6-21 MegaWizard Plug-In Manager [page 3 of 14] Selections

6. Turn off all options on MegaWizard page 4. As you turn them off, pins disappear from the PLL block's graphical preview. See Figure 6-22 for an example.


MegaWizard Plug-In Manager [page 4 of 14]

ALTPLL

Able to implement the requested PLL

Optional Inputs

- [ ] Create an 'pllena' input to selectively enable the PLL
- [ ] Create an 'areset' input to asynchronously reset the PLL
- [ ] Create an 'pfdena' input to selectively enable the phase/frequency detector

Lock Output

- [ ] Create 'locked' output
- [ ] Enable self-reset on loss lock

Advanced Parameters

Using these parameters is recommended for advanced users only

- [ ] Create output file(s) using the 'Advanced' PLL parameters

- Configurations with output clock(s) that use cascade counters are not supported

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

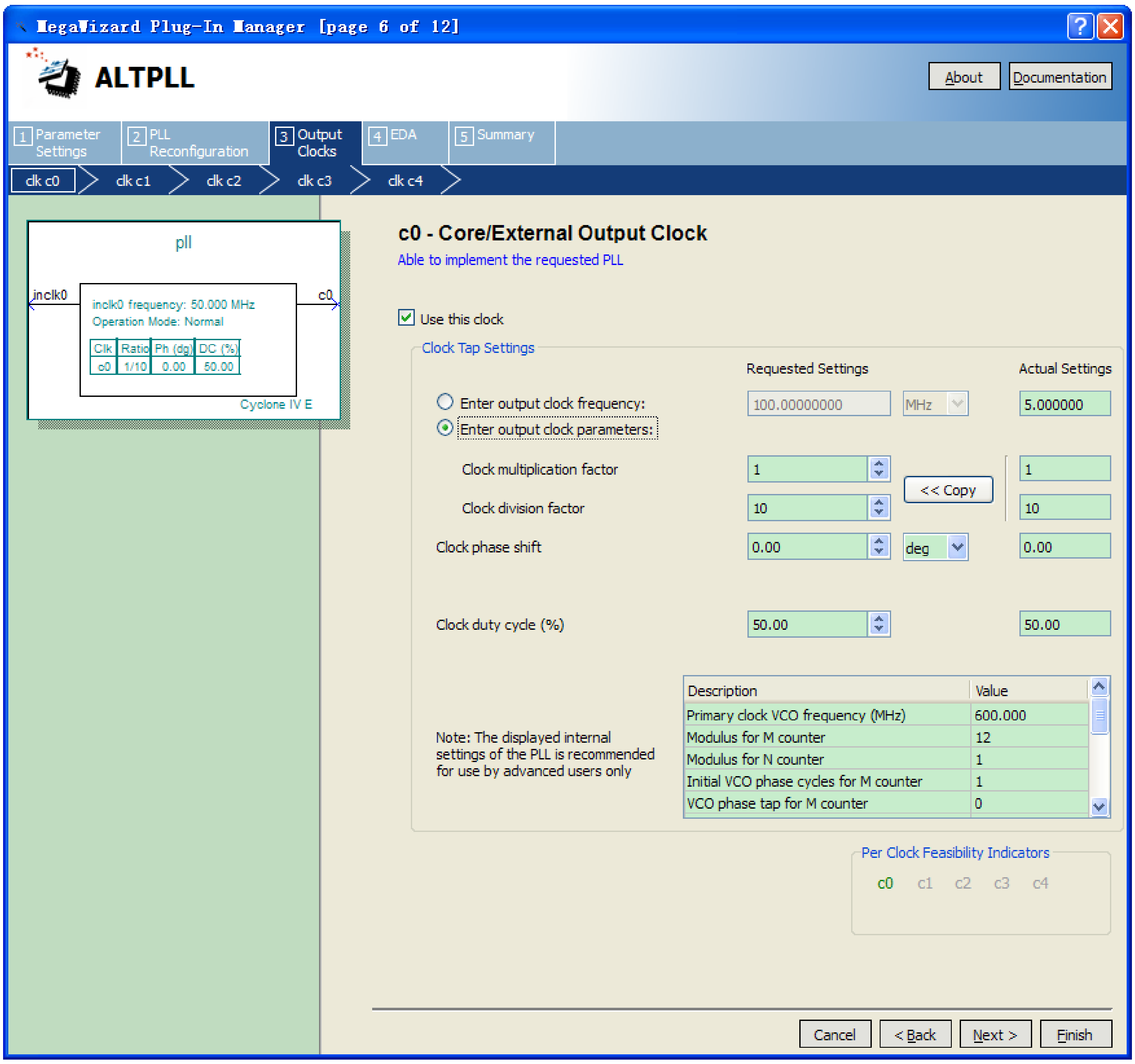
Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00

Figure 6-22 MegaWizard Plug-In Manager [page 4 of 14] Selections

7. Click Next four times.

8. At the top of the wizard, click the tab 3. Output Clocks to jump to the Output Clocks > clk c0 page

   Clock Division Settings input 10 (Figure 6-23).



Clk	Ratio	Ph (dg)	DC (%)
c0	1/10	0.00	50.00

c0 - Core/External Output Clock

Able to implement the requested PLL

[✓] Use this clock

Clock Tap Settings

	Requested Settings			Actual Settings
○ Enter output clock frequency:	100.00000000	MHz		5.000000
[✓ ● Enter output clock parameters:]				
Clock multiplication factor	1		<< Copy	1
Clock division factor	10			10
Clock phase shift	0.00	deg		0.00
Clock duty cycle (%)	50.00			50.00

Note: The displayed internal settings of the PLL is recommended for use by advanced users only

Description	Value
Primary clock VCO frequency (MHz)	600.000
Modulus for M counter	12
Modulus for N counter	1
Initial VCO phase cycles for M counter	1
VCO phase tap for M counter	0

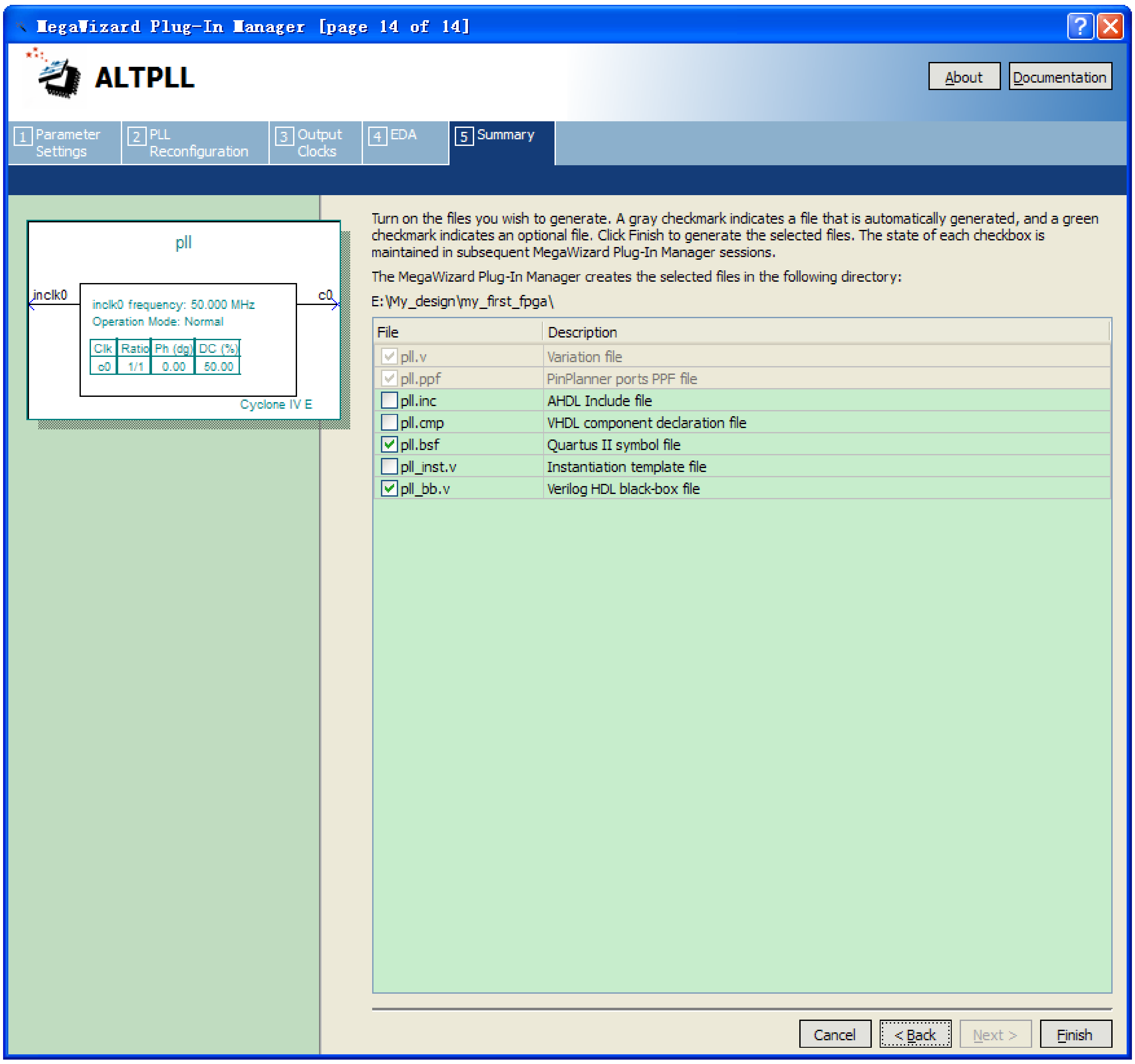
Per Clock Feasibility Indicators

c0 c1 c2 c3 c4

Figure 6-23 MegaWizard Plug-In Manager [page 8 of 14] Selections

9. Click Finish.

10. The wizard displays a summary of the files it creates (see **Figure 6-24**). Select the pll.bsf option and click Finish again.


MegaWizard Plug-In Manager [page 14 of 14]

ALTPLL

Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

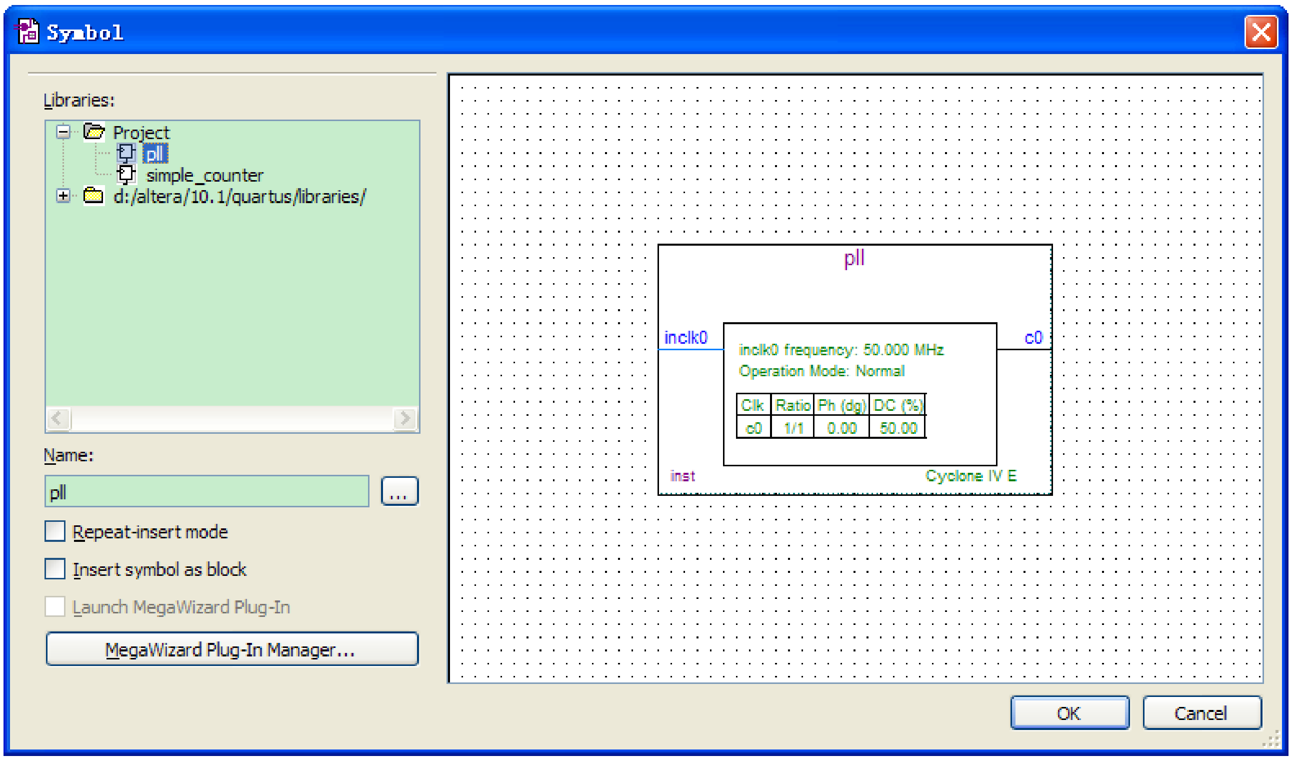
The MegaWizard Plug-In Manager creates the selected files in the following directory:

E:\My_design\my_first_fpga\

File	Description
✓ pll.v	Variation file
✓ pll.ppf	PinPlanner ports PPF file
☐ pll.inc	AHDL Include file
☐ pll.cmp	VHDL component declaration file
✓ pll.bsf	Quartus II symbol file
☐ pll_inst.v	Instantiation template file
✓ pll_bb.v	Verilog HDL black-box file

Figure 6-24 Wizard-Created Files

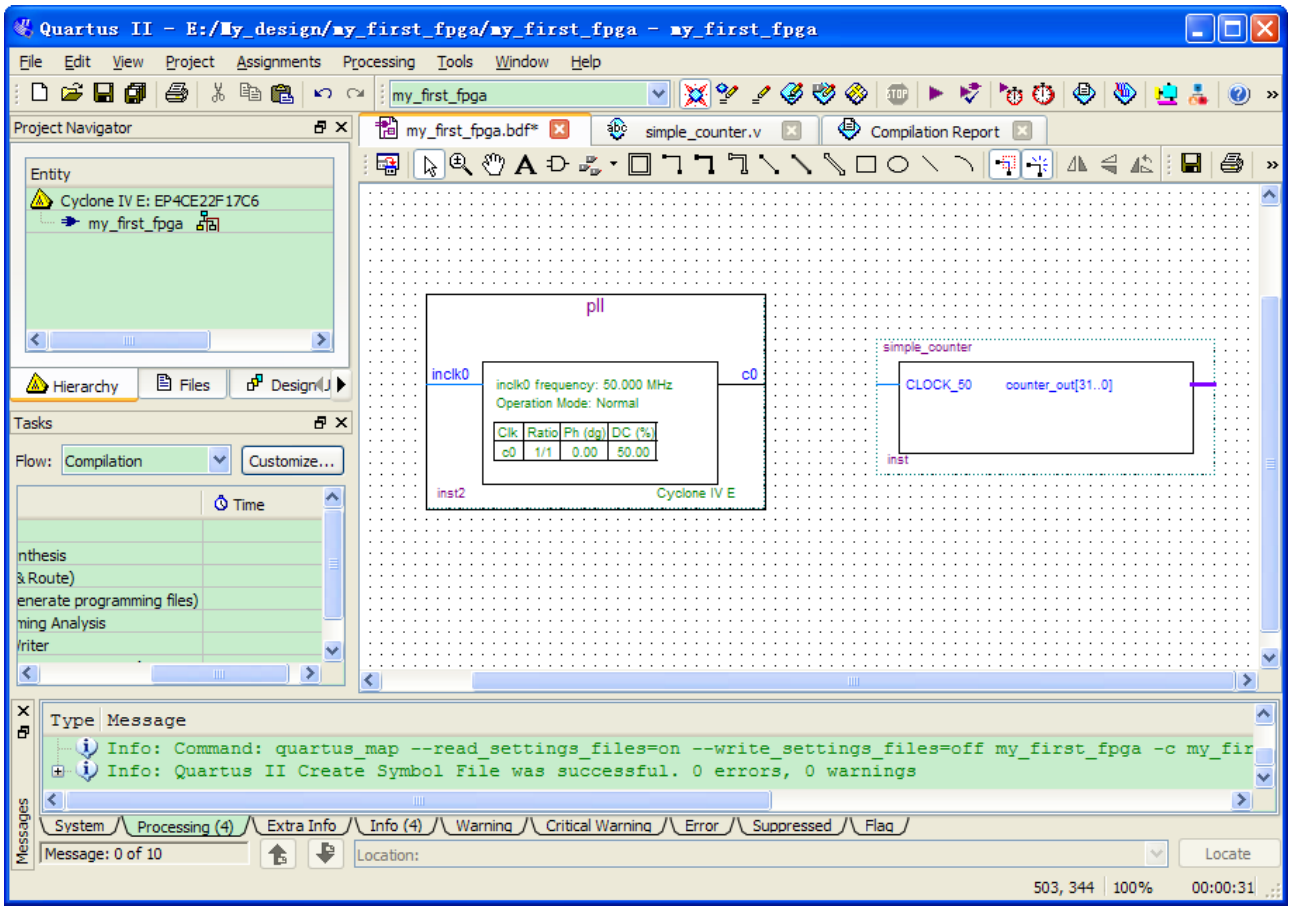
The Symbol window opens, showing the newly created PLL megafunction. See Figure 6-25.



Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00

Figure 6-25 PLL Symbol

11. Click OK and place the pll symbol onto the BDF to the left of the simple_counter symbol. You can move the symbols around by holding down the left mouse button, helping you ensure that they line up properly. See **Figure 6-26**.



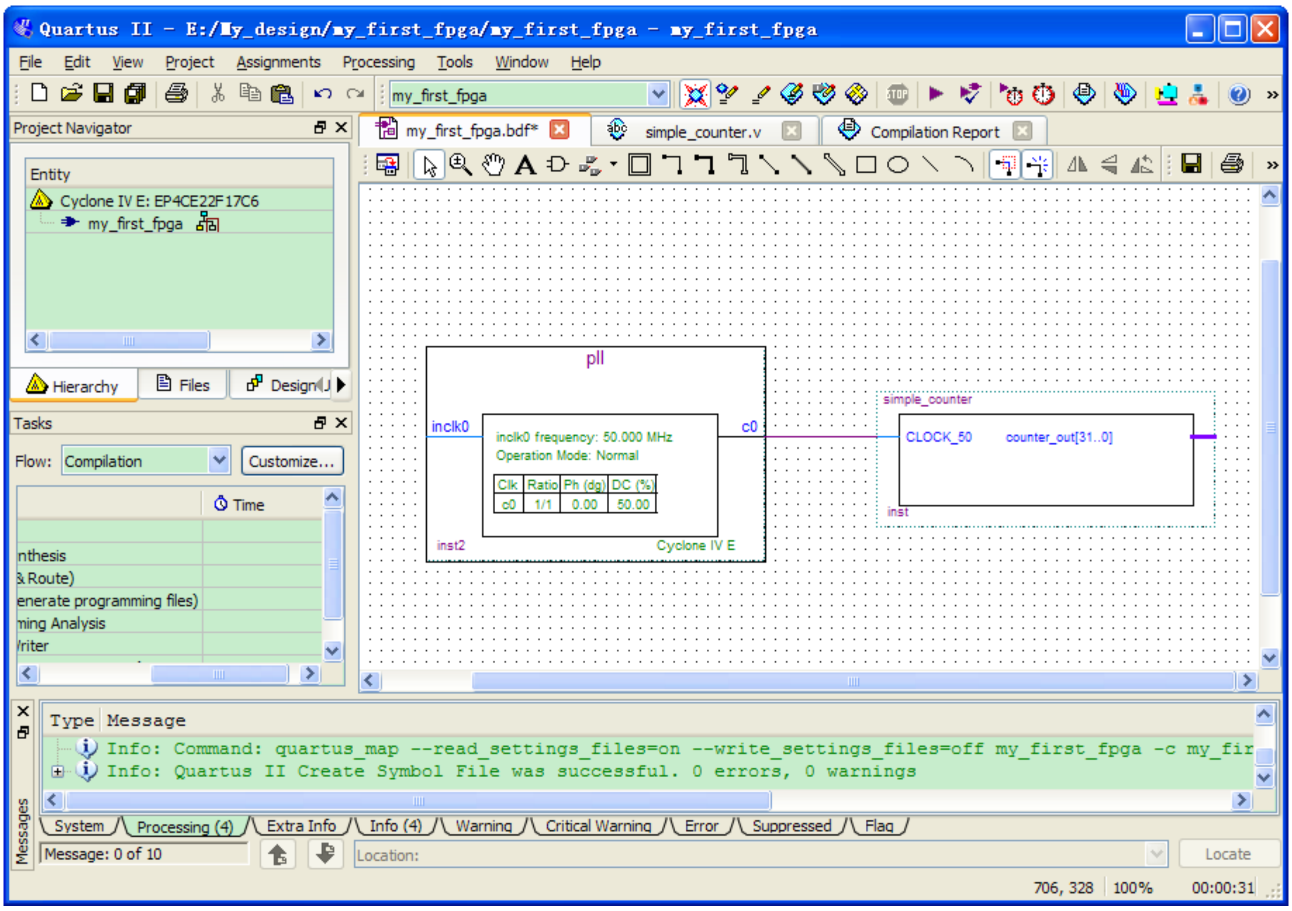
Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00

```
Type Message
  Info: Command: quartus_map --read_settings_files=on --write_settings_files=off my_first_fpga -c my_fir
  Info: Quartus II Create Symbol File was successful. 0 errors, 0 warnings
```

Figure 6-26 Place the PLL Symbol

12. Move the mouse so that the cursor (also called the selection tool) is over the pll symbol's c0 output pin. The orthogonal node tool (cross-hair) icon appears.

13. Click and drag a bus line from the c0 output to the simple_counter clock input. This action ties the pll output to the simple_counter input (see Figure 6-27).



Clk	Ratio	Ph (dg)	DC (%)
c0	1/1	0.00	50.00

```
Type Message
Info: Command: quartus_map --read_settings_files=on --write_settings_files=off my_first_fpga -c my_fir
Info: Quartus II Create Symbol File was successful. 0 errors, 0 warnings
```

Figure 6-27 Draw a Bus Line connect pll c0 port to simple_counter CLOCK_50 port

14. Add an input pin and an output bus with the following steps:

a. Choose Edit > Insert Symbol.

b. Under Libraries, select quartus/libraries > primitives > pin >input. See **Figure 6-28**

c. Click OK

If you need more room to place symbols, you can use the vertical and horizontal scroll bars at the edges of the BDF window to view more drawing space.

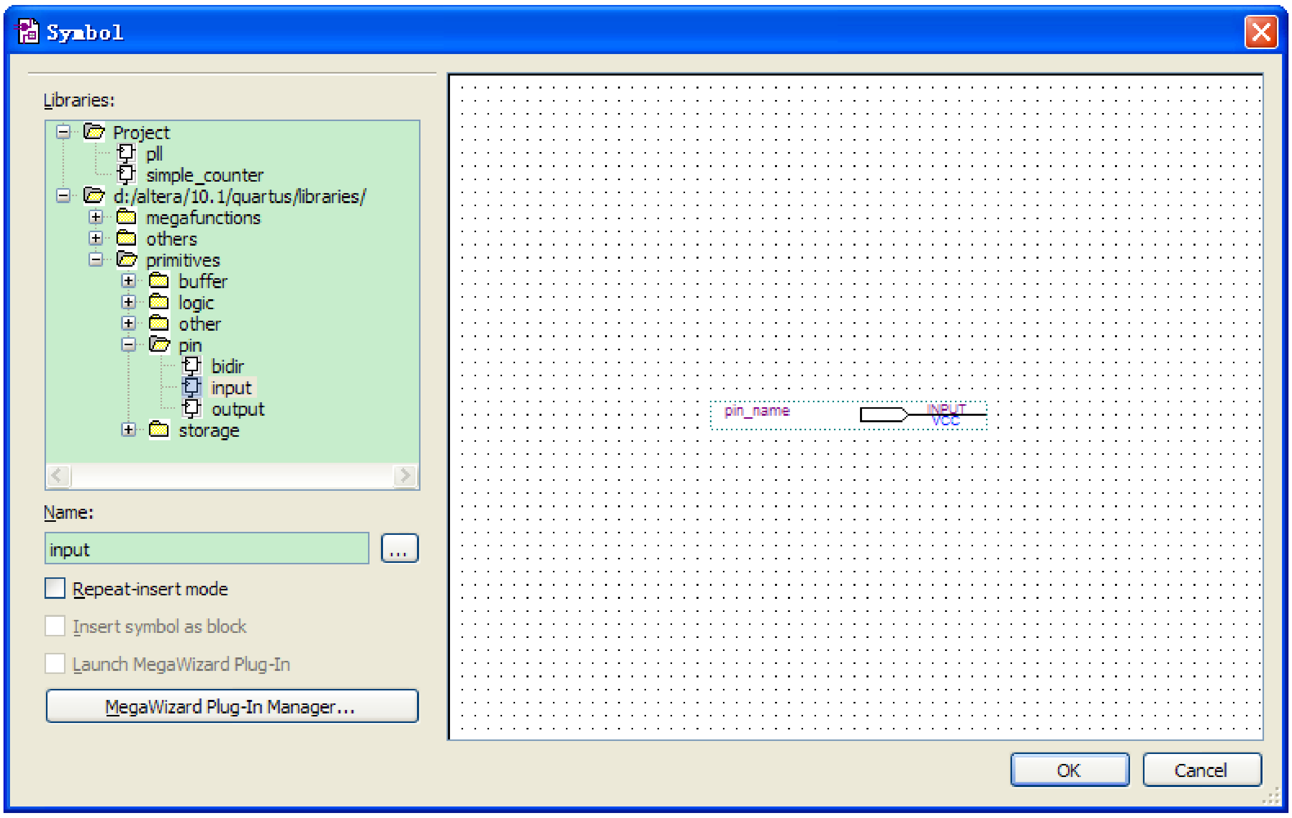


Figure 6-28 Input pin symbol

d. Place the new pin onto the BDF so that it is touching the input to the pll symbol.

e. Use the mouse to click and drag the new input pin to the left; notice that the ports remain connected as shown in **Figure 6-29**.

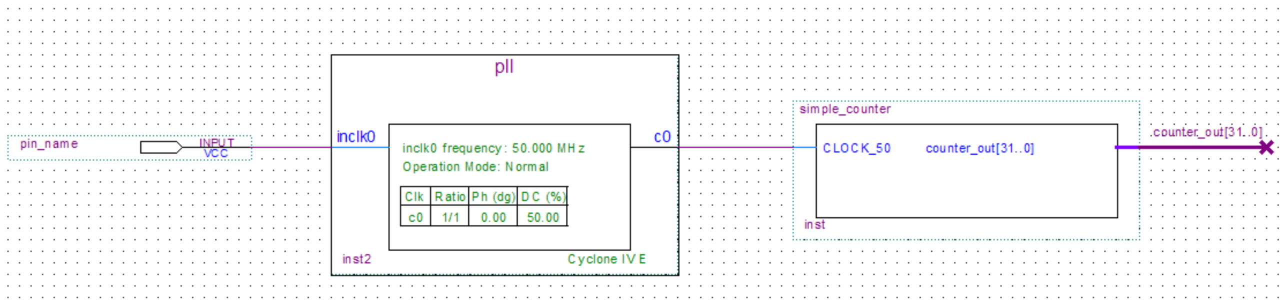
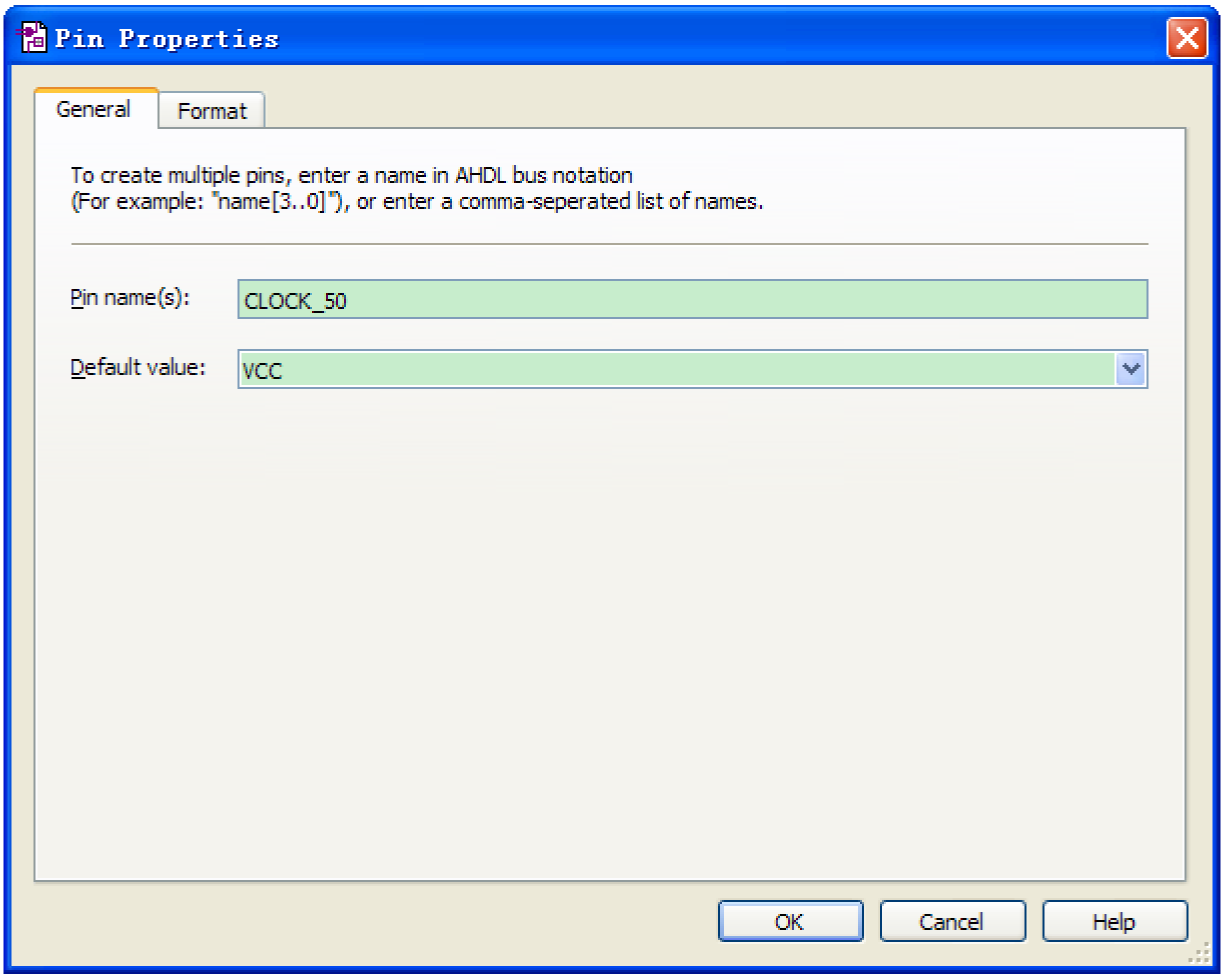


Figure 6-29 Connecting the PLL symbol and Input port

f. Change the pin name by double-clicking pin_name and typing CLOCK_50 (see **Figure 6-30**). This name correlates to the oscillator clock that is connected to the FPGA.

g. Using the Orthogonal Bus tool, draw a bus line connected on one side to the simple_counter output port, and leave the other end unconnected at about 4 to 8 grid spaces to the right of the simple_counter.



To create multiple pins, enter a name in AHDL bus notation
(For example: "name[3..0]"), or enter a comma-seperated list of names.

Pin name(s): CLOCK_50

Default value: VCC

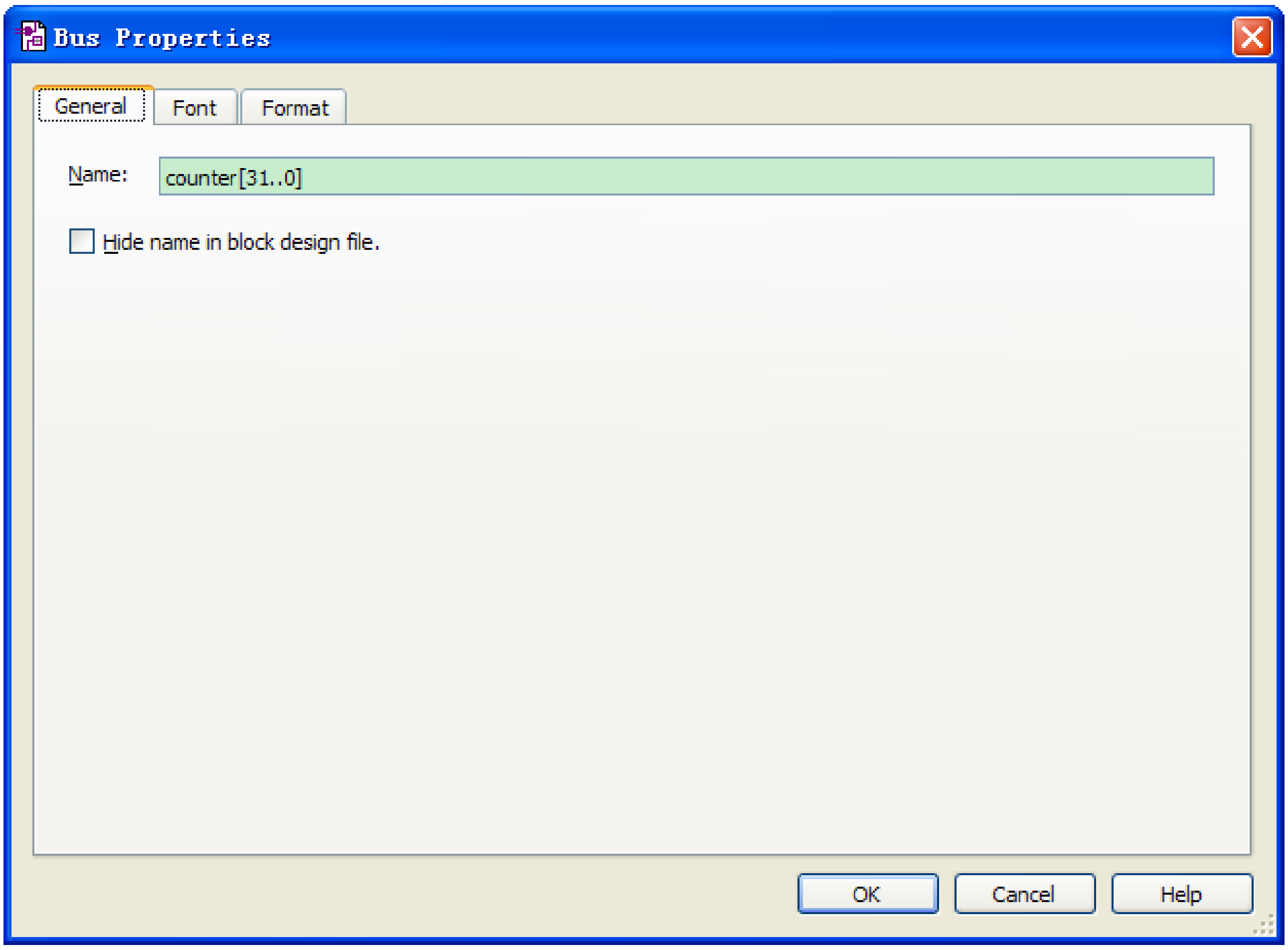
Figure 6-30 Change the input port name

h. Right-click the new output bus line and choose Properties.

i. Using the Orthogonal Bus tool, draw a bus line connected on one side to the simple_counter output port, and leave the other end unconnected at about 6 to 8 grid spaces to the right of the simple_counter.

j. Type counter [31..0] as the bus name (see **Figure 6-31**). The notation [X ..Y] is the Quartus II method for specifying the bus width in BDF schematics, where X is the most significant bit (MSB) and Y is the least significant bit (LSB).

k. Click OK. **Figure 6-32** shows the BDF.


Bus Properties

General | Font | Format

Name: counter[31..0]

☐ Hide name in block design file.

OK | Cancel | Help

Figure 6-31 Change the output BUS name

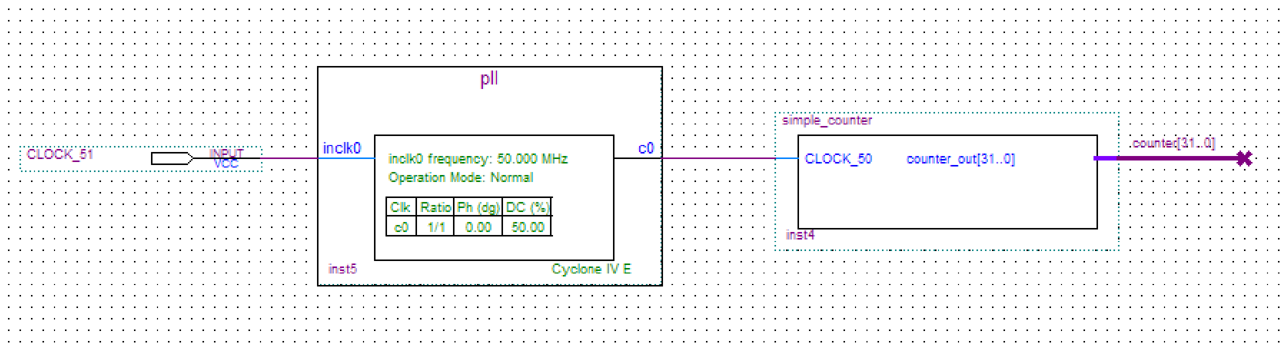
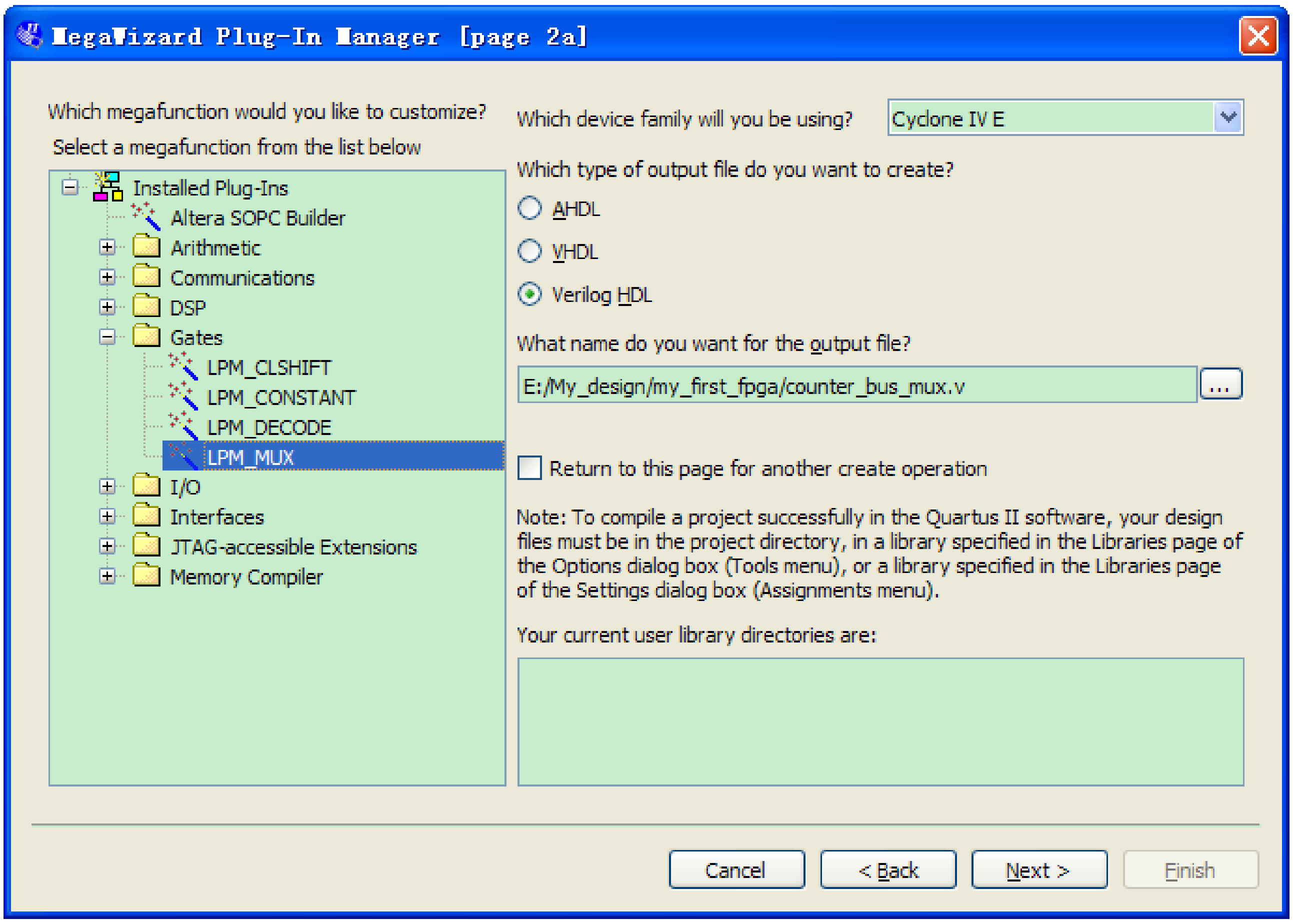


Figure 6-32 BDF

6.6 Add a Multiplexer

This design uses a multiplexer to route the simple_counter output to the LED pins on the DE0-Nano development board. You will use the MegaWizard Plug-In Manager to add the multiplexer, lpm_mux. The design multiplexes two variations of the counter bus to four LEDs on the DE0-Nano development board.

1. Choose Edit > Insert Symbol.
2. Click Megawizard Plug-in Manager.
3. Click Next.
4. Choose Installed Plug-Ins > Gates > LPM_MUX.
5. Choose the device family that corresponds to the device on the development board you are using, choose Verilog HDL as the output file type, and name the output file counter_bus_mux.v (see **Figure 6-33**).
6. Click Next.


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?
Select a megafunction from the list below

- Installed Plug-Ins
  - Altera SOPC Builder
  - Arithmetic
  - Communications
  - DSP
  - Gates
    - LPM_CLSHIFT
    - LPM_CONSTANT
    - LPM_DECODE
    - LPM_MUX
  - I/O
  - Interfaces
  - JTAG-accessible Extensions
  - Memory Compiler

Which device family will you be using? Cyclone IV E

Which type of output file do you want to create?

AHDL / VHDL / [✓ Verilog HDL]

What name do you want for the output file? E:/My_design/my_first_fpga/counter_bus_mux.v

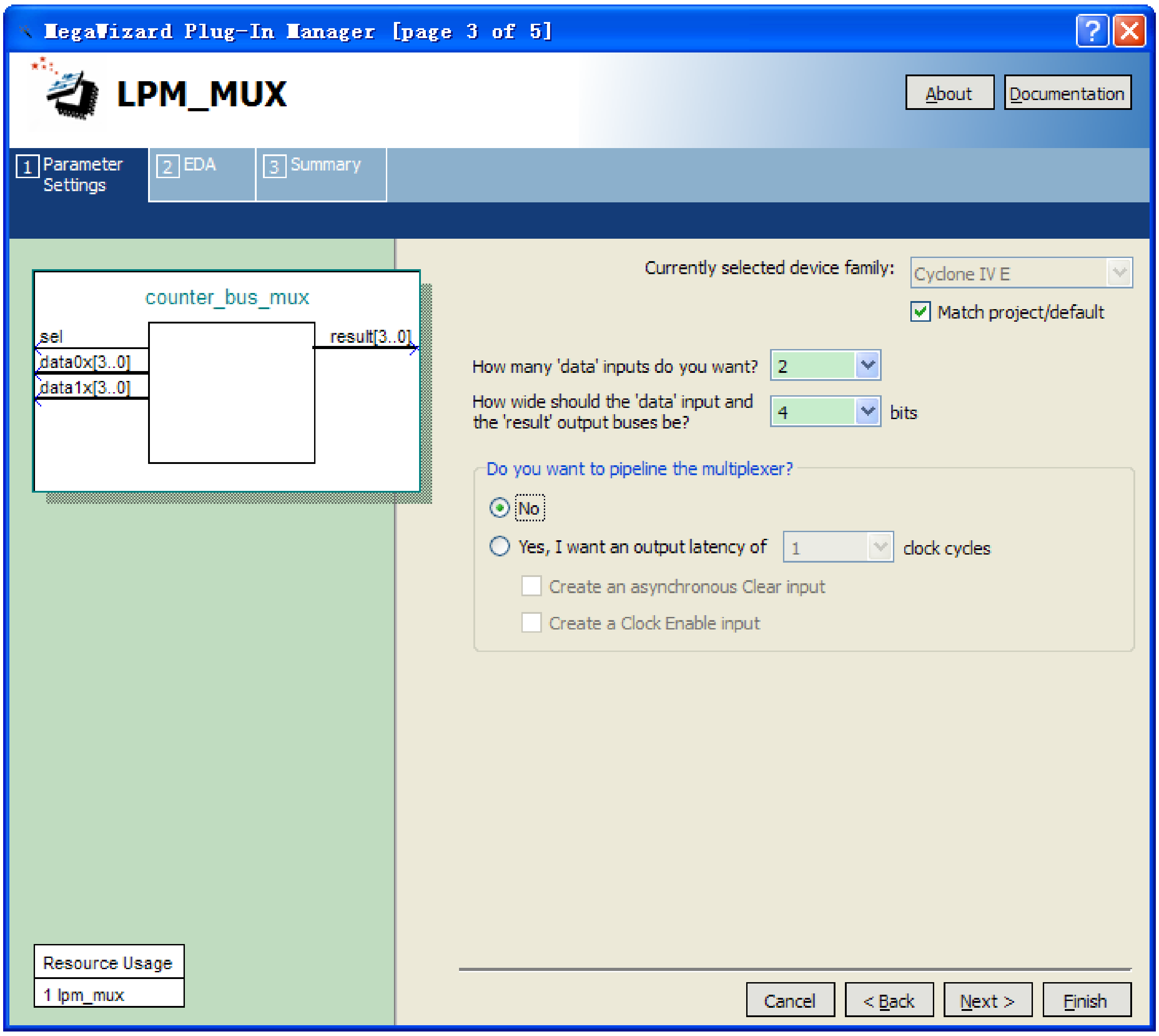
Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in a library specified in the Libraries page of the Options dialog box (Tools menu), or a library specified in the Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

Figure 6-33 Selecting lpm_mux

7. Under How many 'data' inputs do you want? Select 2 inputs (default).
8. Under How 'wide' should the data input and result output be? Select 4 (see **Figure 6-34**).



MegaWizard Plug-In Manager [page 3 of 5]

LPM_MUX

Currently selected device family: Cyclone IV E

✓ Match project/default

How many 'data' inputs do you want? 2

How wide should the 'data' input and the 'result' output buses be? 4 bits

Do you want to pipeline the multiplexer?

[✓ No]

Yes, I want an output latency of 1 clock cycles

Create an asynchronous Clear input

Create a Clock Enable input

Figure 6-34 lpm_mux settings

9. Click Next.

10. Click Next, select the counter_bus_mux.bsf options.

11. Click Finish. The Symbol window appears (see Figure 6-35 for an example).

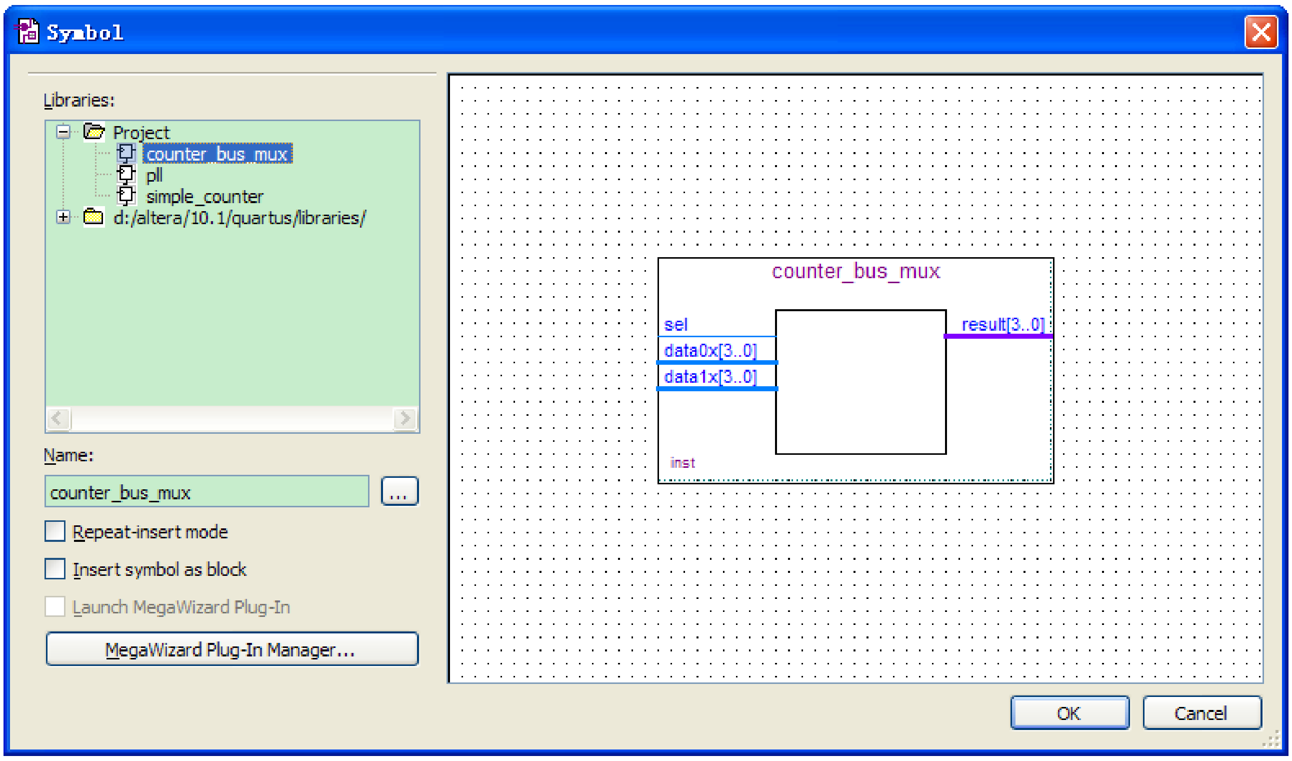


Figure 6-35 lpm_mux Symbol

12. Click OK

13. Place the counter_bus_mux symbol below the existing symbols on the BDF. See **Figure 6-36**.

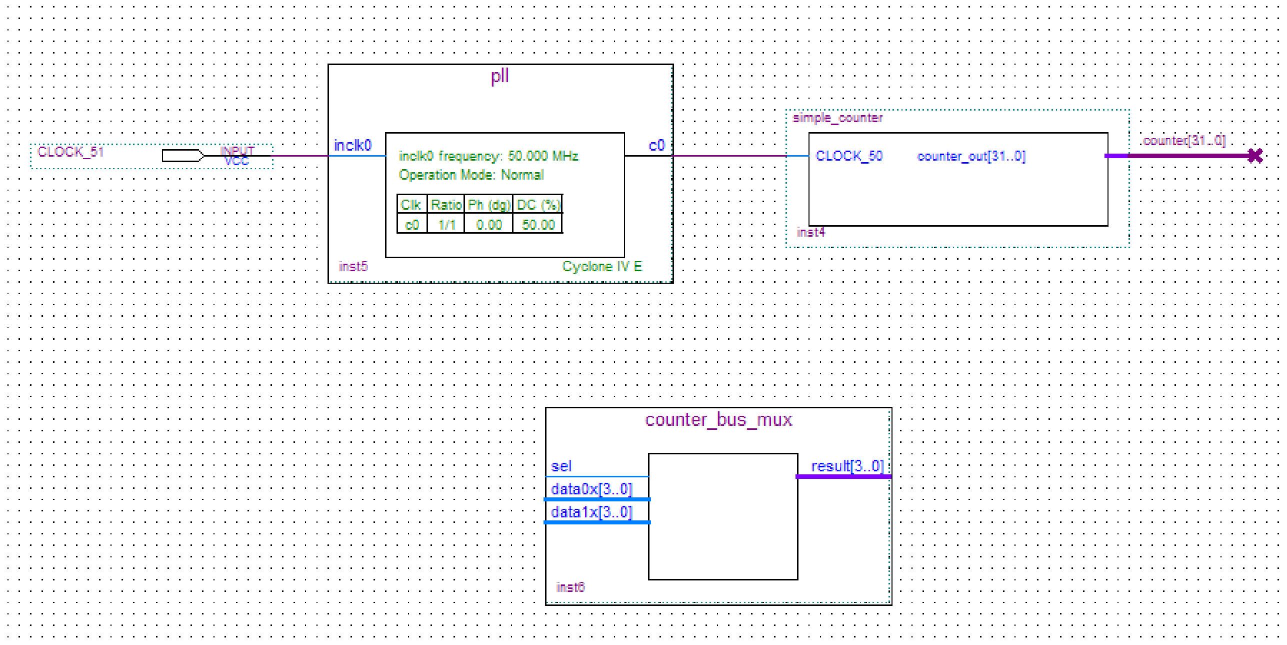


Figure 6-36 Place the lpm_mux symbol

14. Add input buses and output pins to the counter_bus_mux symbol as follows:

a. Using the Orthogonal Bus tool, draw bus lines from the data1x[3..0] and data0x[3..0]

Input ports to about 8 to 12 grid spaces to the left of counter_bus_mux.

b. Draw a bus line from the result [3..0] output port to about 4 to 8 grid spaces to the right of counter_bus_mux.

c. Right-click the bus line connected to data1x[3..0] and choose Properties.

d. Name the bus counter[26..23], which selects only those counter output bits to connect to

the four bits of the data1x input.

Because the input busses to counter_bus_mux have the same names as the output bus from simple_counter, (counter[x .. y]) the Quartus II software knows to connect these busses.

e. Click OK.

f. Right-click the bus line connected to data0x[3..0] and choose Properties.

g. Name the bus counter [24..21], which selects only those counter output bits to connect to the four bits of the data1x input.

h. Click OK. Figure 6-37 shows the renamed buses.

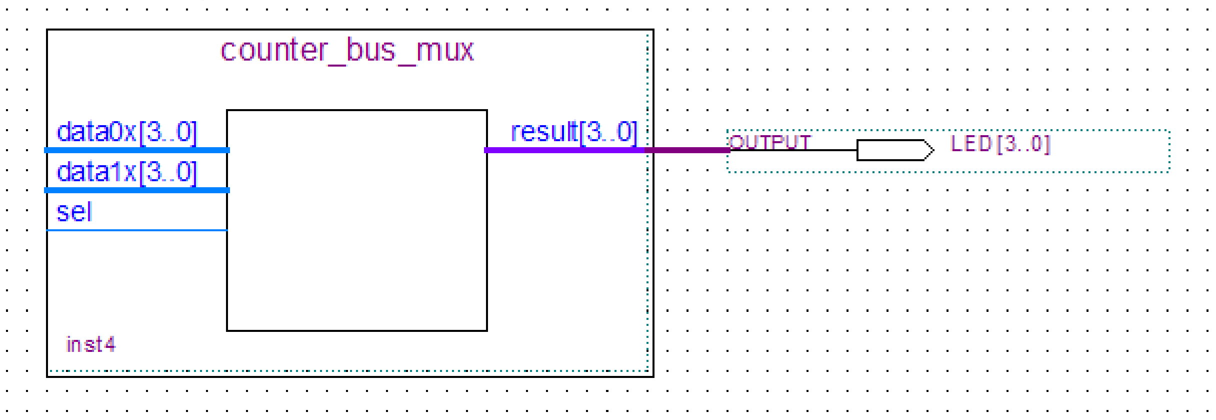


Figure 6-37 Renamed counter_bus_mux Bus Lines

If you have not done so already, save your project file before continuing.

15. Choose Edit > Insert Symbol.

16. Under Libraries, double-click quartus/libraries/ > primitives > pin > output (see Figure 6-38).

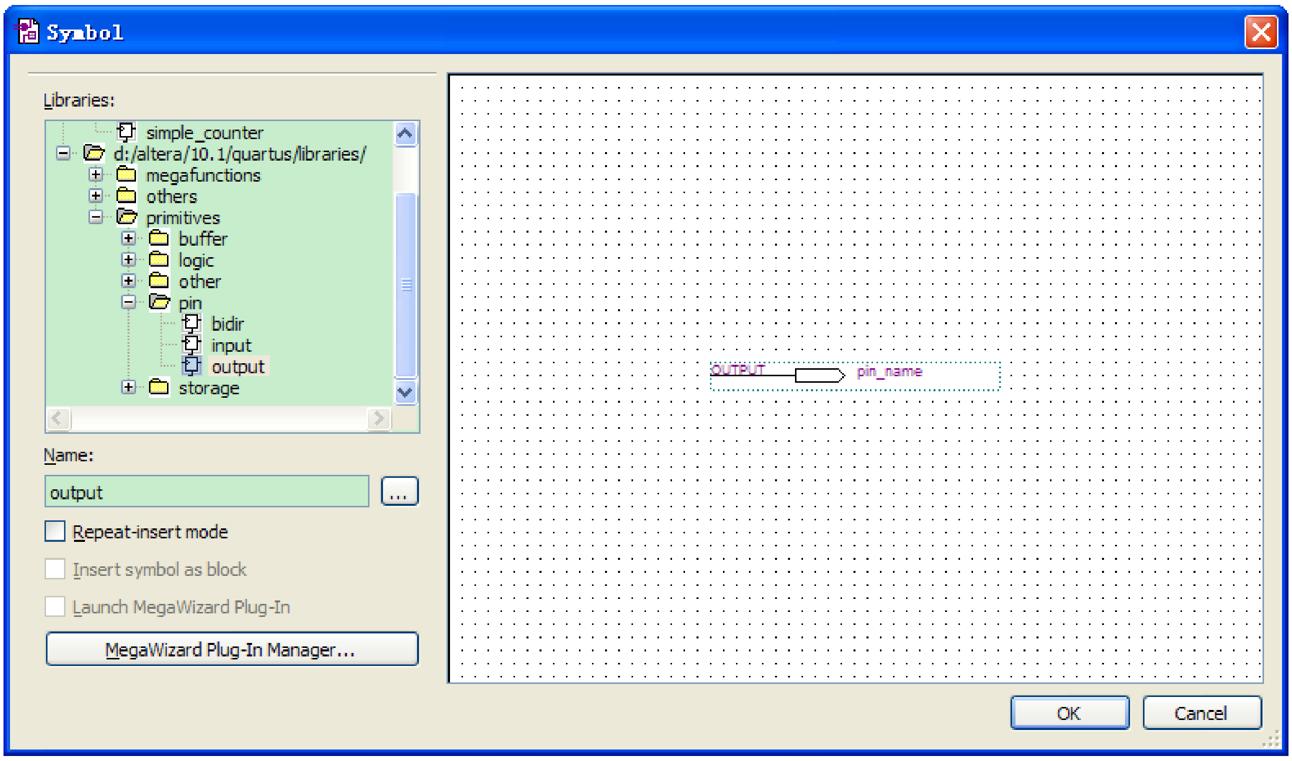


Figure 6-38 choose an output pin

17. Click OK.

18. Place this output pin so that it connects to the counter_bus_mux result [3..0] bus output line.

19. Rename the output pin as LED [3..0] as described in steps 13 c and d. (see Figure 6-39).

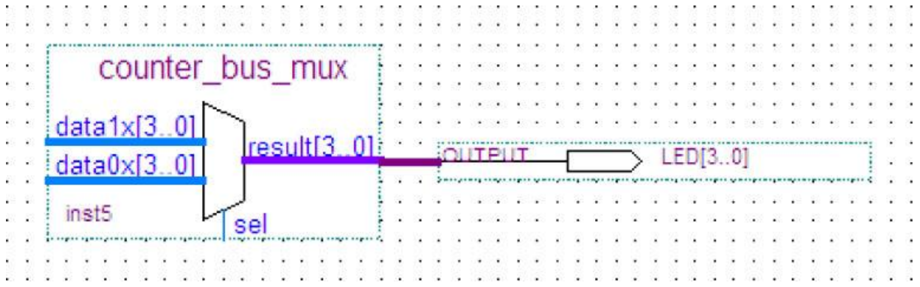


Figure 6-39 Rename the output pin

20. Attach an input pin to the multiplexer select line using an input pin:

a. Choose Edit > Insert Symbol.

b. Under Libraries, double-click quartus/libraries/ > primitives > pin > input.

c. Click OK.

21. Place this input pin below counter_bus_mux.

22. Connect the input pin to the counter_bus_mux sel pin.

23. Rename the input pin as KEY [0] (see Figure 6-40).

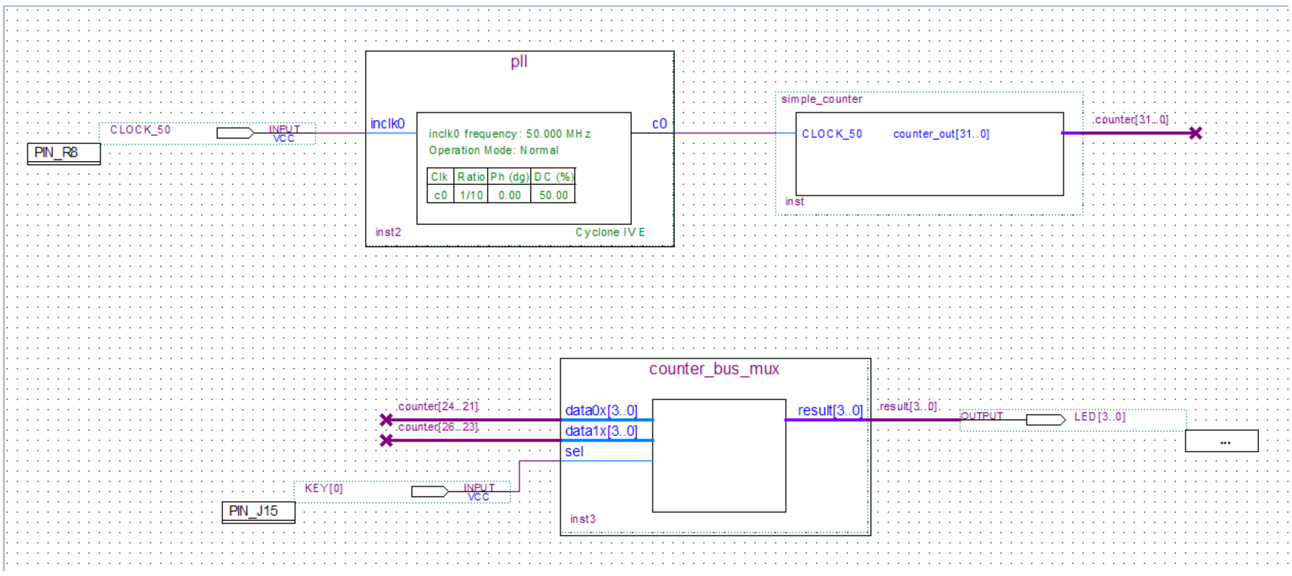


Figure 6-40 Adding the KEY [0] Input Pin

You have finished adding symbols to your design. You can add notes or information to the project as text using the Text tool on the toolbar (indicated with the A symbol). For example, you can add the label “OFF = SLOW, ON = FAST” to the KEY [0] input pin and add a project description, such as “My First FPGA Project.”

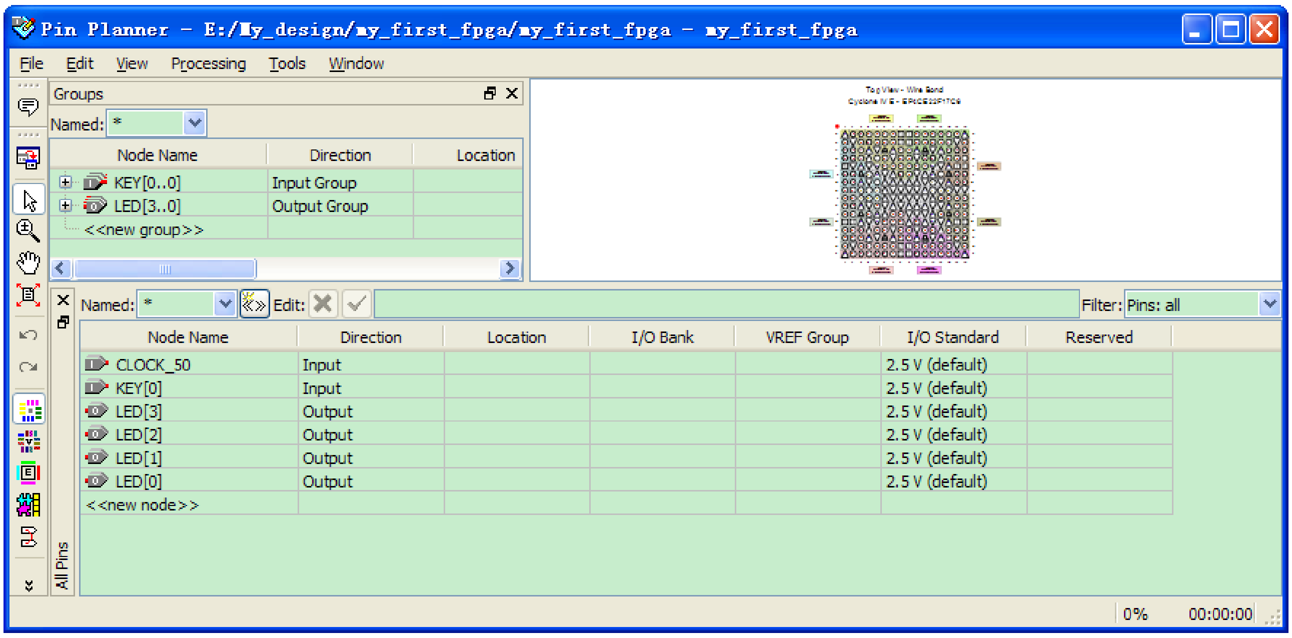
6.7 Assign the Pins

In this section, you will make pin assignments. Before making pin assignments, perform the following steps:

1. Choose Processing > Start > Start Analysis & Elaboration in preparation for assigning pin locations.
2. Click OK in the message window that appears after analysis and elaboration completes.

To make pin assignments that correlate to the KEY [0] and CLOCK_50 input pins and LED[3..0] output pin, perform the following steps:

Choose Assignments > Pins, which opens the Pin Planner, a spreadsheet-like table of specific pin assignments. The Pin Planner shows the design’s six pins. See **Figure 6-41**



Pin Planner - E:/My_design/my_first_fpga/my_first_fpga - my_first_fpga

Node Name	Direction	Location
KEY[0..0]	Input Group	
LED[3..0]	Output Group	
<<new group>>		

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
CLOCK_50	Input				2.5 V (default)	
KEY[0]	Input				2.5 V (default)	
LED[3]	Output				2.5 V (default)	
LED[2]	Output				2.5 V (default)	
LED[1]	Output				2.5 V (default)	
LED[0]	Output				2.5 V (default)	
<<new node>>						

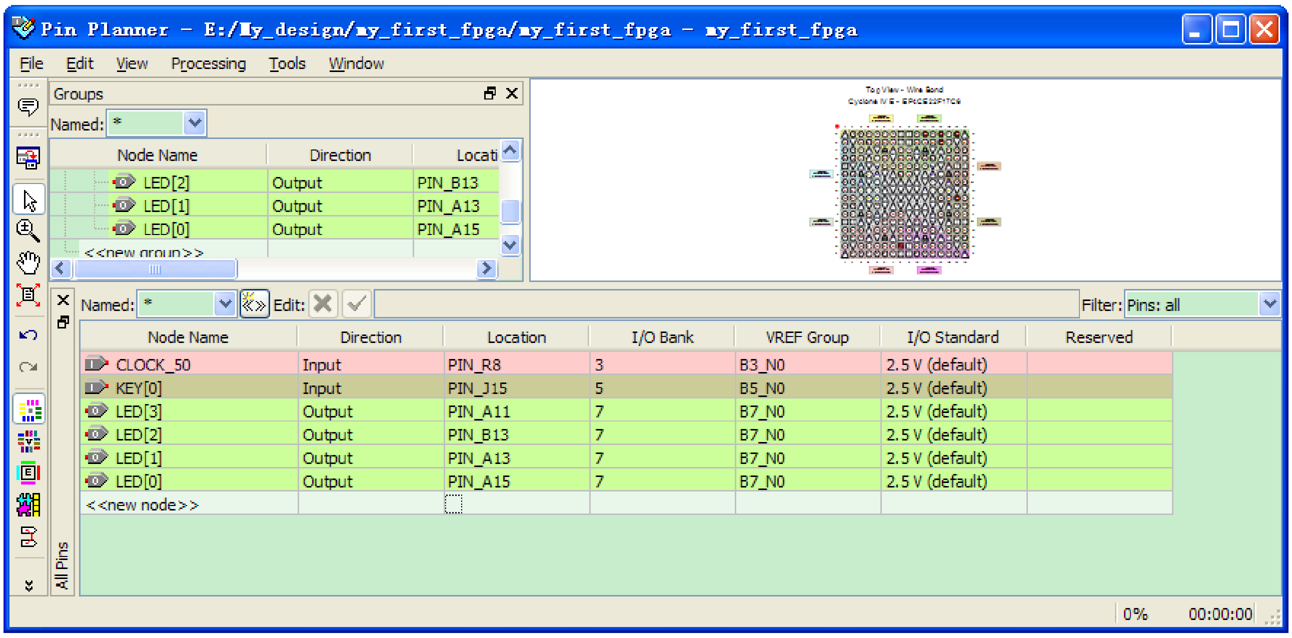
Figure 6-41 Pin Planner Example

In the Location column next to each of the six node names, add the coordinates (pin numbers) as shown in **Table 6-1** for the actual values to use with your DE0-Nano board.

Table 6-1 Pin Information Setting

Pin Name	*FPGA Pin Location*
KEY[0]	**J15**
LED[3]	**A11**
LED[2]	**B13**
LED [1]	**A13**
LED [0]	**A15**
CLOCK_50	**R8**

Double-click in the Location column for any of the six pins to open a drop-down list and type the location shown in the table alternatively, you can select the pin from a drop-down list. For example, if you type F1 and press the Enter key, the Quartus II software fills in the full PIN_F1 location name for you. The software also keeps track of corresponding FPGA data such as the I/O bank and VREF Group. Each bank has a distinct color, which corresponds to the top-view wire bond drawing in the upper right window. See **Figure 6-42**.



Pin Planner - E:/My_design/my_first_fpga/my_first_fpga - my_first_fpga

Node Name	Direction	Locati
LED[2]	Output	PIN_B13
LED[1]	Output	PIN_A13
LED[0]	Output	PIN_A15
<<new group>>		

Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
CLOCK_50	Input	PIN_R8	3	B3_N0	2.5 V (default)	
KEY[0]	Input	PIN_J15	5	B5_N0	2.5 V (default)	
LED[3]	Output	PIN_A11	7	B7_N0	2.5 V (default)	
LED[2]	Output	PIN_B13	7	B7_N0	2.5 V (default)	
LED[1]	Output	PIN_A13	7	B7_N0	2.5 V (default)	
LED[0]	Output	PIN_A15	7	B7_N0	2.5 V (default)	
<<new node>>						

Figure 6-42 Completed Pin Planning Example

Now, you are finished creating your Quartus II design!

6.8 Create a Default TimeQuest SDC File

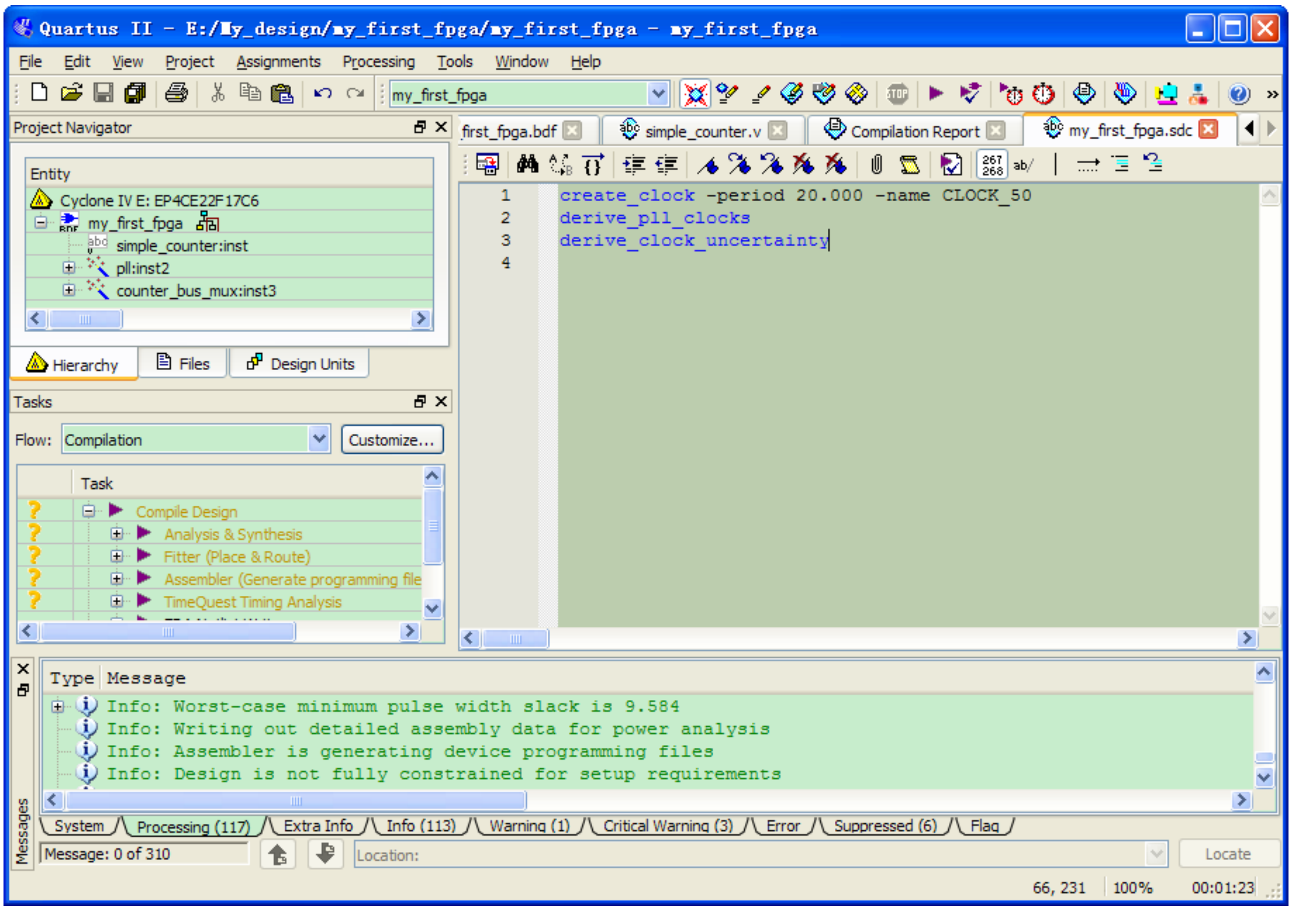
Timing settings are critically important for a successful design. For this tutorial you will create a basic Synopsys Design Constraints File (.sdc) that the Quartus II TimeQuest Timing Analyzer uses during design compilation. For more complex designs, you will need to consider the timing requirements more carefully.

To create an SDC, perform the following steps:

1. Open the TimeQuest Timing Analyzer by choosing Tools > TimeQuest Timing Analyzer.
2. Choose File > New SDC file. The SDC editor opens.
3. Type the following code into the editor:

```
create_clock -period 20.000 -name CLOCK_50

derive_pll_clocks

derive_clock_uncertainty
```

4. Save this file as my_first_fpga.sdc (see Figure 6-43)


Quartus II - E:/My_design/my_first_fpga/my_first_fpga - my_first_fpga

```
create_clock -period 20.000 -name CLOCK_50
derive_pll_clocks
derive_clock_uncertainty
```

```
Type Message
Info: Worst-case minimum pulse width slack is 9.584
Info: Writing out detailed assembly data for power analysis
Info: Assembler is generating device programming files
Info: Design is not fully constrained for setup requirements
```

Figure 6-43 Default SDC

Naming the SDC with the same name as the top-level file except for the .sdc extension causes the Quartus II software to using this timing analysis file automatically by default. If you used another name, you would need to add the SDC to the assignments file list.

6.9 Compile Your Design

After creating your design you must compile it. Compilation converts the design into a bitstream that can be downloaded into the FPGA. The most important output of compilation is an SRAM Object File (.sof), which you use to program the device. The software also generates other report files that provide information about your code as it compiles.

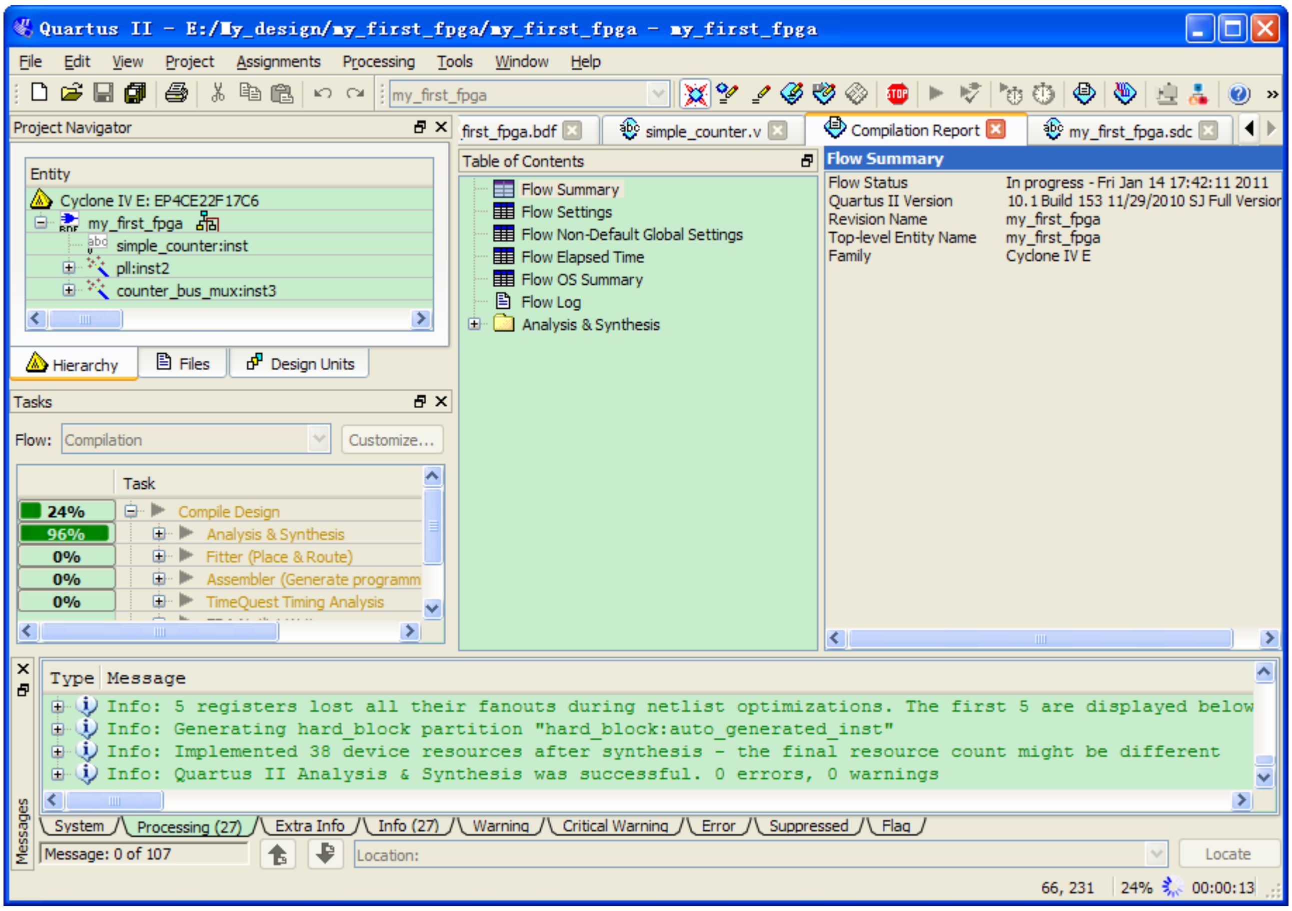
If you want to store .SOF in memory device (such as flash or EEPROMs), you must first convert the SOF to a file type specifically for the targeted memory device.

Now that you have created a complete Quartus II project and entered all assignments, you can compile the design.

In the Processing menu, choose Start Compilation or click the Play button on the toolbar.

If you are asked to save changes to your BDF, click Yes.

While compiling your design, the Quartus II software provides useful information about the compilation (see **Figure 6-44**).



Task	Progress
Compile Design	24%
Analysis & Synthesis	96%
Fitter (Place & Route)	0%
Assembler (Generate programm	0%
TimeQuest Timing Analysis	0%

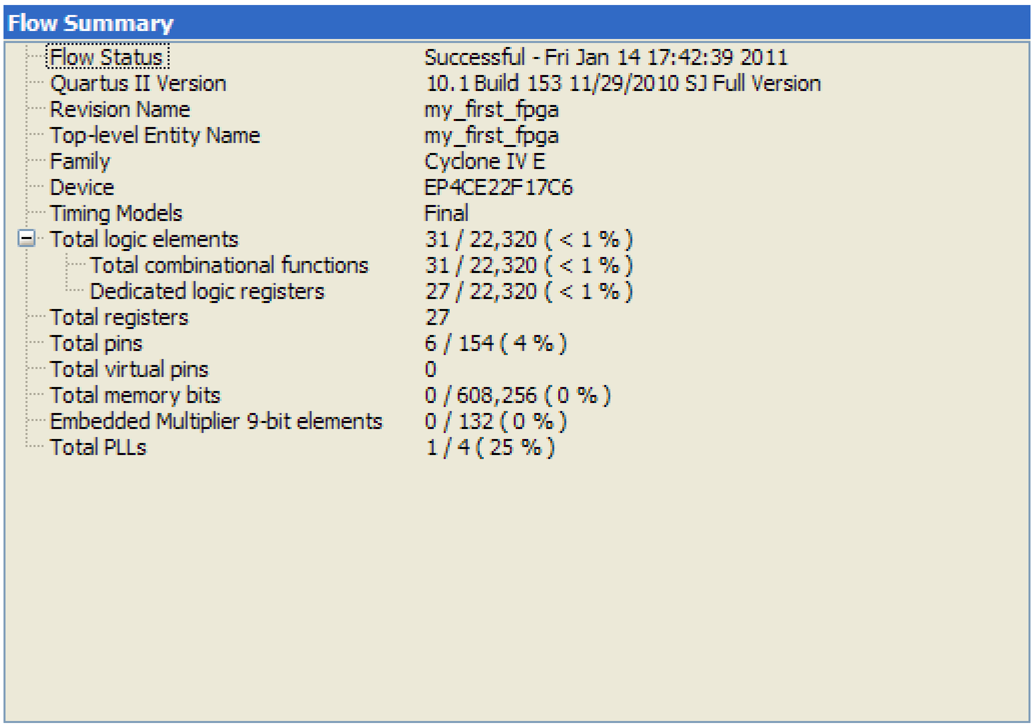
Flow Summary	
Flow Status	In progress - Fri Jan 14 17:42:11 2011
Quartus II Version	10.1 Build 153 11/29/2010 SJ Full Version
Revision Name	my_first_fpga
Top-level Entity Name	my_first_fpga
Family	Cyclone IV E

```
Type Message
Info: 5 registers lost all their fanouts during netlist optimizations. The first 5 are displayed below
Info: Generating hard_block partition "hard_block:auto_generated_inst"
Info: Implemented 38 device resources after synthesis - the final resource count might be different
Info: Quartus II Analysis & Synthesis was successful. 0 errors, 0 warnings
```

Figure 6-44 Compilation Message for project

When compilation is complete, the Quartus II software displays a message. Click OK to close the message box.

The Quartus II Messages window displays many messages during compilation. It should not display any critical warnings; it may display a few warnings that indicate that the device timing information is preliminary or that some parameters on the I/O pins used for the LEDs were not set. The software provides the compilation results in the Compilation Report tab as shown in **Figure 6-45**.


Flow Summary

Flow Status	Successful - Fri Jan 14 17:42:39 2011
Quartus II Version	10.1 Build 153 11/29/2010 SJ Full Version
Revision Name	my_first_fpga
Top-level Entity Name	my_first_fpga
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	31 / 22,320 (< 1 %)
Total combinational functions	31 / 22,320 (< 1 %)
Dedicated logic registers	27 / 22,320 (< 1 %)
Total registers	27
Total pins	6 / 154 (4 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 6-45 Compilation Report Example

6.10 Program the FPGA Device

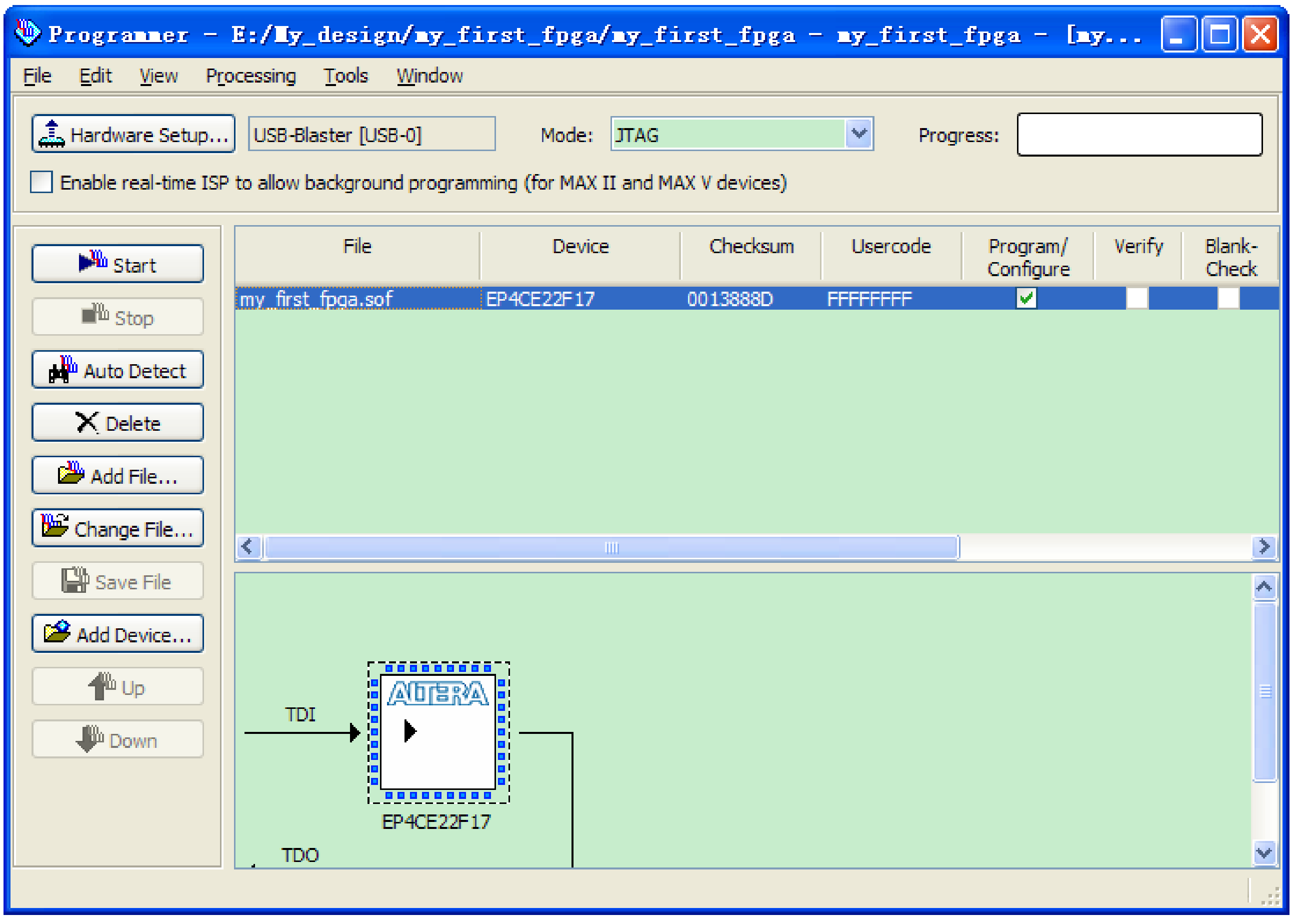
After compiling and verifying your design you are ready to program the FPGA on the development board. You download the SOF you just created into the FPGA using the USB-Blaster circuitry on the board. Set up your hardware for programming using the following steps:

For the DE0-Nano board, connect the USB-Blaster (included in your development kit) to J3 and the USB cable to the USB-Blaster. Connect the other end of the USB cable to the host computer.

Refer to the getting started user guide for detailed instructions on how to connect the cables.

Program the FPGA using the following steps.

1. Choose Tools > Programmer. The Programmer window opens. See Figure 6-46.


Programmer - E:/My_design/my_first_fpga/my_first_fpga - my_first_fpga - [my...

Hardware: USB-Blaster [USB-0] Mode: JTAG Progress:

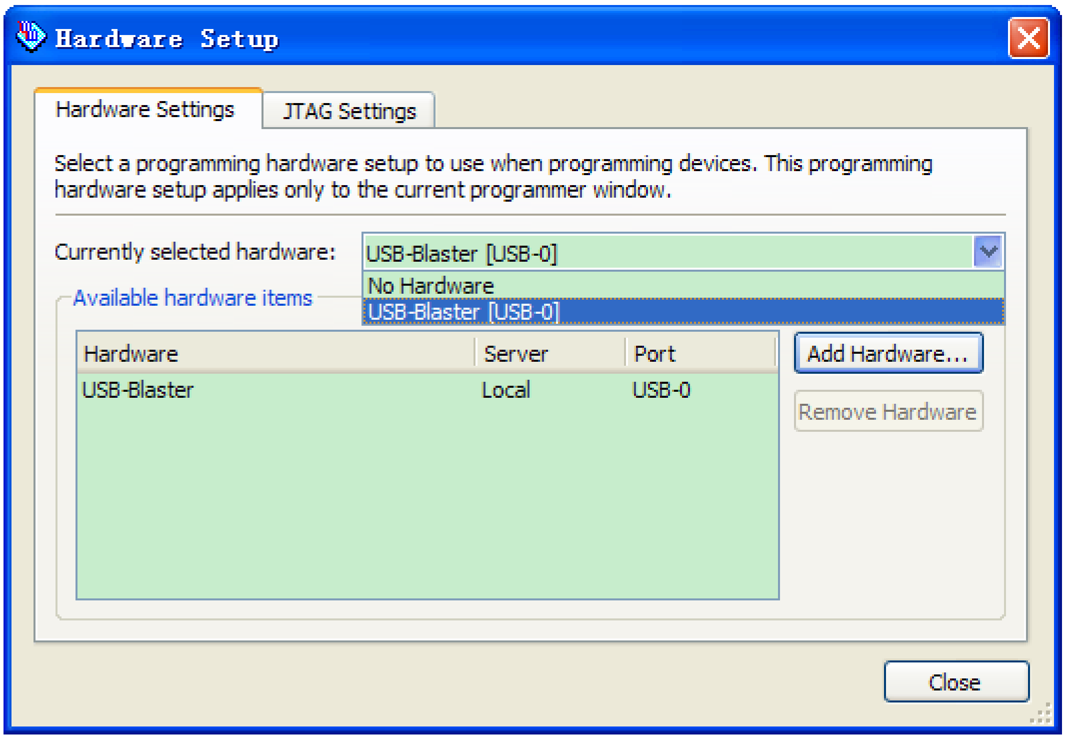
☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check
my_first_fpga.sof	EP4CE22F17	0013888D	FFFFFFFF	✓	☐	☐

EP4CE22F17

Figure 6-46 Programmer Window

2. Click Hardware Setup.
3. If it is not already turned on, turn on the USB-Blaster [USB-0] option under currently selected hardware. See **Figure 6-47**.


Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

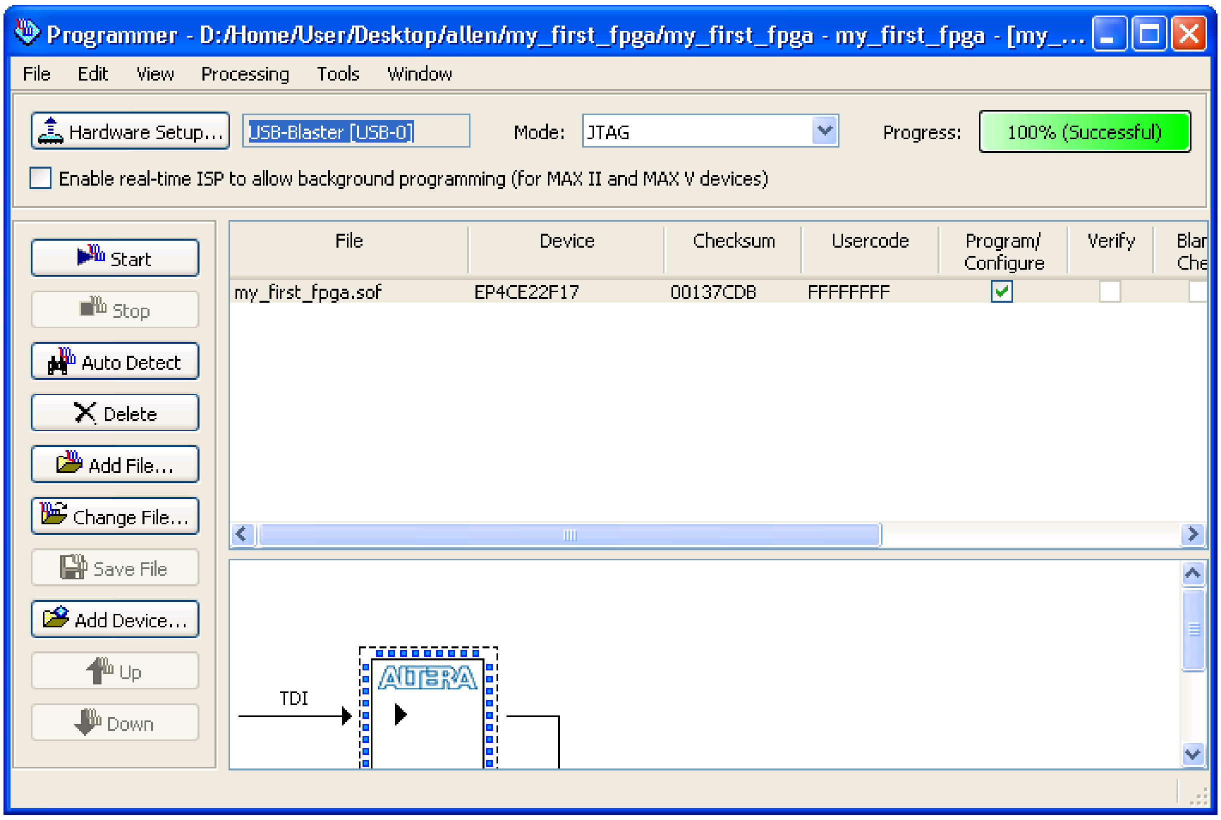
No Hardware
USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

Figure 6-47 Hardware Setting

4. Click Close.
5. If the file name in the Programmer does not show my_first_fpga.sof, click Add File.
6. Select the my_first_fpga.sof file from the project directory (see **Figure 6-48**).


Programmer - D:/Home/User/Desktop/allen/my_first_fpga/my_first_fpga - my_first_fpga - [my_...

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blan Che
my_first_fpga.sof	EP4CE22F17	00137CDB	FFFFFFFF	✓		

Figure 6-48 Downloading Complete

Congratulations, you have created, compiled, and programmed your first FPGA design! The compiled SRAM Object File (.sof) is loaded onto the FPGA on the development board and the design should be running.

6.11 Verify The Hardware

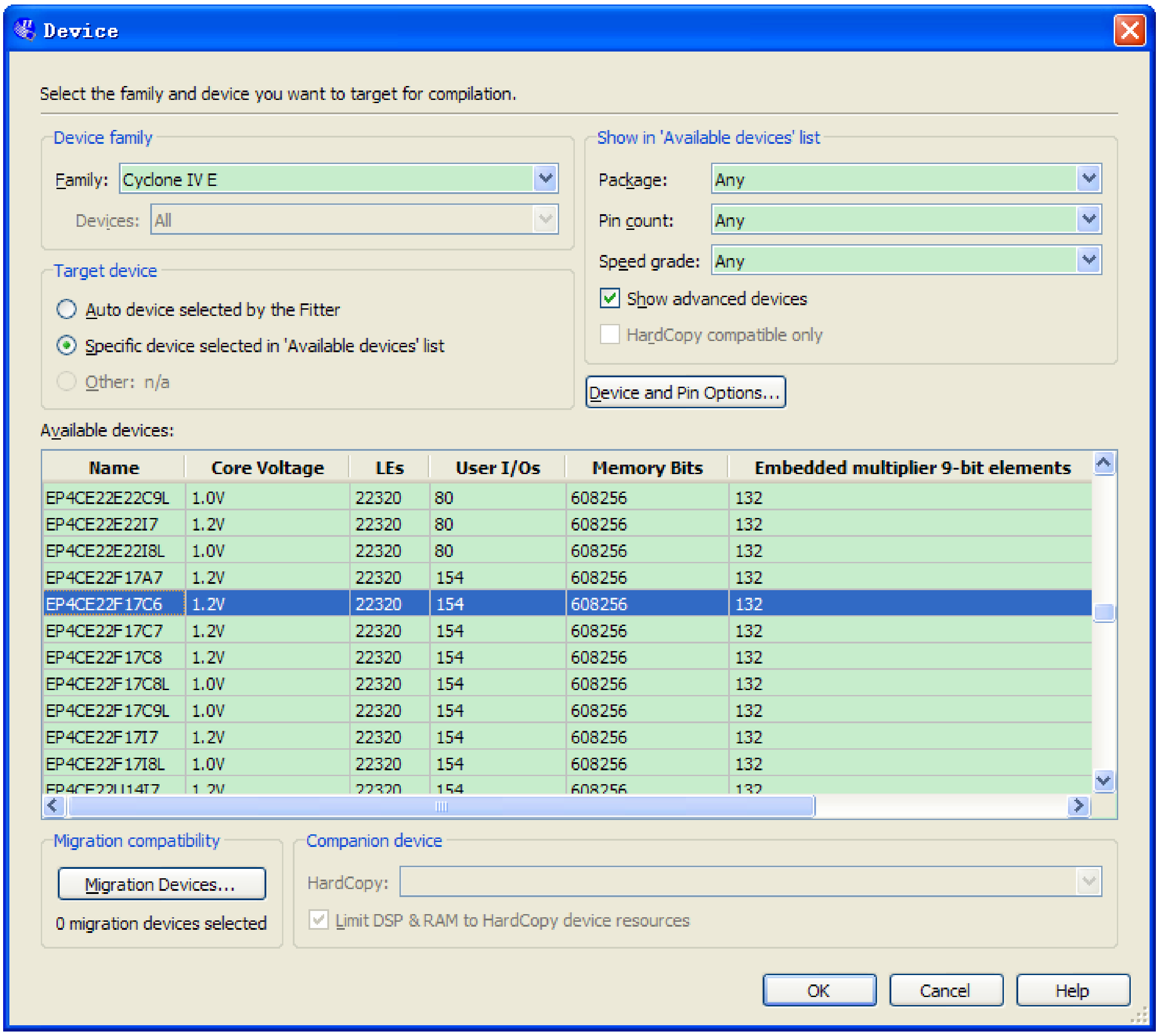
When you verify the design in hardware, you observe the runtime behavior of the FPGA hardware design and ensure that it is functioning appropriately.

Verify the design by performing the following steps:

1. Observe that the four development board LEDs appear to be advancing slowly in a binary count pattern, which is driven by the simple_counter bits [26..23].

The LEDs are active low, therefore, when counting begins all LEDs are turned on (the 0000 state).

2. Press and hold KEY [0] on the development board and observe that the LEDs advance more quickly. Pressing this KEY causes the design to multiplex using the faster advancing part of the counter (bits [24..21]).
3. If other LEDs emit faintness light, Choose Assignments > Device. Click Device and Options. See Figure 6-49.



Device

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

Device family

Family: Cyclone IV E

Devices: All

Target device

( ) Auto device selected by the Fitter

(●) Specific device selected in 'Available devices' list

( ) Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

[✓] Show advanced devices

[ ] HardCopy compatible only

Device and Pin Options...

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements
EP4CE22E22C9L	1.0V	22320	80	608256	132
EP4CE22E22I7	1.2V	22320	80	608256	132
EP4CE22E22I8L	1.0V	22320	80	608256	132
EP4CE22F17A7	1.2V	22320	154	608256	132
EP4CE22F17C6	1.2V	22320	154	608256	132
EP4CE22F17C7	1.2V	22320	154	608256	132
EP4CE22F17C8	1.2V	22320	154	608256	132
EP4CE22F17C8L	1.0V	22320	154	608256	132
EP4CE22F17C9L	1.0V	22320	154	608256	132
EP4CE22F17I7	1.2V	22320	154	608256	132
EP4CE22F17I8L	1.0V	22320	154	608256	132
EP4CE22U14I7	1.2V	22320	154	608256	132

Migration compatibility

Migration Devices...

0 migration devices selected

Companion device

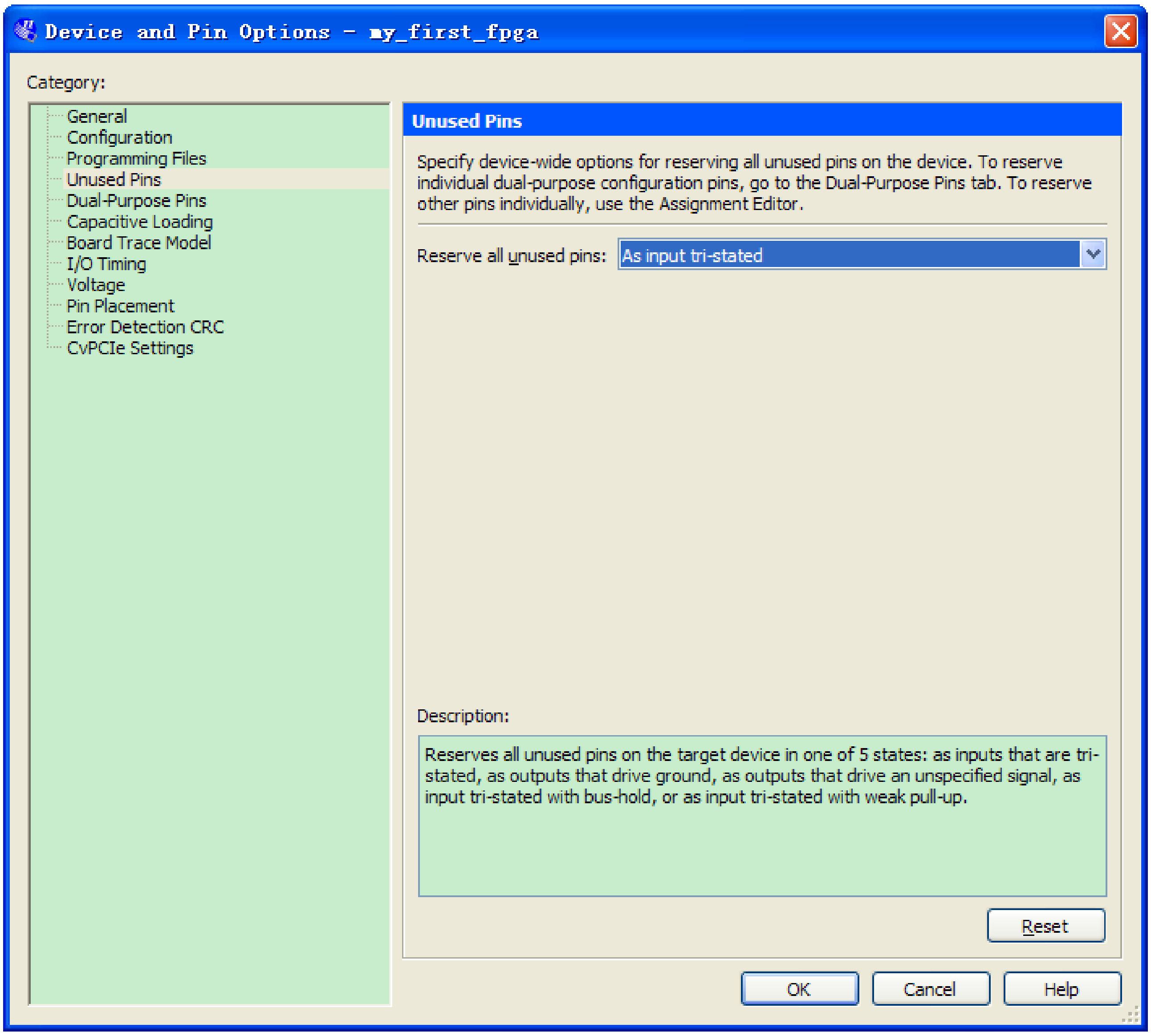
HardCopy:

[✓] Limit DSP & RAM to HardCopy device resources

OK Cancel Help

Figure 6-49 Device and Options

Choose unused pins. Reserve all unused pins: Choose the As input tri-stated option. See **Figure 6-50**.


Device and Pin Options - my_first_fpga

Category:

General
Configuration
Programming Files
Unused Pins
Dual-Purpose Pins
Capacitive Loading
Board Trace Model
I/O Timing
Voltage
Pin Placement
Error Detection CRC
CvPCIe Settings

Unused Pins

Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated

Description:

Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

Figure 6-50 Setting unused pins

Click twice OK.

4. In the Processing menu, choose Start Compilation. After the compile, Choose Tools > Programmer. Select the my_first_fpga.sof file from the project directory. Click Start. At this time you could find the other LEDs are unlighted.

Chapter 7

My First Nios II Project Tutorial

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.

7.1 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 DE0-Nano 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

7.2 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 7-1** and **Figure 7-2**.

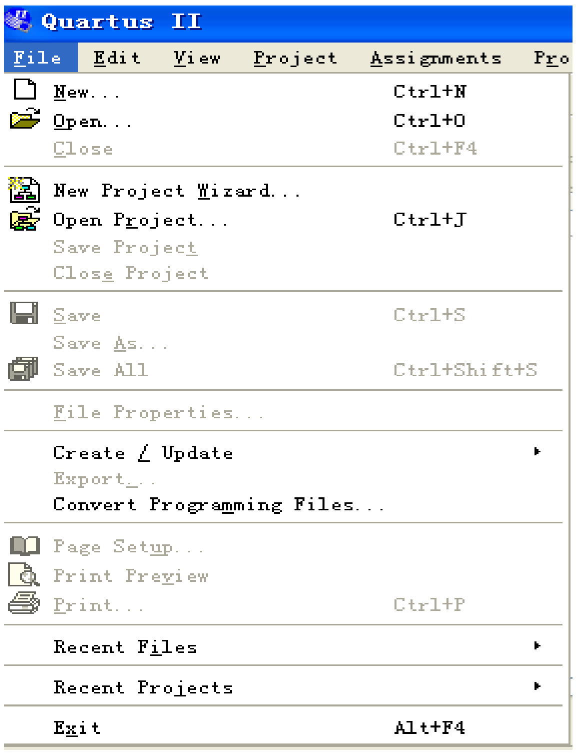
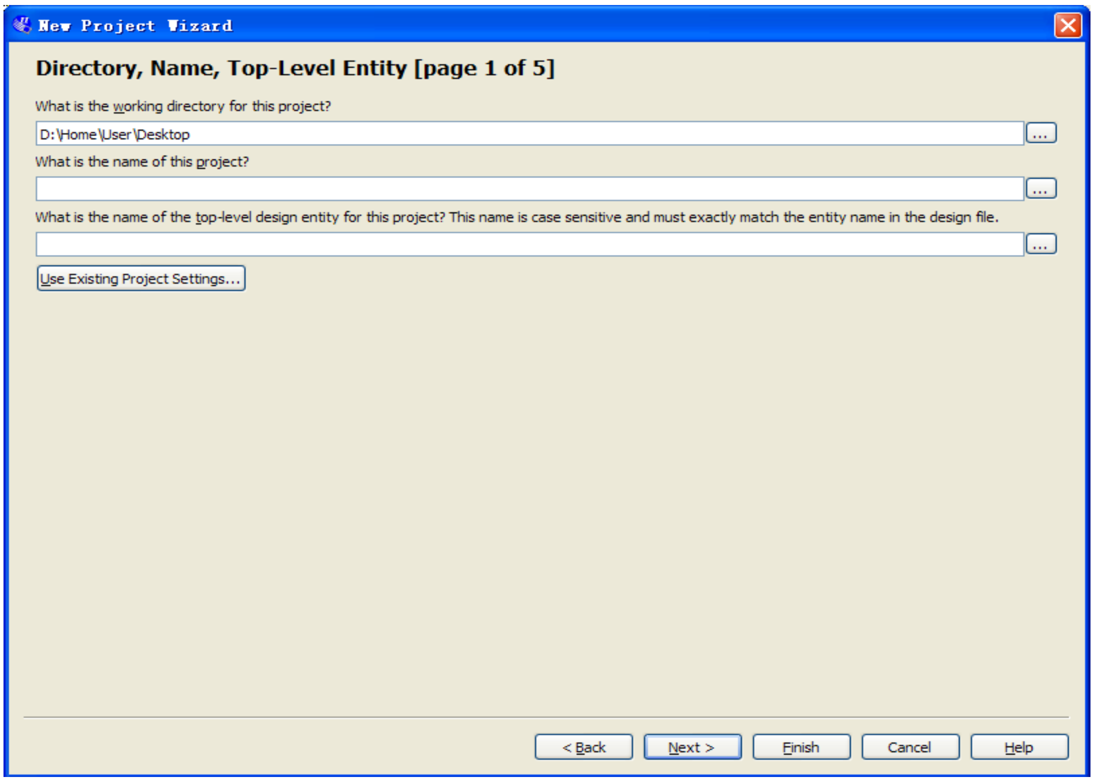


Figure 7-1 Start to Create a New Project


New Project Wizard

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

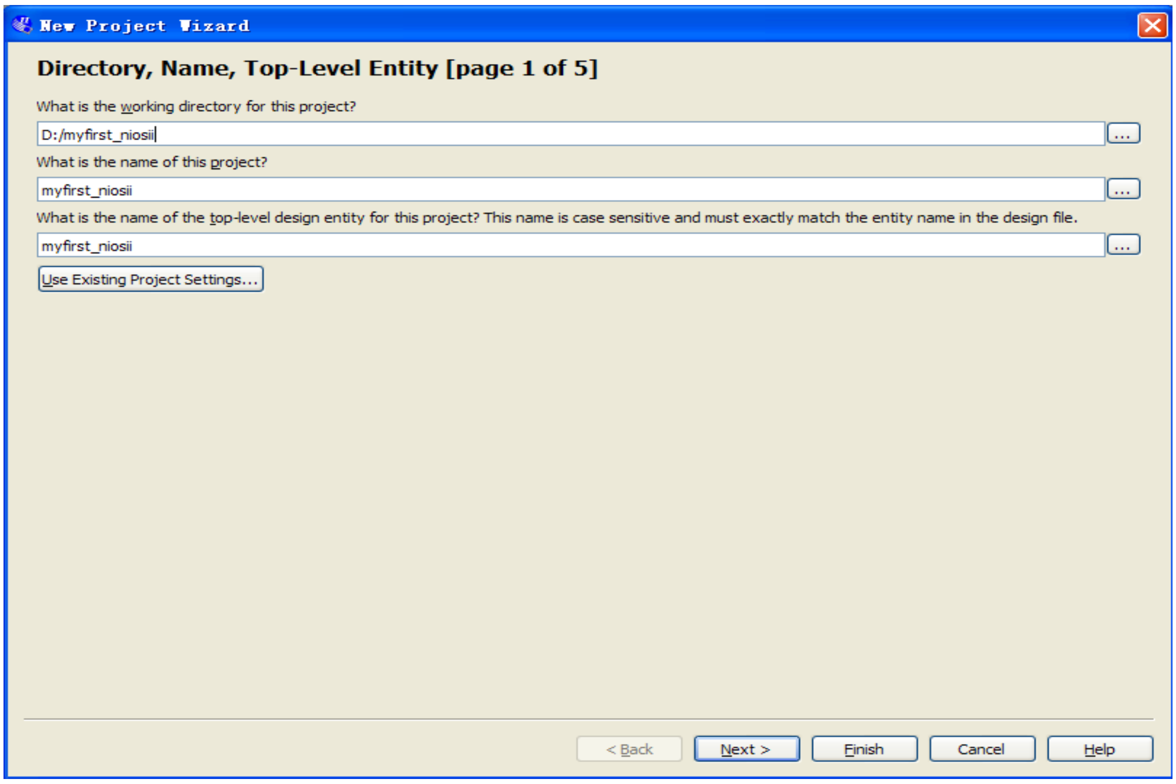
D:\Home\User\Desktop

What is the name of this project?

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

Figure 7-2 New Project Wizard

2. Choose a working directory for this project, type project name and top-level entity name as shown in Figure 7-3. Then click **Next**, you will see a window as shown in Figure 7-4.


New Project Wizard

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

D:/myfirst_niosii

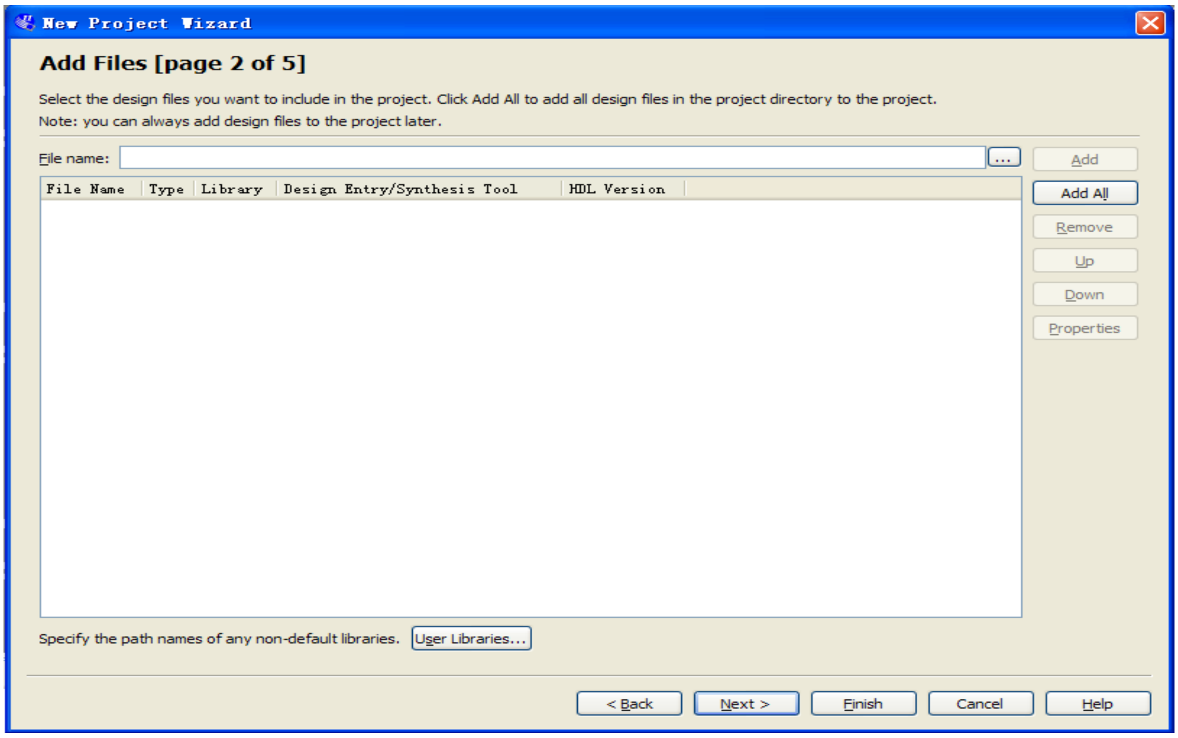
What is the name of this project?

myfirst_niosii

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

myfirst_niosii

Figure 7-3 Input the working directory, the name of project, top-level design entity


New Project Wizard

Add Files [page 2 of 5]

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

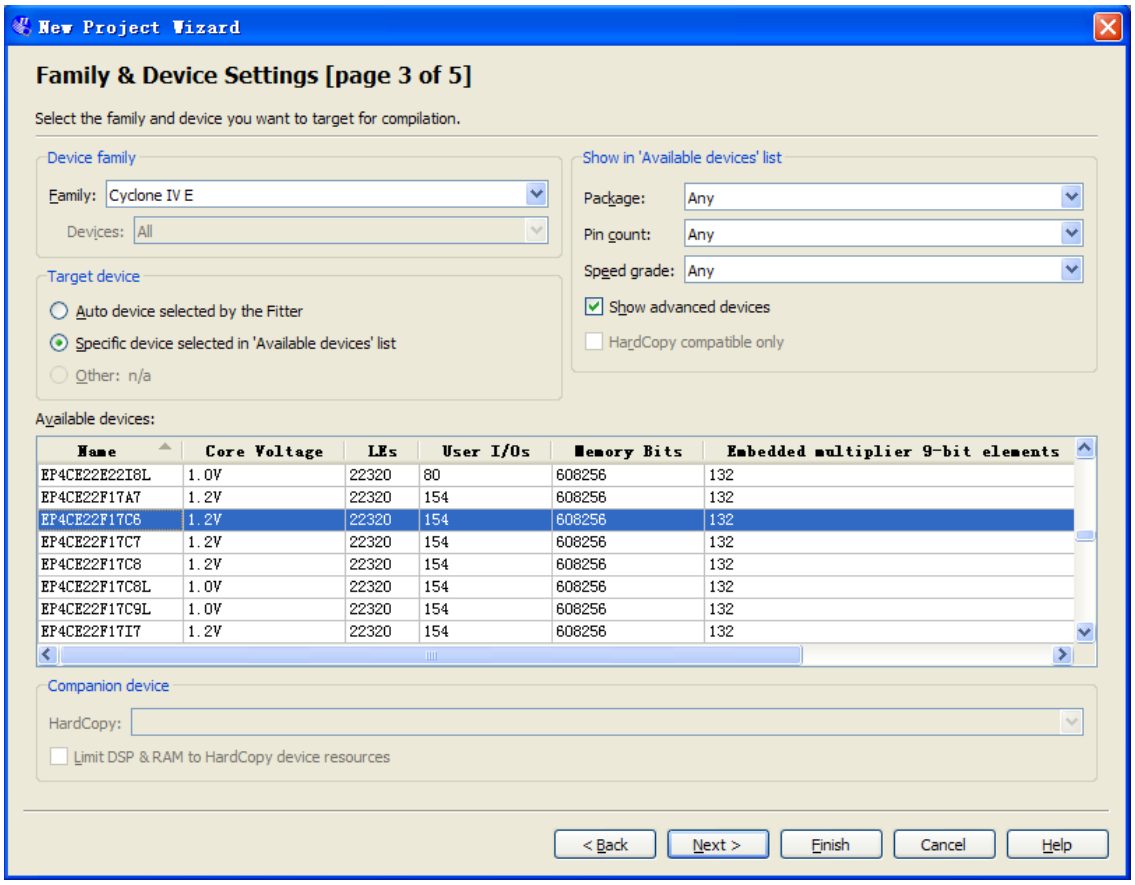
File name:

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version

Specify the path names of any non-default libraries.

Figure 7-4 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 7-5**. Then click **Next** to next window as shown in **Figure 7-6**.


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 IV E

Devices: All

Target device

◯ Auto device selected by the Fitter

[✓ Specific device selected in 'Available devices' list]

◯ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

[✓ Show advanced devices]

☐ HardCopy compatible only

Available devices:

Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements
EP4CE22E22I8L	1.0V	22320	80	608256	132
EP4CE22F17A7	1.2V	22320	154	608256	132
EP4CE22F17C6	1.2V	22320	154	608256	132
EP4CE22F17C7	1.2V	22320	154	608256	132
EP4CE22F17C8	1.2V	22320	154	608256	132
EP4CE22F17C8L	1.0V	22320	154	608256	132
EP4CE22F17C9L	1.0V	22320	154	608256	132
EP4CE22F17I7	1.2V	22320	154	608256	132

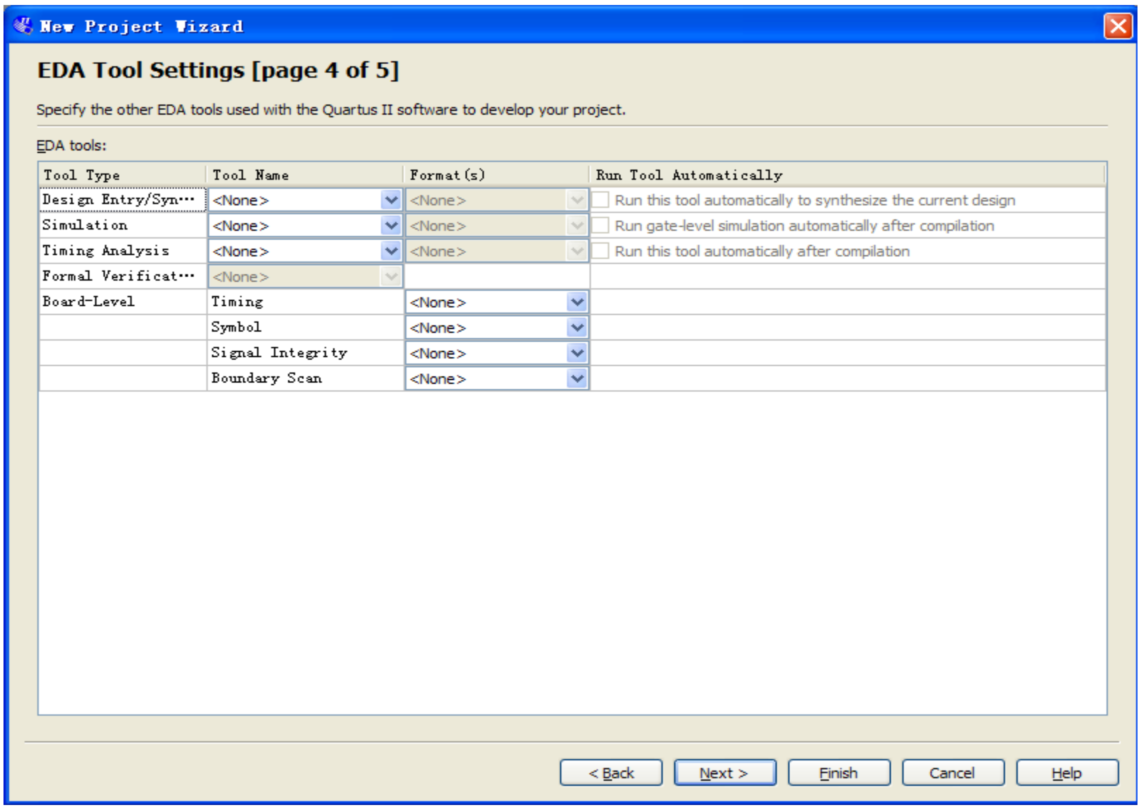
Companion device

HardCopy:

☐ Limit DSP & RAM to HardCopy device resources

Figure 7-5 New Project Wizard: Family & Device Settings [page 3 of 5]

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



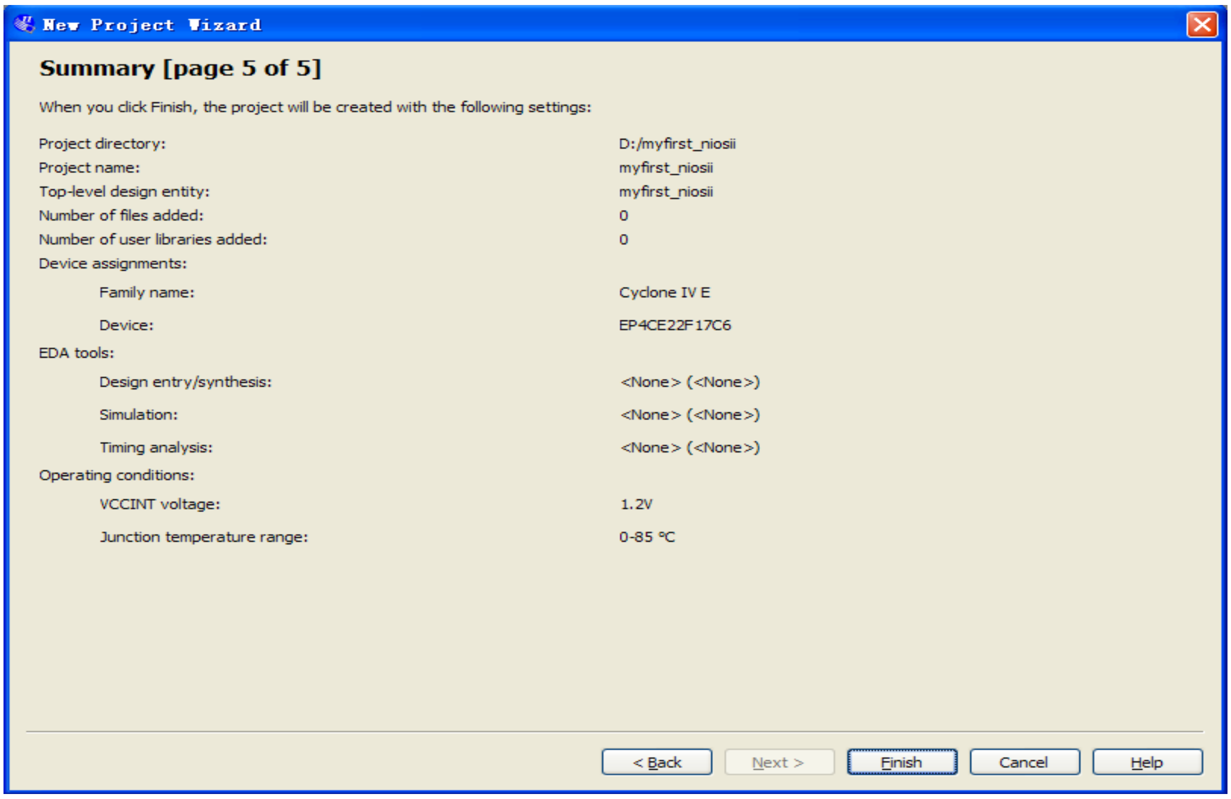
EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Syn…	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	<None>	<None>	Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	Run this tool automatically after compilation
Formal Verificat…	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figure 7-6 New Project Wizard: EDA Tool Settings [page 4 of 5]



Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

Project directory:	D:/myfirst_niosii
Project name:	myfirst_niosii
Top-level design entity:	myfirst_niosii
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone IV E
Device:	EP4CE22F17C6
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	<None> (<None>)
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 ℃

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

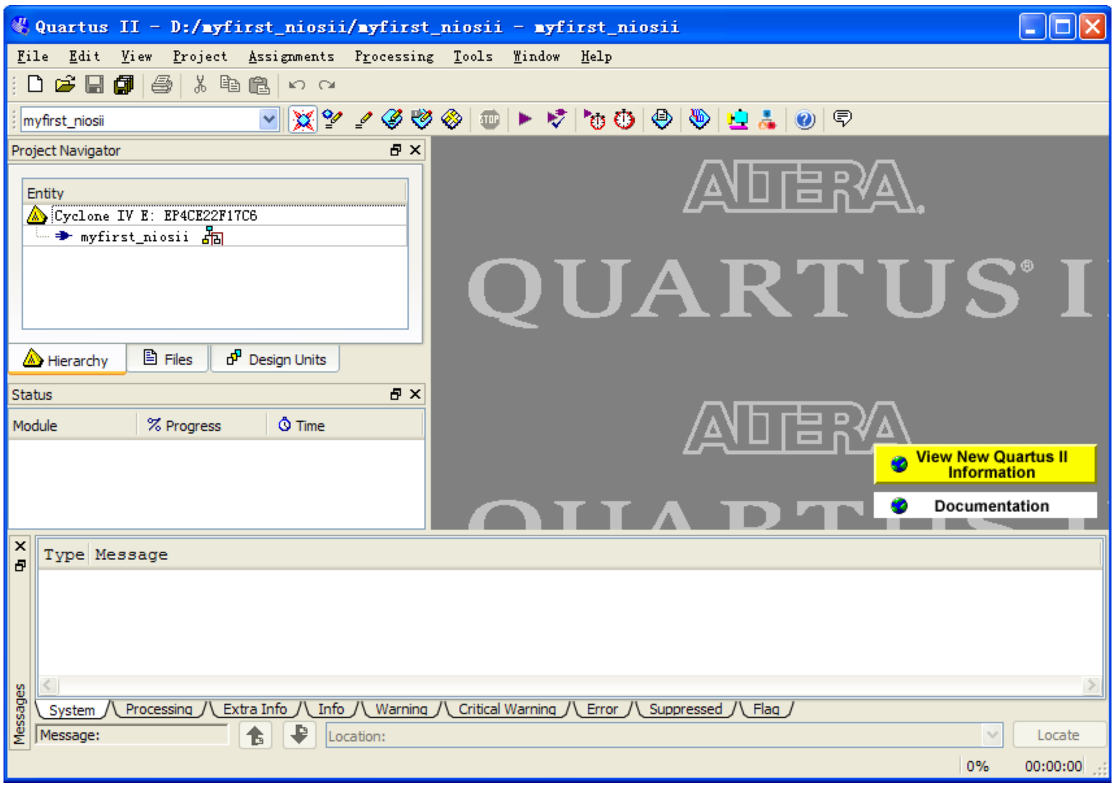

Quartus II - D:/myfirst_niosii/myfirst_niosii - myfirst_niosii
File Edit View Project Assignments Processing Tools Window Help
myfirst_niosii
Project Navigator
Entity
Cyclone IV E: EP4CE22F17C6
myfirst_niosii
Hierarchy
Files
Design Units
Status
Module
Progress
Time
View New Quartus II Information
Documentation
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Message:
Location:
Locate
0%
00:00:00

Figure 7-8 A New Complete Project

5. Choose **Tools** > **SOPC Builder** to open new SOPC system wizard .See Figure 7-9 and Figure 7-10.

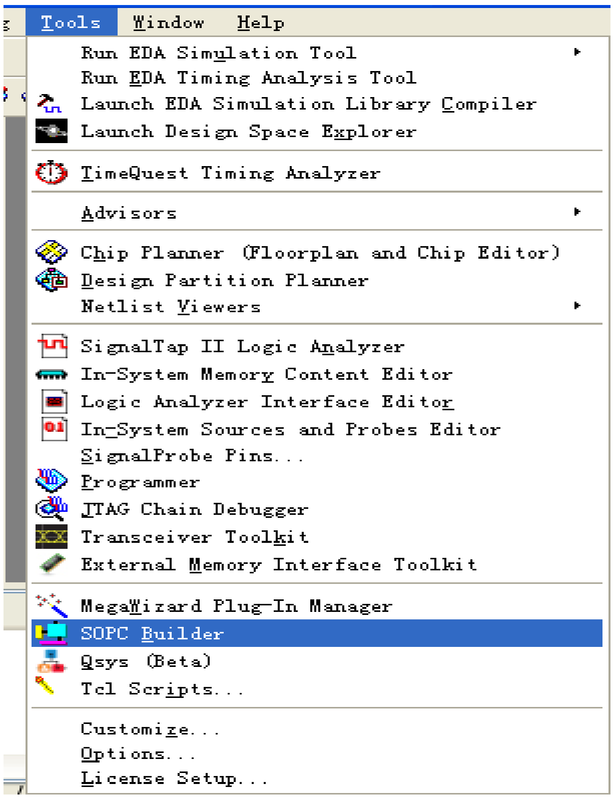
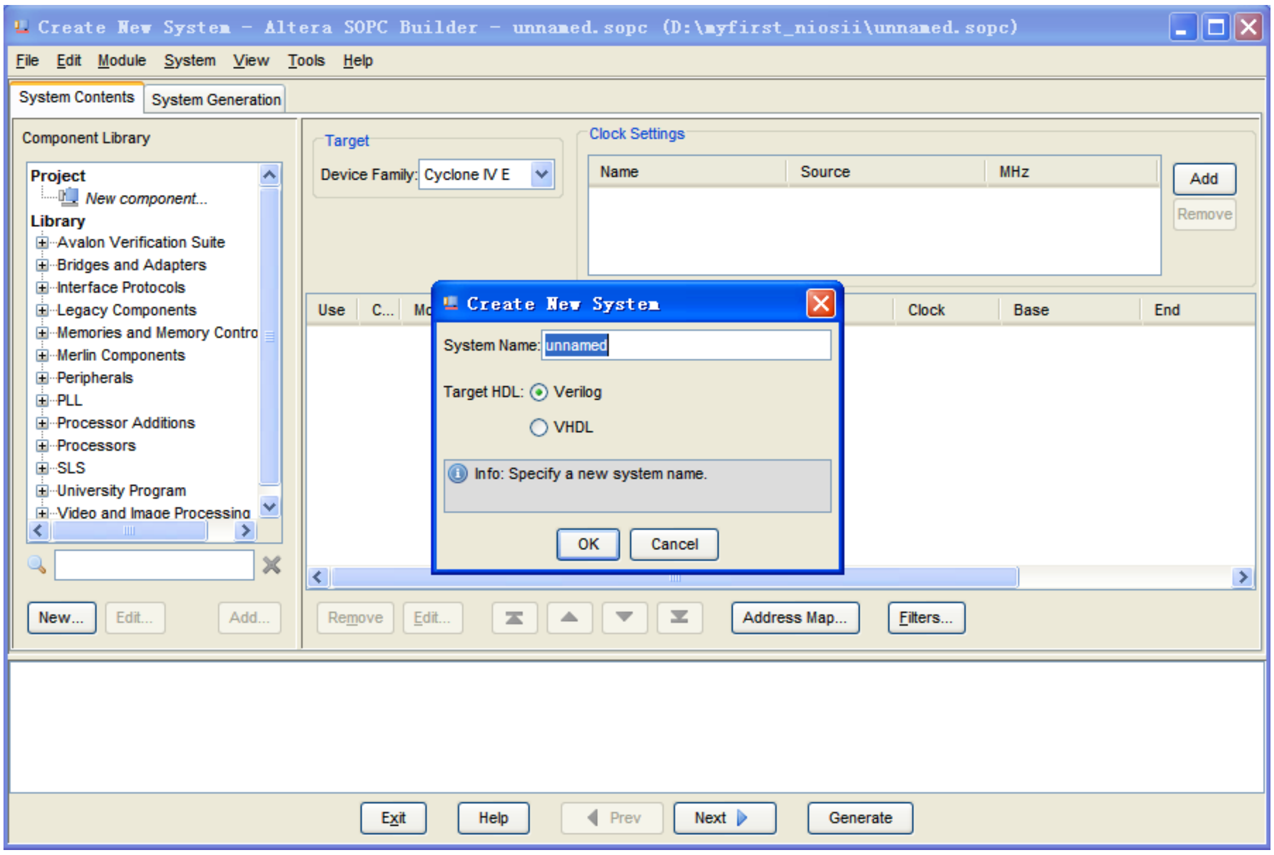

Tools
Window
Help
Run EDA Simulation Tool
Run EDA Timing Analysis Tool
Launch EDA Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner (Floorplan and Chip Editor)
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Transceiver Toolkit
External Memory Interface Toolkit
MegaWizard Plug-In Manager
SOPC Builder
Qsys (Beta)
Tcl Scripts...
Customize...
Options...
License Setup...

Figure 7-9 SOPC Builder Menu



Device Family: Cyclone IV E

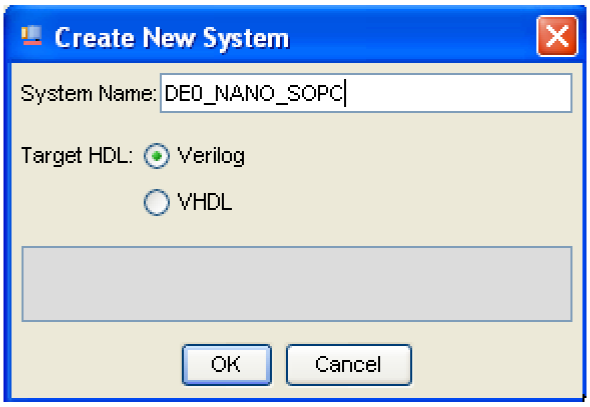
System Name: unnamed

Target HDL: [✓ Verilog] VHDL

Info: Specify a new system name.

Figure 7-10 Create New SOPC System [0]

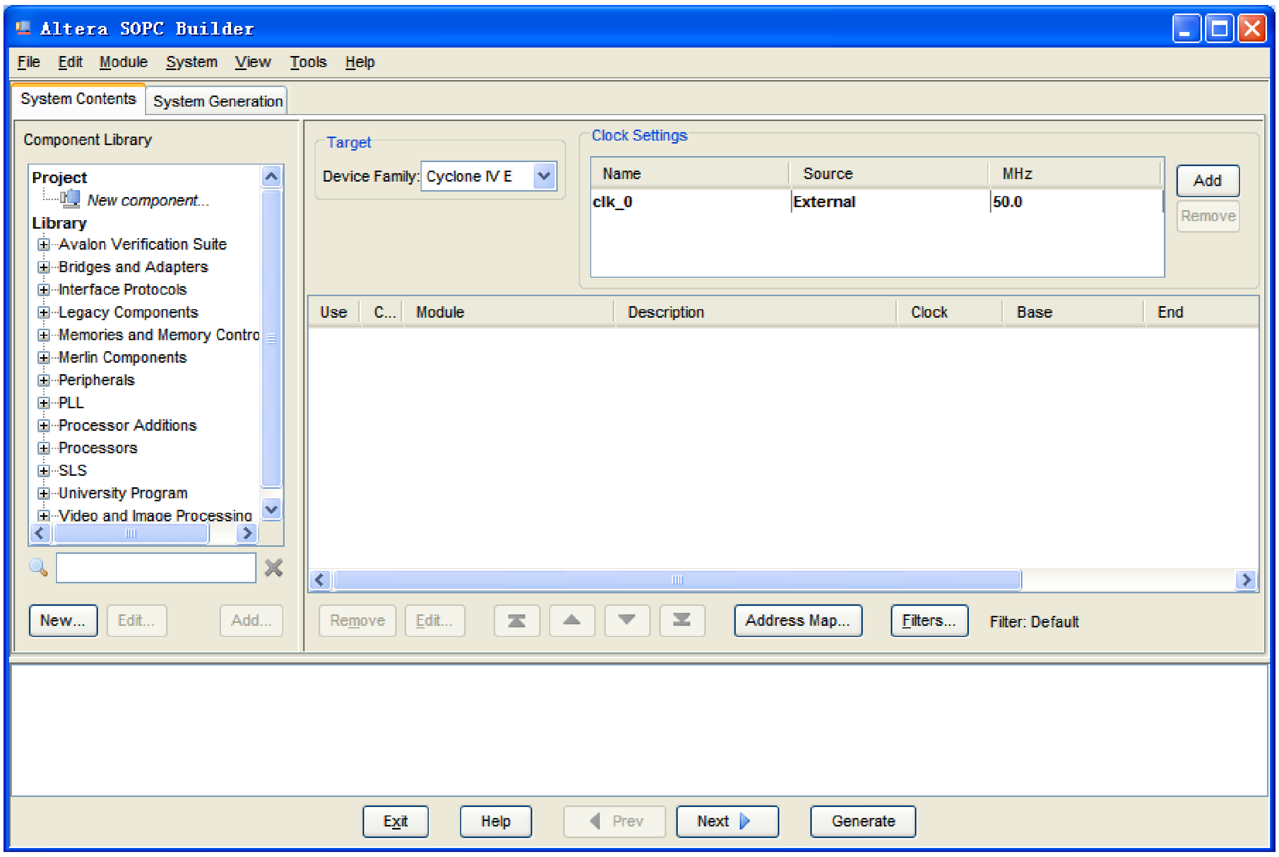
6. Rename **System Name** as shown in **Figure 7-11**. Click **OK** and your will see a window as shown in **Figure 7-12**.



System Name: DE0_NANO_SOPC

Target HDL: [✓ Verilog] VHDL

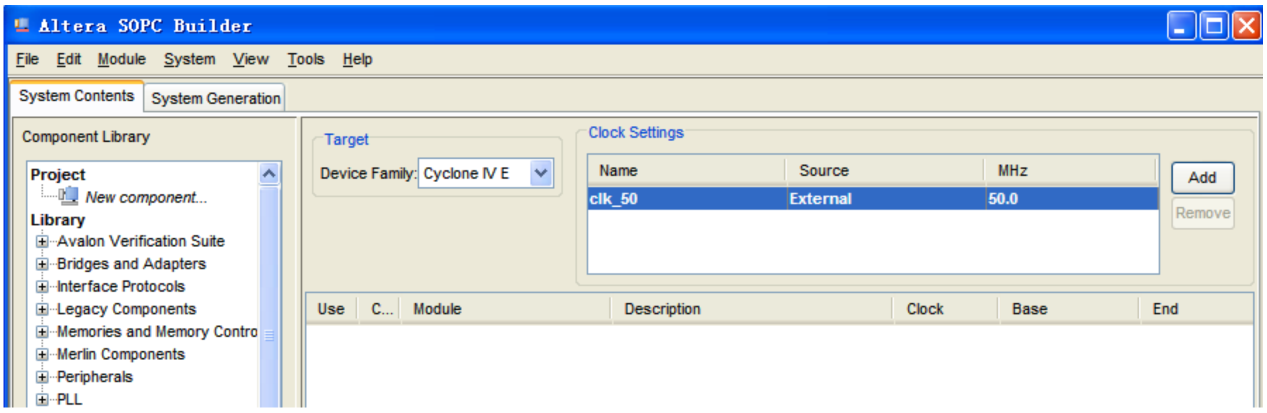
Figure 7-11 Create New System [1]


Altera SOPC Builder
File Edit Module System View Tools Help
System Contents System Generation
Component Library
Project
New component...
Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Contro
Merlin Components
Peripherals
PLL
Processor Additions
Processors
SLS
University Program
Video and Image Processing
New... Edit... Add...
Target
Device Family: Cyclone IV E
Clock Settings
Add
Remove
Use C... Module Description Clock Base End
Remove Edit... Address Map... Filters... Filter: Default
Exit Help Prev Next Generate

Name	Source	MHz
clk_0	External	50.0

Figure 7-12 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 7-13.


Altera SOPC Builder
File Edit Module System View Tools Help
System Contents System Generation
Component Library
Project
New component...
Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Contro
Merlin Components
Peripherals
PLL
Target
Device Family: Cyclone IV E
Clock Settings
Add
Remove
Use C... Module Description Clock Base End

Name	Source	MHz
clk_50	External	50.0

Figure 7-13 Rename Clock Name

8. Choose **Library** > **Processors** > **Nios II Processor** to open wizard of adding cpu component. See Figure 7-14 and Figure 7-15.

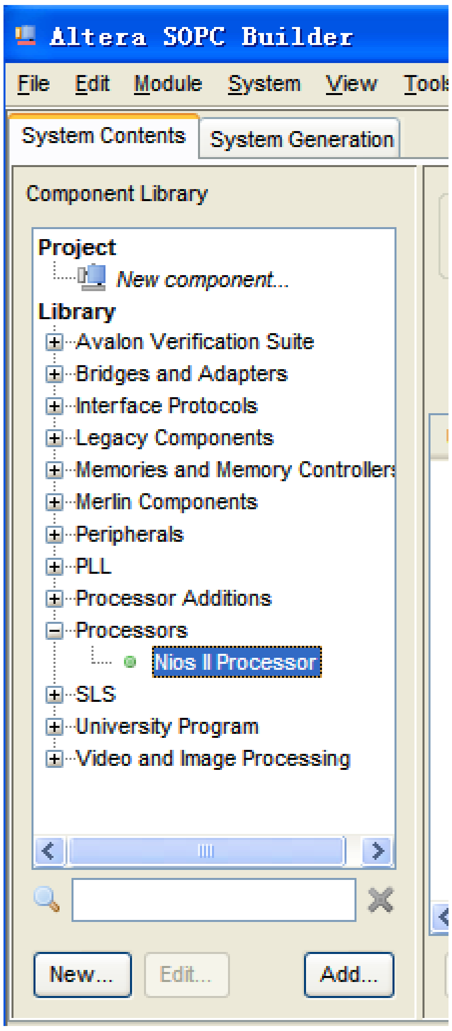
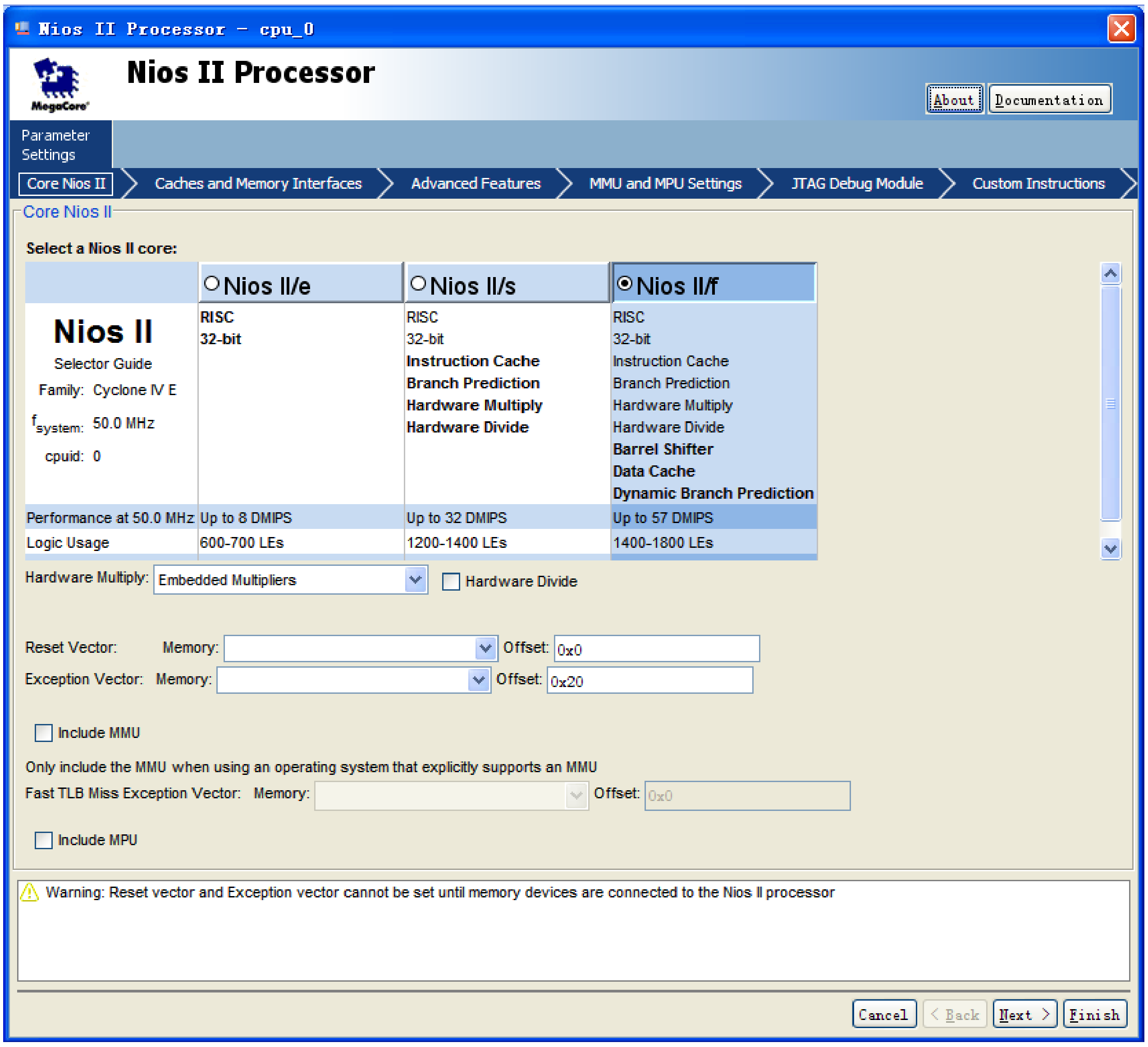


Figure 7-14 Add NIOS II Processor


Nios II Processor - cpu_0

Nios II Processor

Parameter Settings: Core Nios II > Caches and Memory Interfaces > Advanced Features > MMU and MPU Settings > JTAG Debug Module > Custom Instructions

Core Nios II

Select a Nios II core:

Nios II Selector Guide Family: Cyclone IV E f_{system}: 50.0 MHz cpuid: 0	○ Nios II/e	○ Nios II/s	[✓ ● Nios II/f]
	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Performance at 50.0 MHz	Up to 8 DMIPS	Up to 32 DMIPS	Up to 57 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs

Hardware Multiply: Embedded Multipliers ☐ Hardware Divide

Reset Vector: Memory: Offset: 0x0

Exception Vector: Memory: Offset: 0x20

☐ Include MMU

Only include the MMU when using an operating system that explicitly supports an MMU

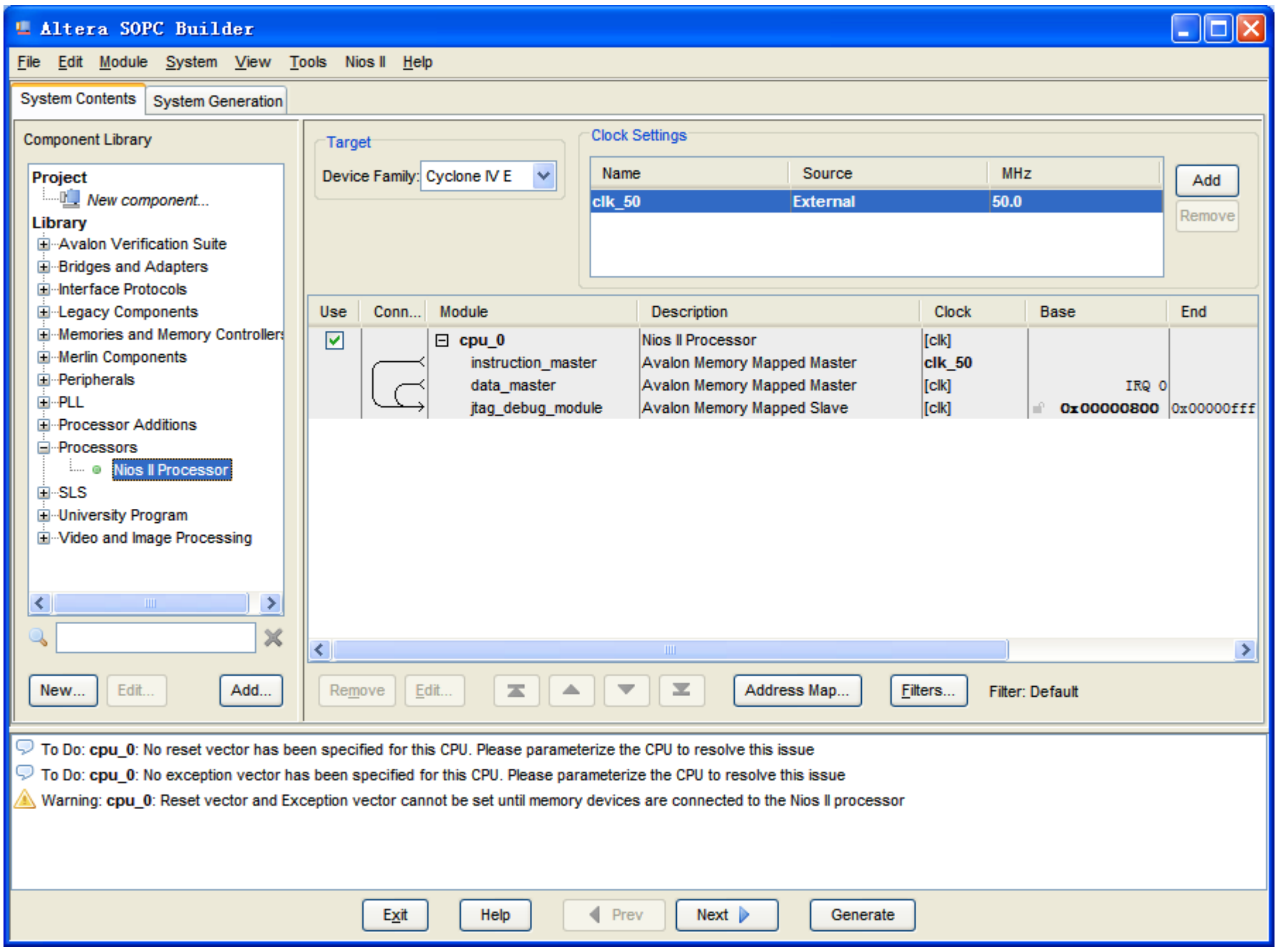
Fast TLB Miss Exception Vector: Memory: Offset: 0x0

☐ Include MPU

⚠ Warning: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

Figure 7-15 Nios II Processor

9. Click **Finish** to return to main window as shown in Figure 7-16.



Name	Source	MHz
clk_50	External	50.0

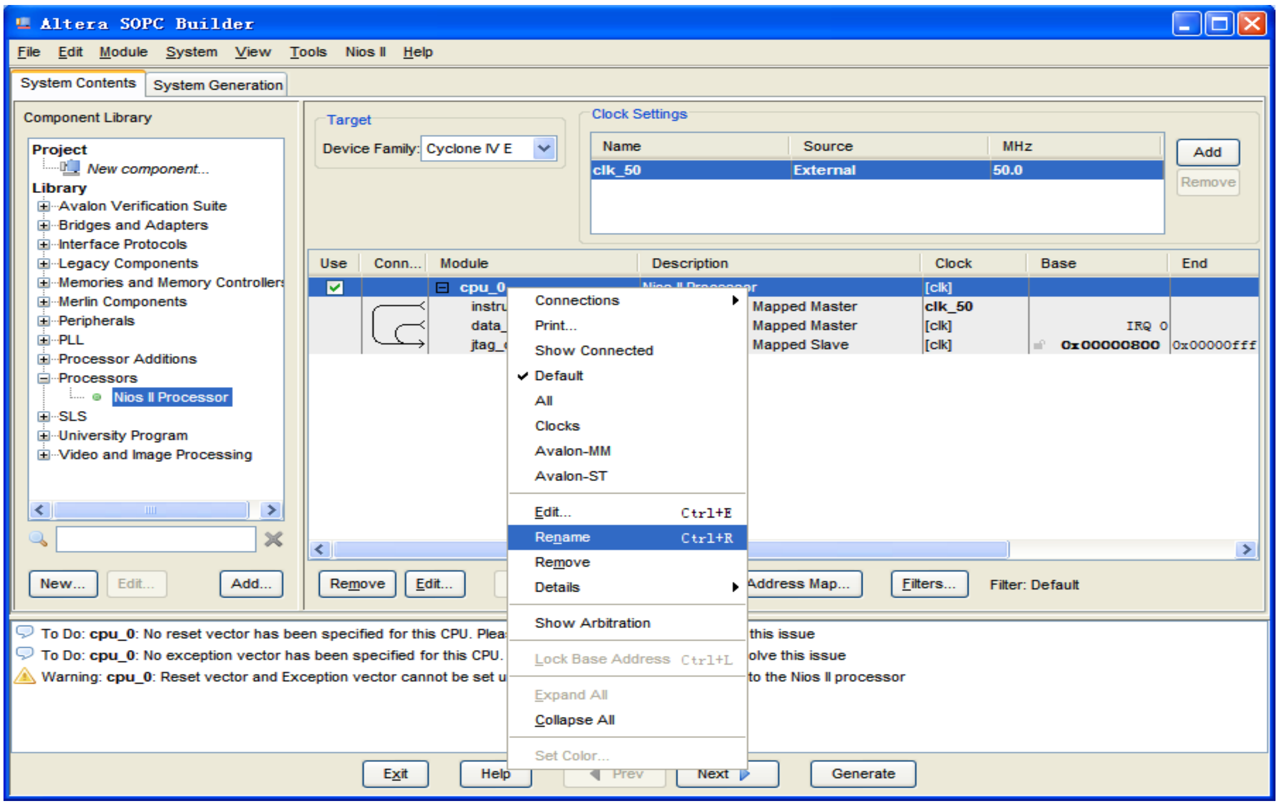
Use	Conn...	Module	Description	Clock	Base	End
✓		cpu_0	Nios II Processor	[clk]		
		instruction_master	Avalon Memory Mapped Master	clk_50		
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00000800	0x00000fff

To Do: **cpu_0**: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
To Do: **cpu_0**: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
Warning: **cpu_0**: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

Figure 7-16 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 7-17** and **Figure 7-18**.



Target
Device Family: Cyclone IV E

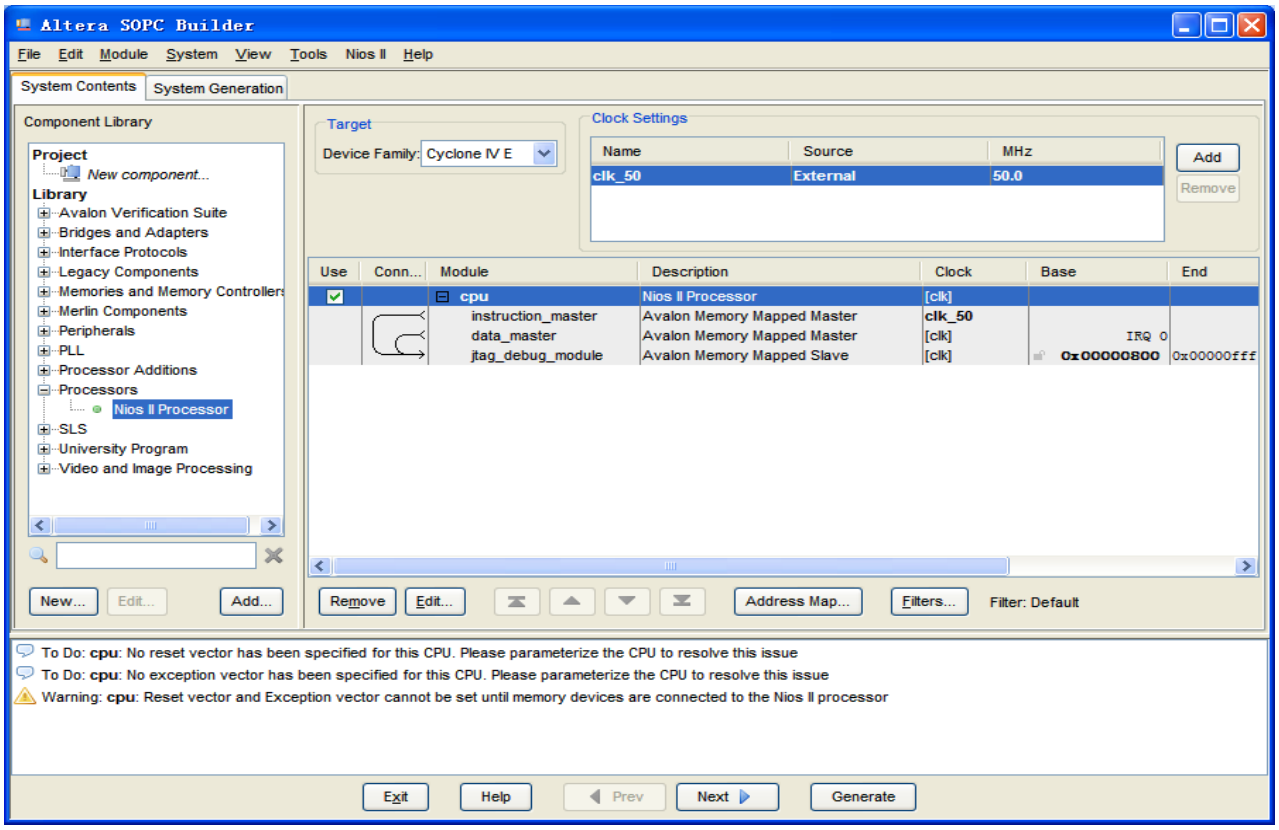
Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base	End
✓		cpu_0	Nios II Processor	[clk]		
		instru	Mapped Master	clk_50		
		data_	Mapped Master	[clk]	IRQ 0	
		jtag_	Mapped Slave	[clk]	0x00000800	0x00000fff

To Do: **cpu_0**: No reset vector has been specified for this CPU. Plea this issue
To Do: **cpu_0**: No exception vector has been specified for this CPU. olve this issue
Warning: **cpu_0**: Reset vector and Exception vector cannot be set u to the Nios II processor

Figure 7-17 Rename CPU name (1)


Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents System Generation
Component Library
Project
New component...
Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
Merlin Components
Peripherals
PLL
Processor Additions
Processors
Nios II Processor
SLS
University Program
Video and Image Processing
New... Edit... Add...
Remove Edit... Address Map... Filters... Filter: Default
Exit Help Prev Next Generate

Target
Device Family: Cyclone IV E

Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base	End
✓		cpu	Nios II Processor	[clk]		
		instruction_master	Avalon Memory Mapped Master	clk_50		
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00000800	0x00000fff

To Do: **cpu**: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
To Do: **cpu**: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
Warning: **cpu**: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

Figure 7-18 Rename CPU Name (2)

11. Choose **Library** > **Interface Protocols** > **Serial** > **JTAG UART** to open wizard of adding JTAG UART. See Figure 7-19 and Figure 7-20.

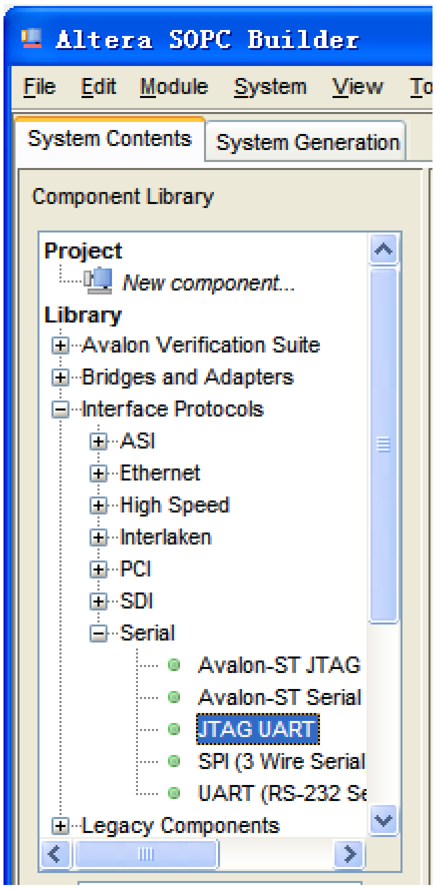
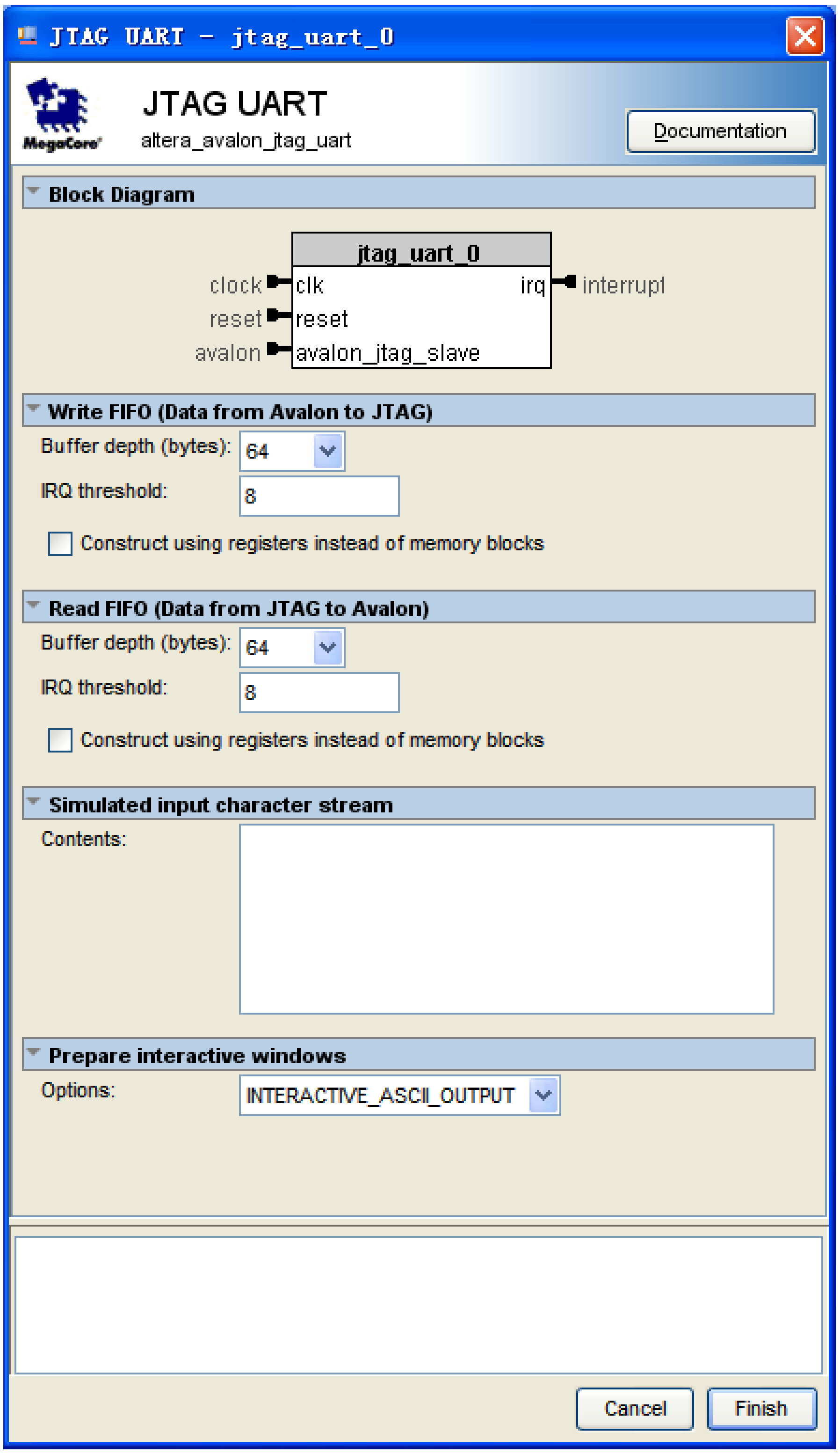


Figure 7-19 Add JTAG UART (1)



Write FIFO (Data from Avalon to JTAG)

Buffer depth (bytes): 64

IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Read FIFO (Data from JTAG to Avalon)

Buffer depth (bytes): 64

IRQ threshold: 8

☐ Construct using registers instead of memory blocks

Simulated input character stream

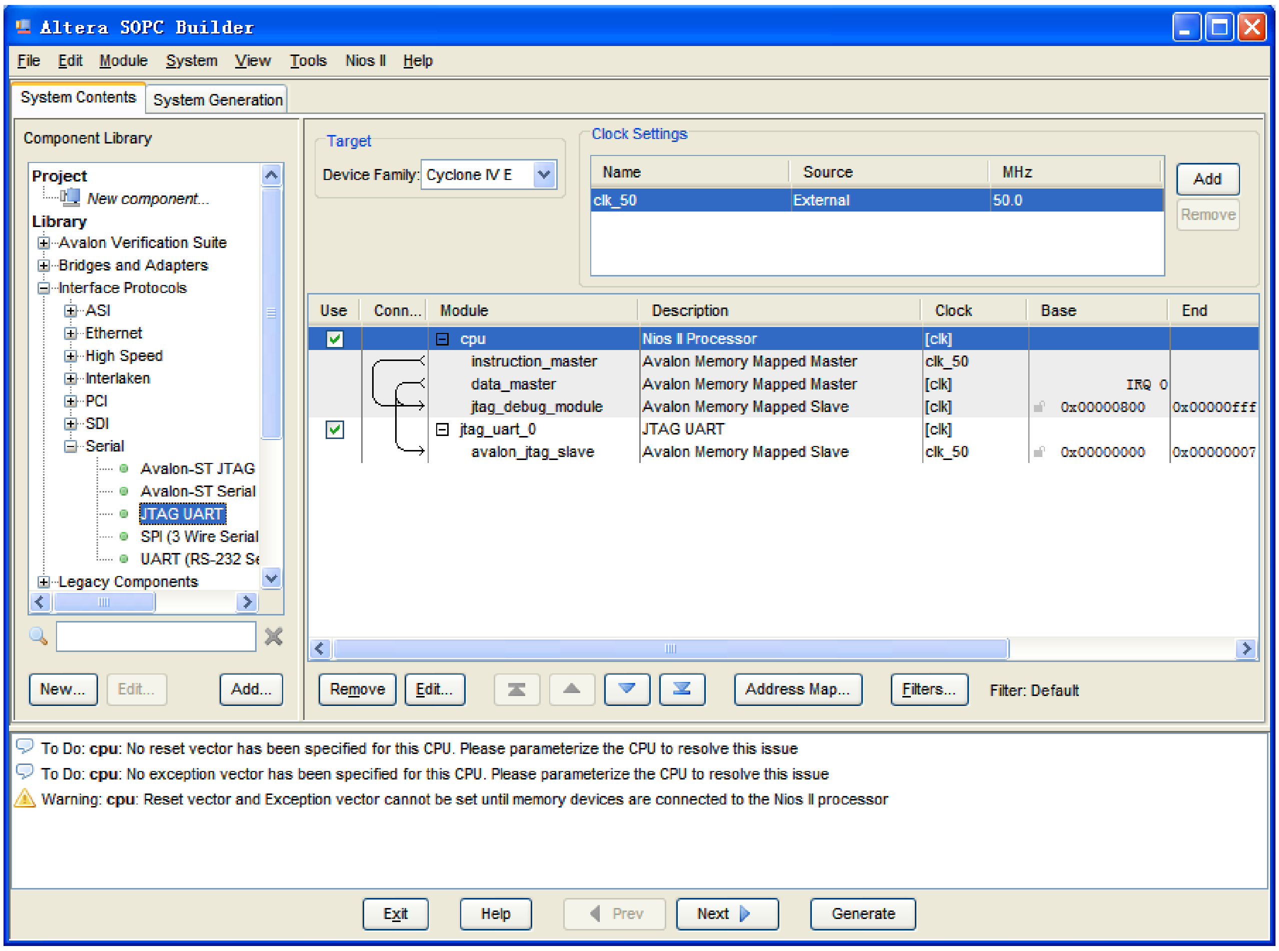
Contents:

Prepare interactive windows

Options: INTERACTIVE_ASCII_OUTPUT

Figure 7-20 JTAG UART

12. Click **Finish** to close the wizard and return to the window as shown in Figure 7-21.



Target — Device Family: Cyclone IV E

Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base	End
✓		cpu	Nios II Processor	[clk]		
		instruction_master	Avalon Memory Mapped Master	clk_50		
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00000800	0x00000fff
✓		jtag_uart_0	JTAG UART	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x00000000	0x00000007

Filter: Default

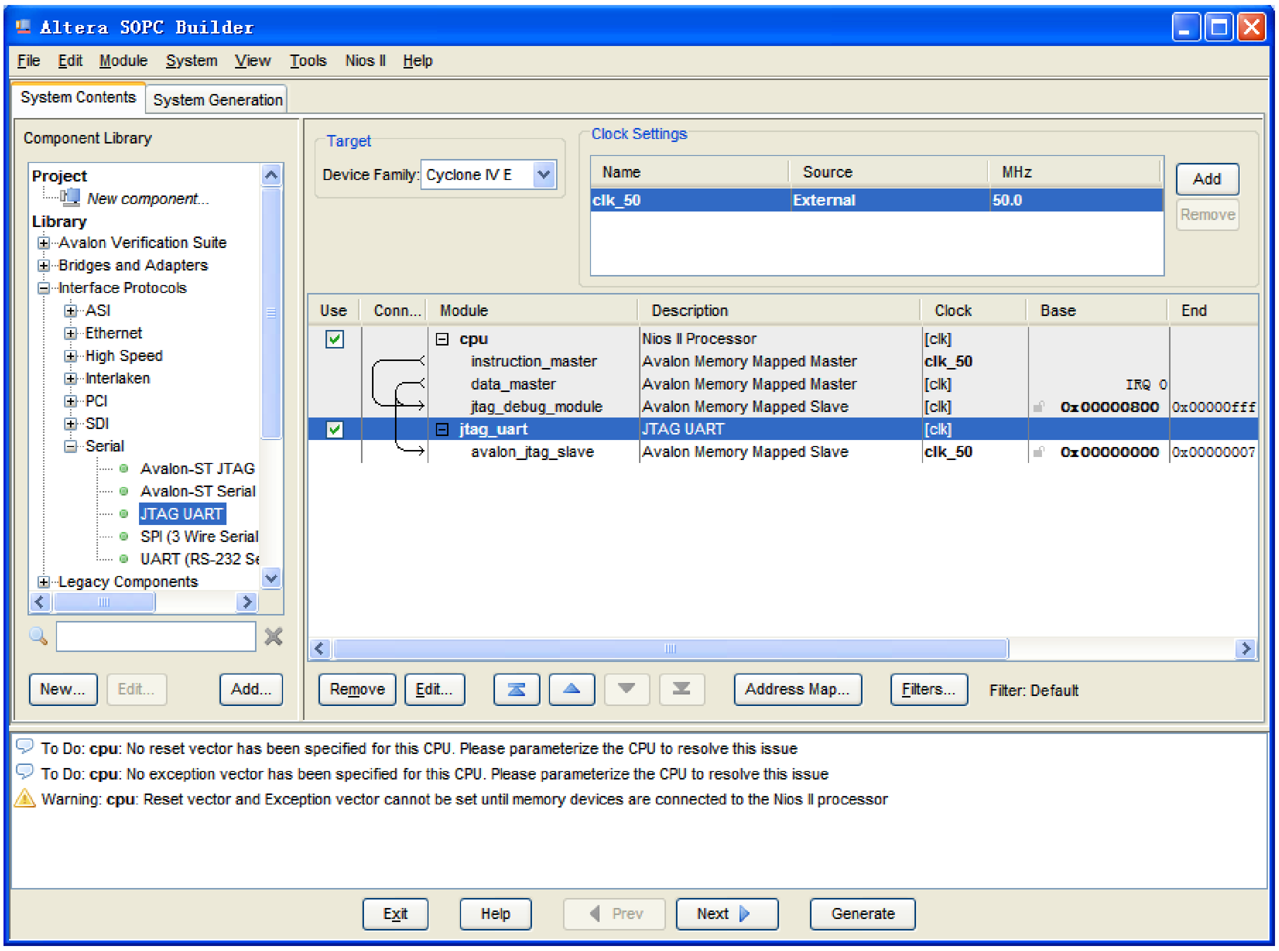
To Do: **cpu**: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue

To Do: **cpu**: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue

Warning: **cpu**: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

Figure 7-21 JTAG UART

13. Choose **jtag_uart_0** and rename it to **jtag_uart** as shown in Figure 7-22.



Target

Device Family: Cyclone IV E

Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base	End
✓		cpu	Nios II Processor	[clk]		
		instruction_master	Avalon Memory Mapped Master	clk_50		
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00000800	0x00000fff
✓		jtag_uart	JTAG UART	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x00000000	0x00000007

To Do: **cpu**: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue

To Do: **cpu**: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue

Warning: **cpu**: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

Figure 7-22 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 7-23** and **Figure 7-24**.

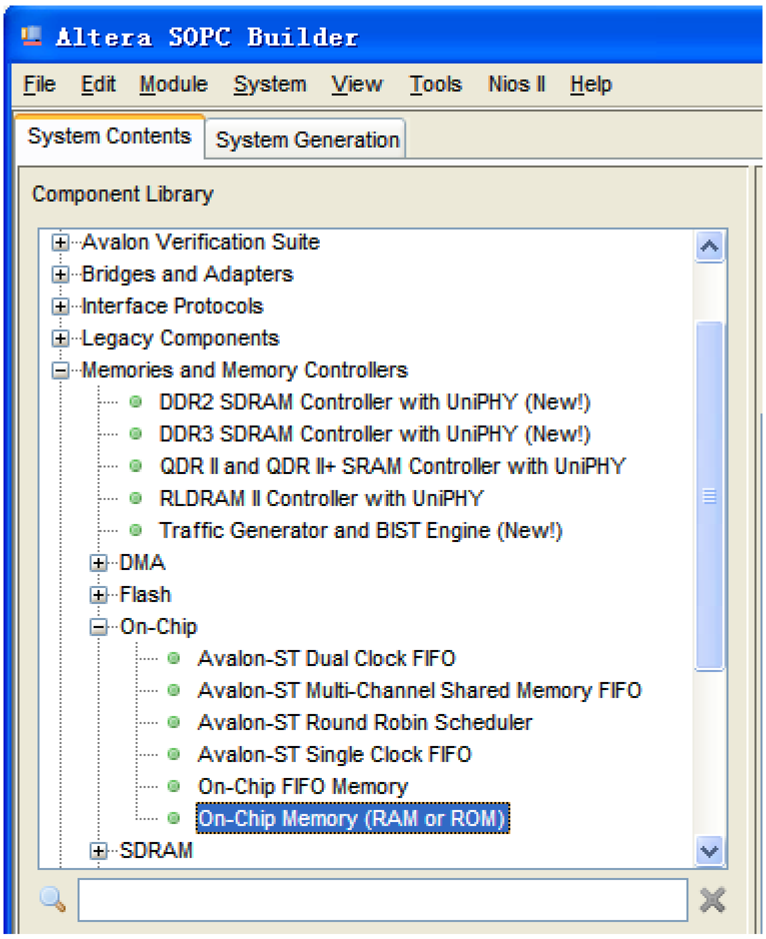
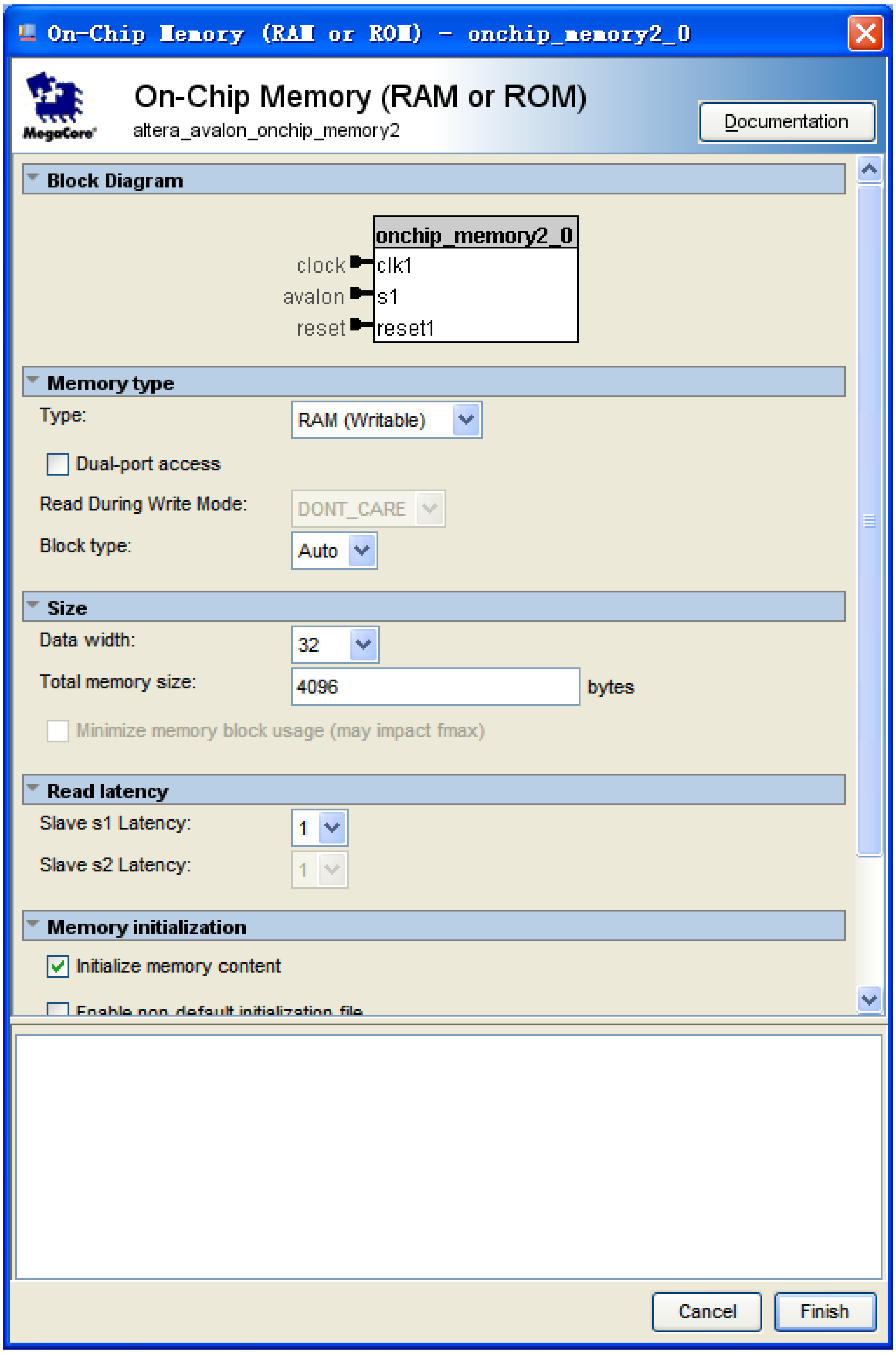

Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Component Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
DDR2 SDRAM Controller with UniPHY (New!)
DDR3 SDRAM Controller with UniPHY (New!)
QDR II and QDR II+ SRAM Controller with UniPHY
RLDRAM II Controller with UniPHY
Traffic Generator and BIST Engine (New!)
DMA
Flash
On-Chip
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory FIFO
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM

Figure 7-23 Add On-Chip Memory


On-Chip Memory (RAM or ROM) - onchip_memory2_0

On-Chip Memory (RAM or ROM)

altera_avalon_onchip_memory2

Memory type

Type: RAM (Writable)

☐ Dual-port access

Read During Write Mode: DONT_CARE

Block type: Auto

Size

Data width: 32

Total memory size: 4096 bytes

☐ Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1

Slave s2 Latency: 1

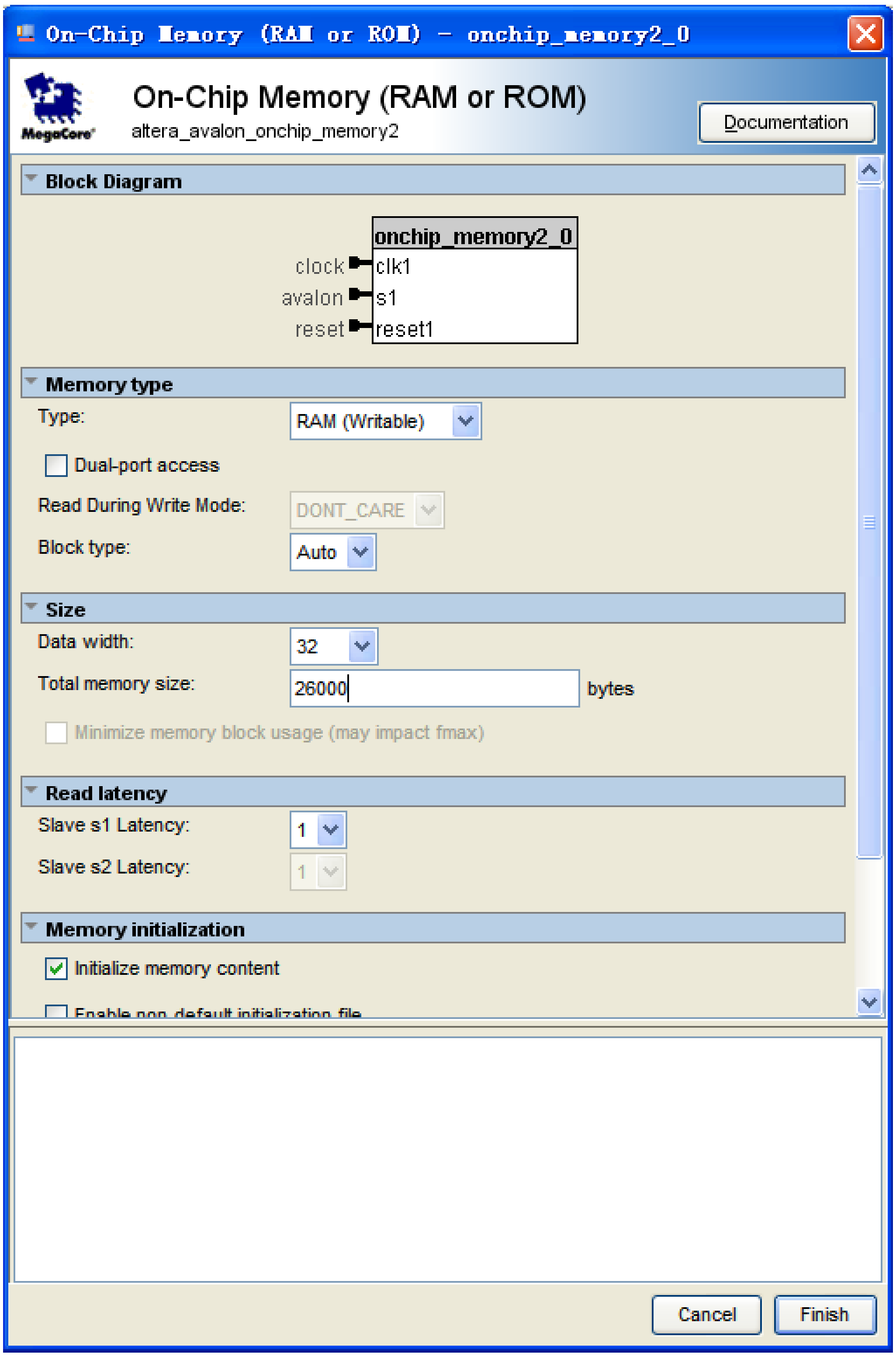
Memory initialization

☑ Initialize memory content

☐ Enable non-default initialization file

Figure 7-24 On-Chip Memory Box

16. Modify **Total memory size** to **26000** as shown in Figure 7-25. Click **Finish** to return to the window as in Figure 7-26.



On-Chip Memory (RAM or ROM) - onchip_memory2_0

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Memory type

Type: RAM (Writable)

☐ Dual-port access

Read During Write Mode: DONT_CARE

Block type: Auto

Size

Data width: 32

Total memory size: 26000 bytes

☐ Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1

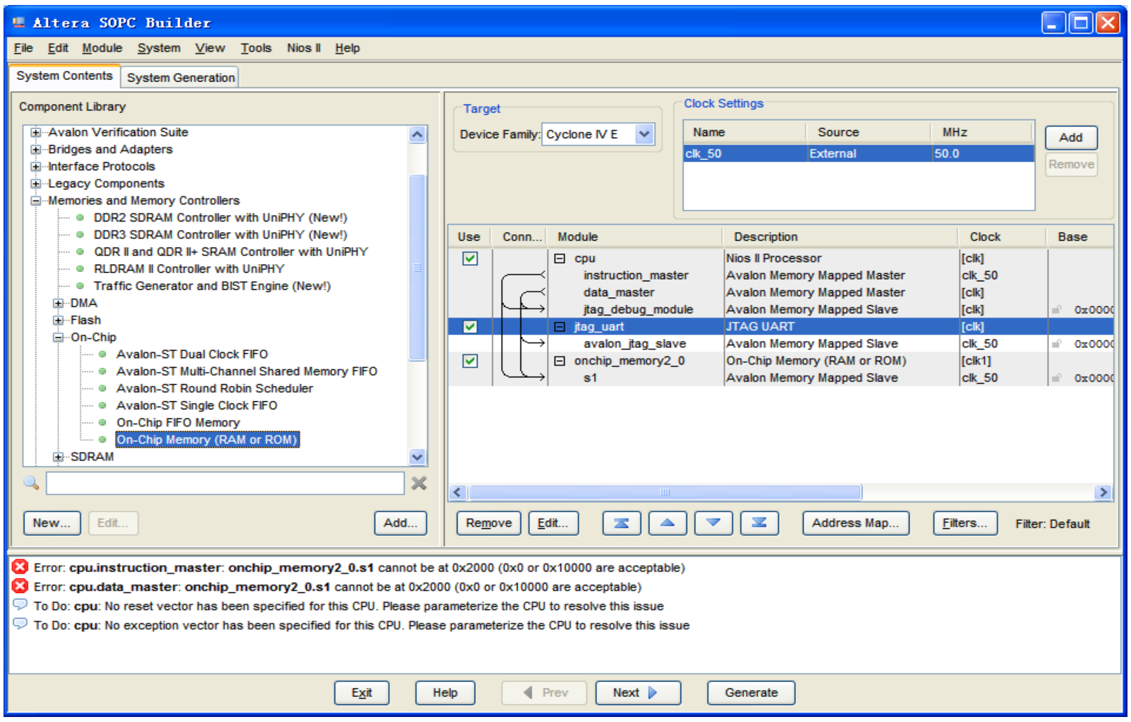
Slave s2 Latency: 1

Memory initialization

✓ Initialize memory content

☐ Enable non-default initialization file

Figure 7-25 Update Total memory size


Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents System Generation
Component Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
DDR2 SDRAM Controller with UniPHY (New!)
DDR3 SDRAM Controller with UniPHY (New!)
QDR II and QDR II+ SRAM Controller with UniPHY
RLDRAM II Controller with UniPHY
Traffic Generator and BIST Engine (New!)
DMA
Flash
On-Chip
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory FIFO
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM
New... Edit... Add...
Target
Clock Settings
Add Remove
Remove Edit... Address Map... Filters... Filter: Default
Exit Help Prev Next Generate

Device Family: Cyclone IV E

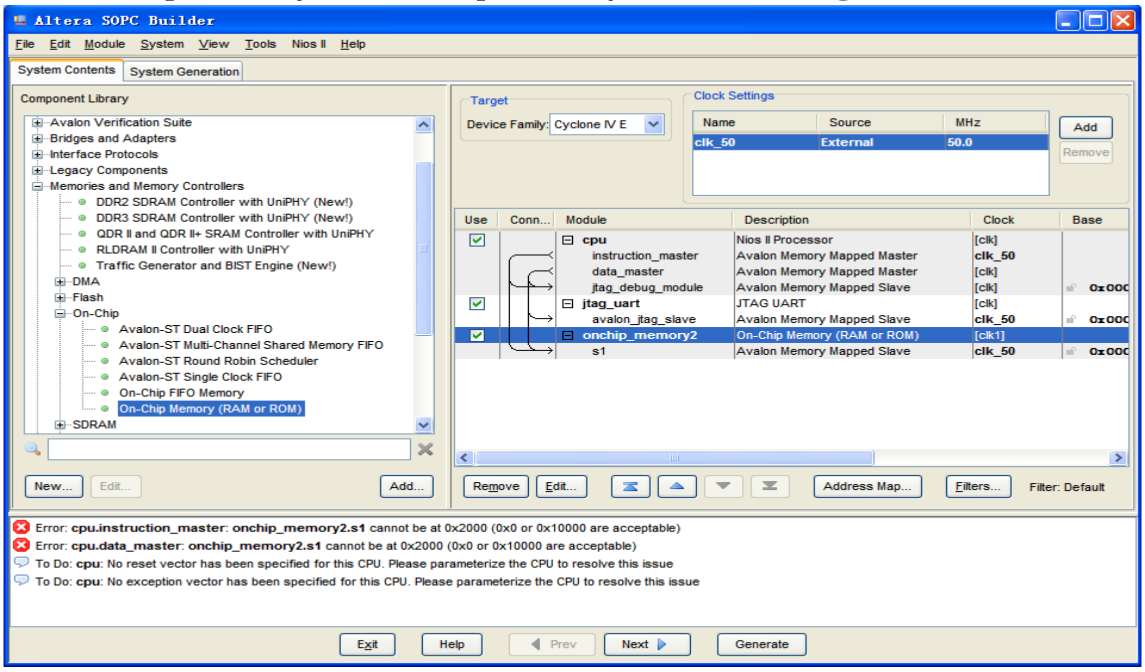
Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base
✓		cpu	Nios II Processor	[clk]	
		instruction_master	Avalon Memory Mapped Master	clk_50	
		data_master	Avalon Memory Mapped Master	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x0000
✓		jtag_uart	JTAG UART	[clk]	
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x0000
✓		onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]	
		s1	Avalon Memory Mapped Slave	clk_50	0x0000

```
Error: cpu.instruction_master: onchip_memory2_0.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: cpu.data_master: onchip_memory2_0.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
To Do: cpu: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
To Do: cpu: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
```

Figure 7-26 Add On-Chip memory

17. Rename **onchip_memory2_0** to **onchip_memory2** as shown in Figure 7-27.


Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents System Generation
Component Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
DDR2 SDRAM Controller with UniPHY (New!)
DDR3 SDRAM Controller with UniPHY (New!)
QDR II and QDR II+ SRAM Controller with UniPHY
RLDRAM II Controller with UniPHY
Traffic Generator and BIST Engine (New!)
DMA
Flash
On-Chip
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory FIFO
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM
New... Edit... Add...
Target
Clock Settings
Add Remove
Remove Edit... Address Map... Filters... Filter: Default
Exit Help Prev Next Generate

Device Family: Cyclone IV E

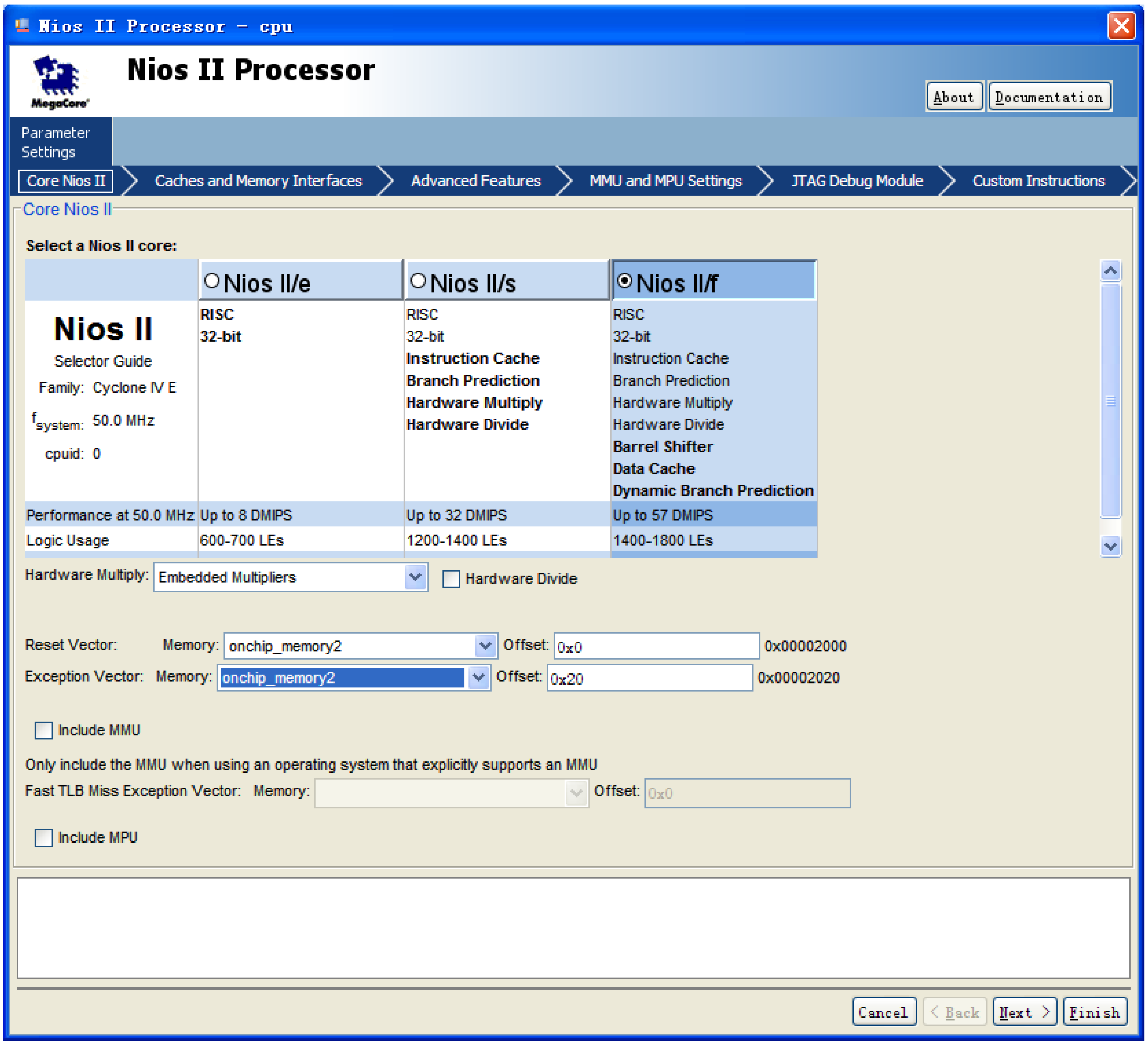
Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base
✓		cpu	Nios II Processor	[clk]	
		instruction_master	Avalon Memory Mapped Master	clk_50	
		data_master	Avalon Memory Mapped Master	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x000
✓		jtag_uart	JTAG UART	[clk]	
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x000
✓		onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000

```
Error: cpu.instruction_master: onchip_memory2.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: cpu.data_master: onchip_memory2.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
To Do: cpu: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
To Do: cpu: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
```

Figure 7-27 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 7-28. Then click **Finish** to return to the window as shown Figure 7-29.


Nios II Processor - cpu

Nios II Processor

Core Nios II

Select a Nios II core:

	Nios II/e	Nios II/s	[✓ Nios II/f]
Nios II Selector Guide Family: Cyclone IV E f_{system}: 50.0 MHz cpuid: 0	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Performance at 50.0 MHz	Up to 8 DMIPS	Up to 32 DMIPS	Up to 57 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs

Hardware Multiply: Embedded Multipliers ☐ Hardware Divide

Reset Vector: Memory: onchip_memory2 Offset: 0x0 0x00002000

Exception Vector: Memory: onchip_memory2 Offset: 0x20 0x00002020

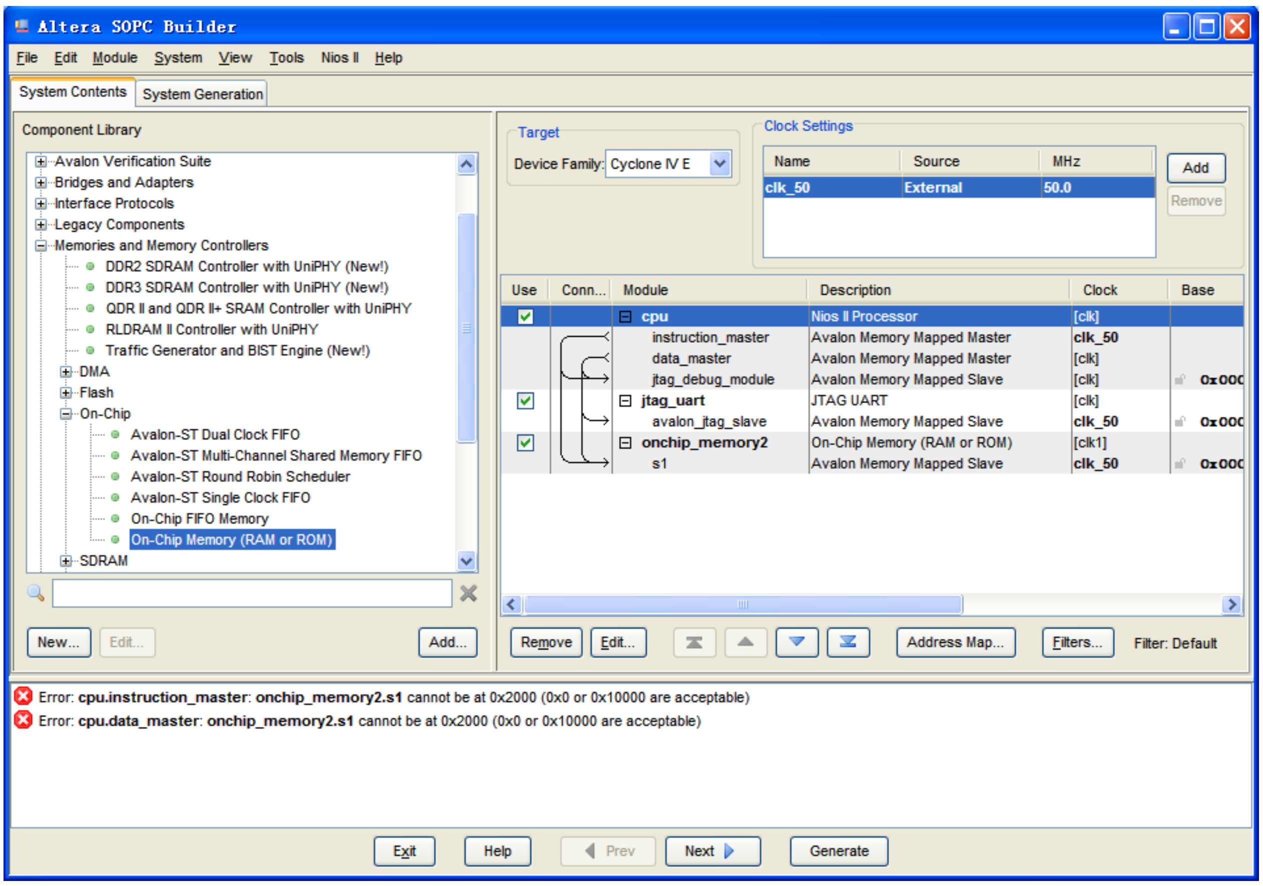
☐ Include MMU

Only include the MMU when using an operating system that explicitly supports an MMU

Fast TLB Miss Exception Vector: Memory: Offset: 0x0

☐ Include MPU

Figure 7-28 Update CPU settings


Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Component Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
DDR2 SDRAM Controller with UniPHY (New!)
DDR3 SDRAM Controller with UniPHY (New!)
QDR II and QDR II+ SRAM Controller with UniPHY
RLDRAM II Controller with UniPHY
Traffic Generator and BIST Engine (New!)
DMA
Flash
On-Chip
Avalon-ST Dual Clock FIFO
Avalon-ST Multi-Channel Shared Memory FIFO
Avalon-ST Round Robin Scheduler
Avalon-ST Single Clock FIFO
On-Chip FIFO Memory
On-Chip Memory (RAM or ROM)
SDRAM
New... Edit... Add...
Target
Clock Settings
Add Remove
Remove Edit... Address Map... Filters... Filter: Default
Exit Help Prev Next Generate

Device Family: Cyclone IV E

Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base
✓		cpu	Nios II Processor	[clk]	
		instruction_master	Avalon Memory Mapped Master	clk_50	
		data_master	Avalon Memory Mapped Master	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x000
✓		jtag_uart	JTAG UART	[clk]	
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x000
✓		onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000

```
Error: cpu.instruction_master: onchip_memory2.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: cpu.data_master: onchip_memory2.s1 cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
```

Figure 7-29 Update CPU settings Completely

19. Choose **Library** > **Peripherals** > **Microcontroller Peripherals** >**PIO (Paralle I/O)** to open wizard of adding PIO. See **Figure 7-30** and **Figure 7-31**.

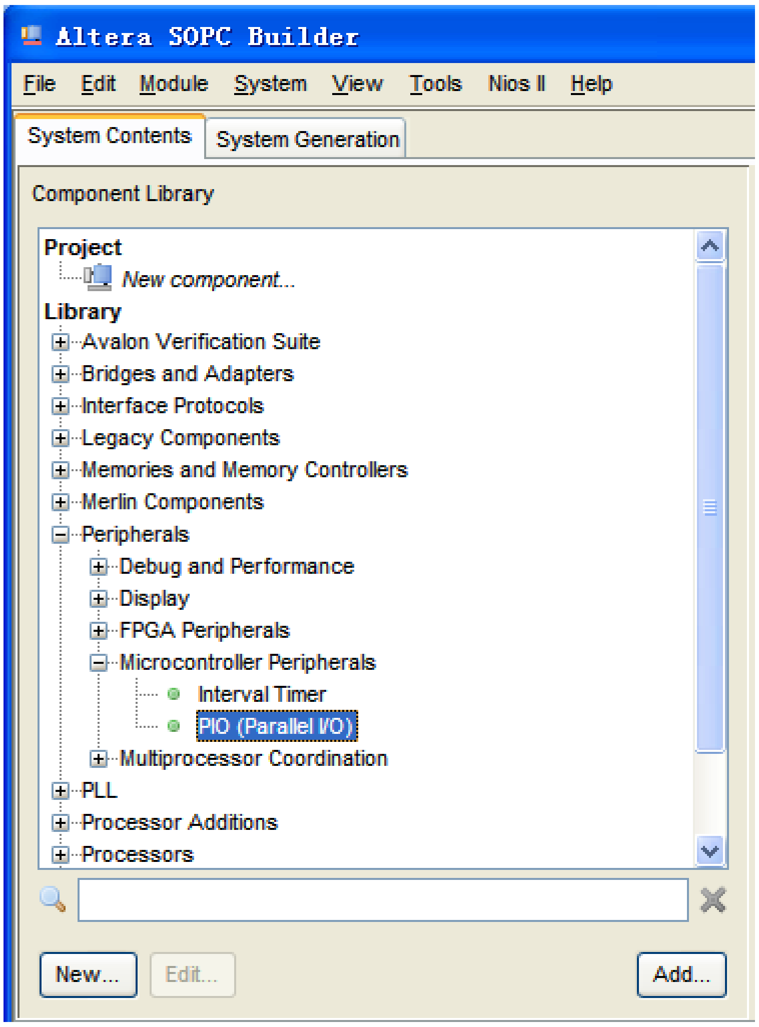
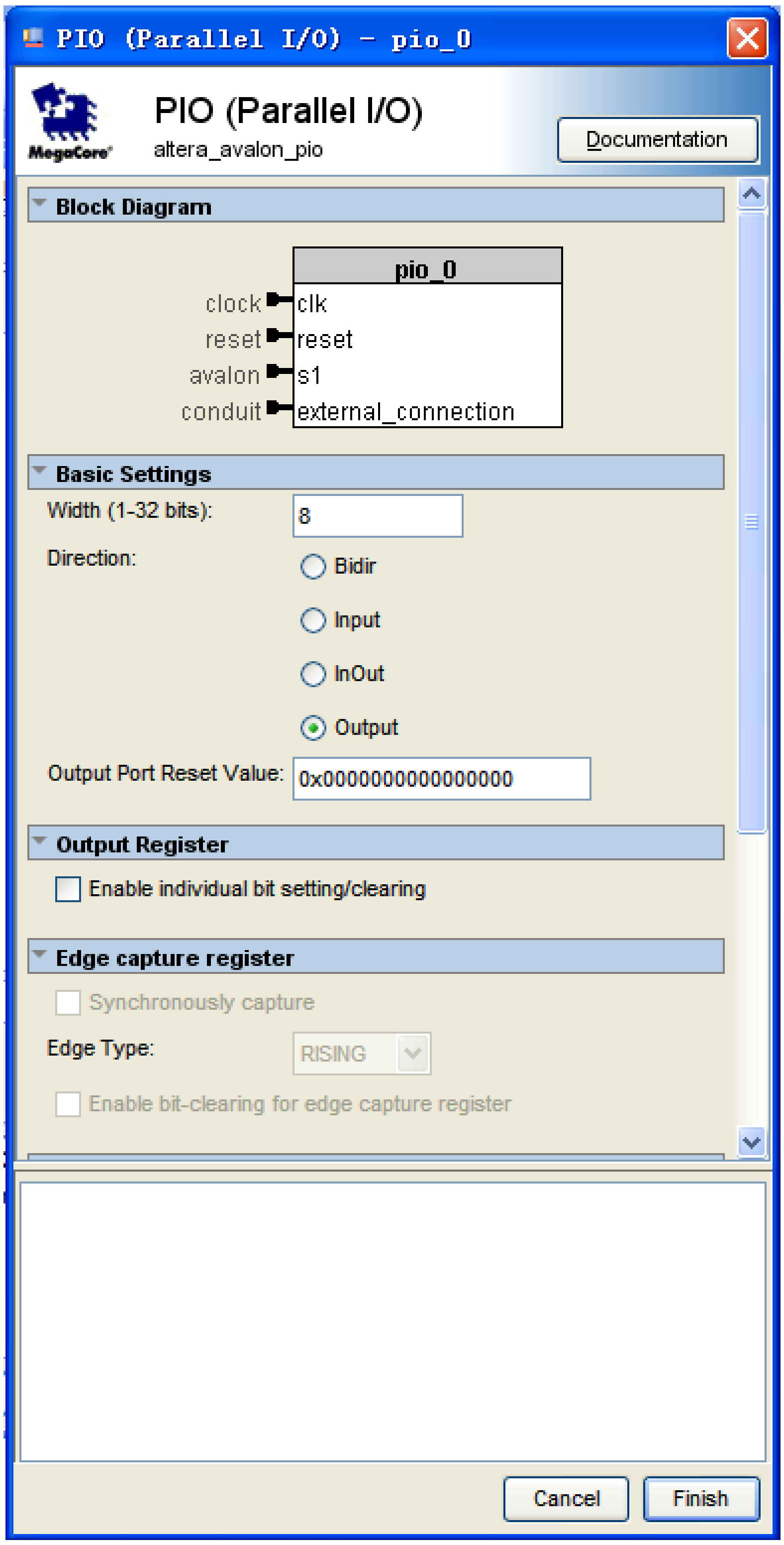


Figure 7-30 Add PIO


PIO (Parallel I/O) - pio_0

PIO (Parallel I/O)
altera_avalon_pio

Block Diagram

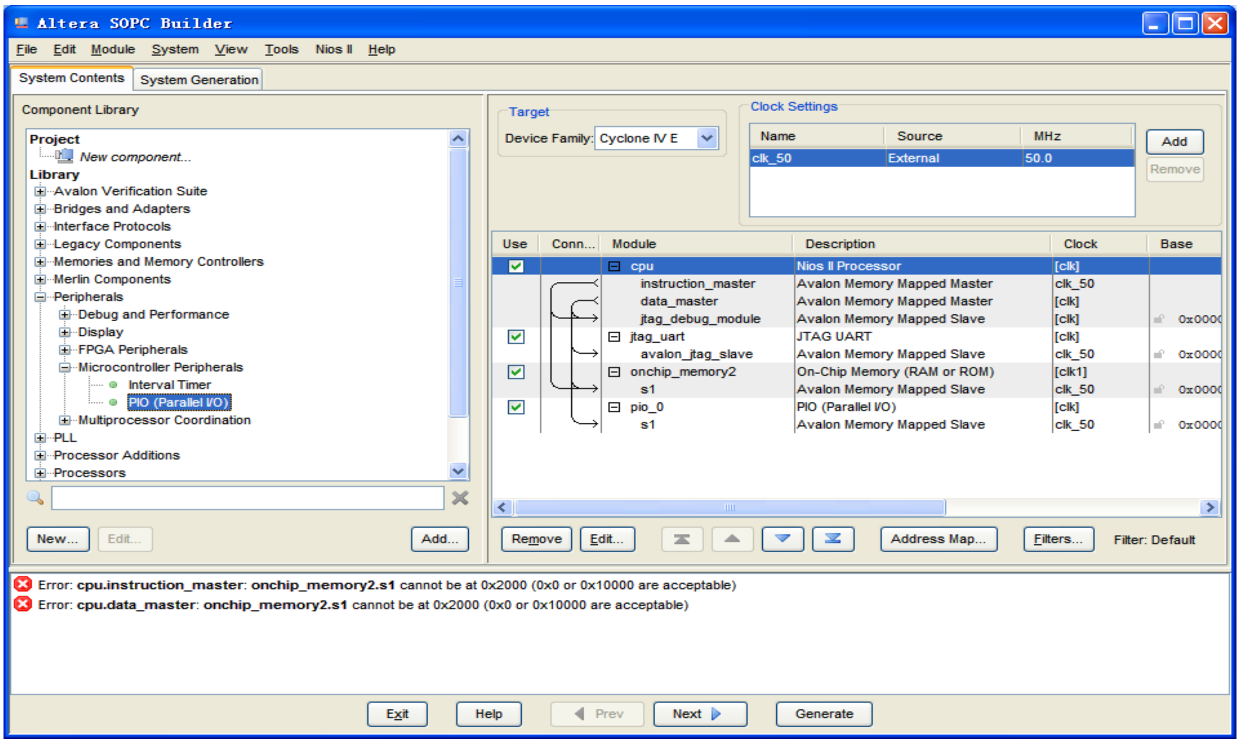
Basic Settings
Width (1-32 bits): 8
Direction: ( ) Bidir ( ) Input ( ) InOut [✓ Output]
Output Port Reset Value: 0x0000000000000000

Output Register
[ ] Enable individual bit setting/clearing

Edge capture register
[ ] Synchronously capture
Edge Type: RISING
[ ] Enable bit-clearing for edge capture register

Figure 7-31 Add PIO

20. Click **Finish** to close PIO box and return to the window as shown in Figure 7-32.



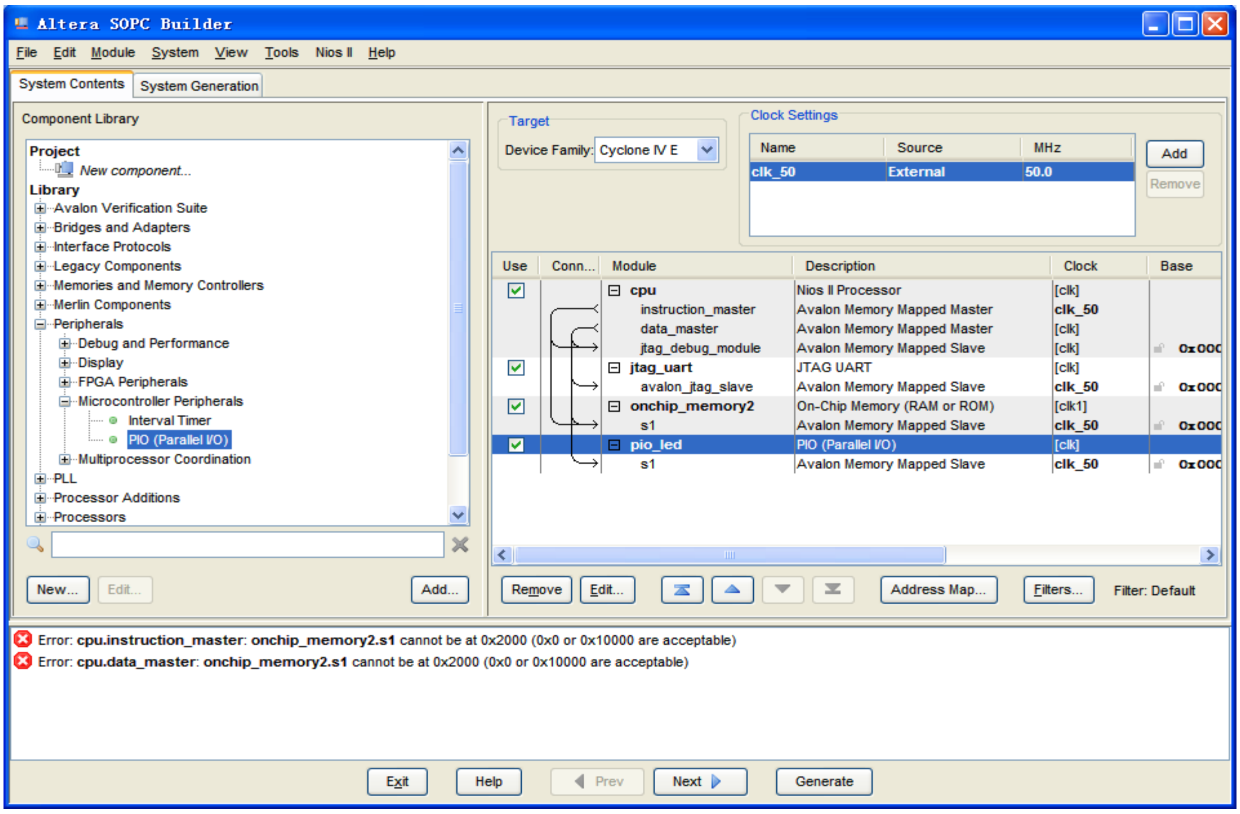
Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base
✓		cpu	Nios II Processor	[clk]	
		instruction_master	Avalon Memory Mapped Master	clk_50	
		data_master	Avalon Memory Mapped Master	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x000
✓		jtag_uart	JTAG UART	[clk]	
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x000
✓		onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000
✓		pio_0	PIO (Parallel I/O)	[clk]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000

Error: **cpu.instruction_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: **cpu.data_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)

Figure 7-32 PIO

21. Rename **pio_0** to **pio_led** as shown in **Figure 7-33**.



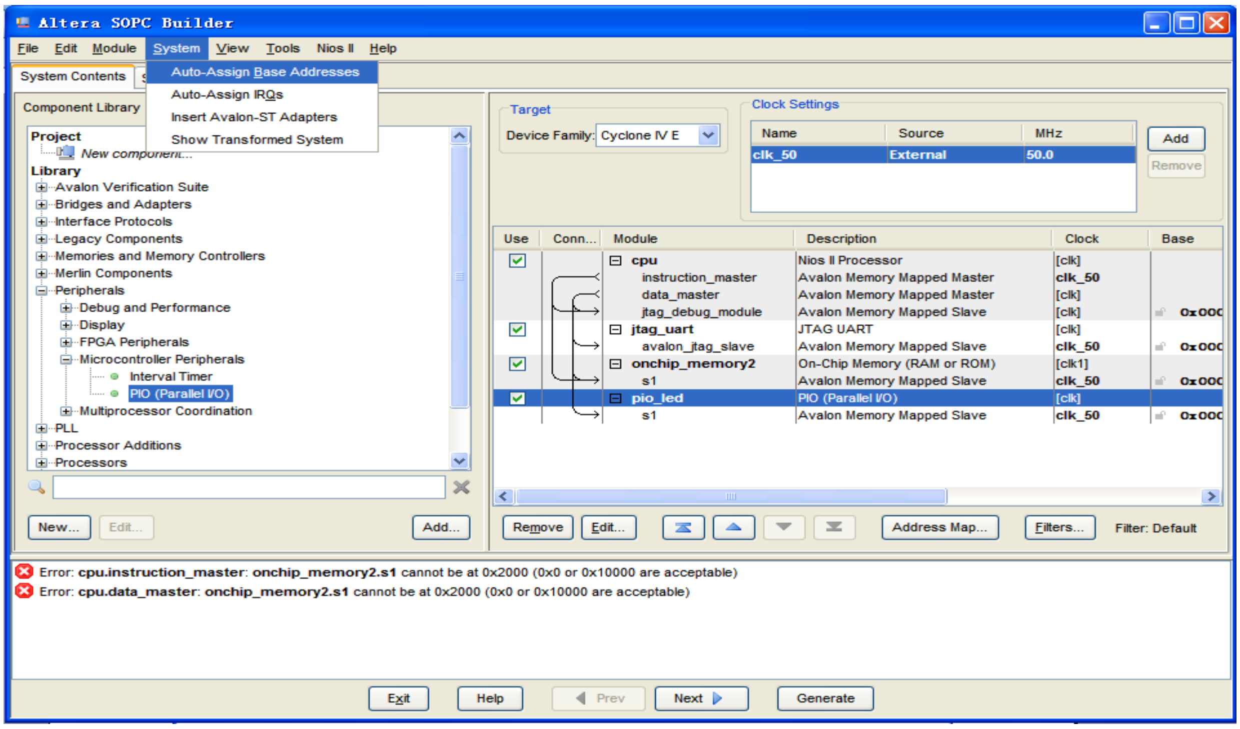
Name	Source	MHz
clk_50	External	50.0

Use	Conn...	Module	Description	Clock	Base
✓		cpu	Nios II Processor	[clk]	
		instruction_master	Avalon Memory Mapped Master	clk_50	
		data_master	Avalon Memory Mapped Master	[clk]	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x000
✓		jtag_uart	JTAG UART	[clk]	
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x000
✓		onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000
✓		pio_led	PIO (Parallel I/O)	[clk]	
		s1	Avalon Memory Mapped Slave	clk_50	0x000

Error: **cpu.instruction_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: **cpu.data_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)

Figure 7-33 Rename PIO

22. Choose **System** > **Auto-Assign Base Addresses** as shown in **Figure 7-34**. Then choose **File** > **Refresh System**. After that you will find that there is no error in the message window as shown in **Figure 7-35**.



Target — Device Family: Cyclone IV E

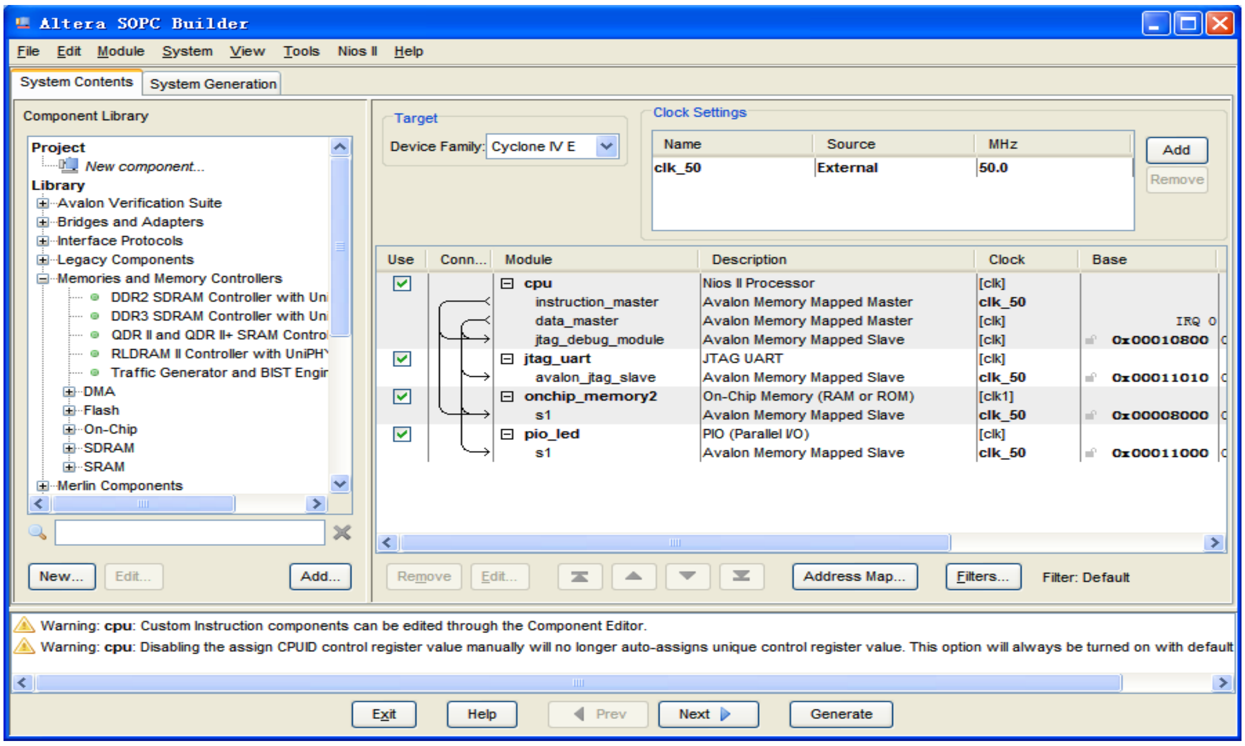
Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Module	Description	Clock	Base
✓	cpu	Nios II Processor	[clk]	
	instruction_master	Avalon Memory Mapped Master	clk_50	
	data_master	Avalon Memory Mapped Master	[clk]	
	jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x000
✓	jtag_uart	JTAG UART	[clk]	
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x000
✓	onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
	s1	Avalon Memory Mapped Slave	clk_50	0x000
✓	pio_led	PIO (Parallel I/O)	[clk]	
	s1	Avalon Memory Mapped Slave	clk_50	0x000

Error: **cpu.instruction_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)
Error: **cpu.data_master**: **onchip_memory2.s1** cannot be at 0x2000 (0x0 or 0x10000 are acceptable)

Figure 7-34 Auto-Assign Base Addresses



Altera SOPC Builder
File Edit Module System View Tools Nios II Help
System Contents System Generation
Component Library
Project
New component...
Library
Avalon Verification Suite
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
DDR2 SDRAM Controller with Uni
DDR3 SDRAM Controller with Uni
QDR II and QDR II+ SRAM Contro
RLDRAM II Controller with UniPHY
Traffic Generator and BIST Engir
DMA
Flash
On-Chip
SDRAM
SRAM
Merlin Components
New... Edit... Add...
Remove Edit... Address Map... Filters... Filter: Default
Add Remove
Exit Help Prev Next Generate

Target — Device Family: Cyclone IV E

Clock Settings

Name	Source	MHz
clk_50	External	50.0

Use	Module	Description	Clock	Base
✓	cpu	Nios II Processor	[clk]	
	instruction_master	Avalon Memory Mapped Master	clk_50	
	data_master	Avalon Memory Mapped Master	[clk]	IRQ 0
	jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00010800
✓	jtag_uart	JTAG UART	[clk]	
	avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x00011010
✓	onchip_memory2	On-Chip Memory (RAM or ROM)	[clk1]	
	s1	Avalon Memory Mapped Slave	clk_50	0x00008000
✓	pio_led	PIO (Parallel I/O)	[clk]	
	s1	Avalon Memory Mapped Slave	clk_50	0x00011000

Warning: **cpu**: Custom Instruction components can be edited through the Component Editor.
Warning: **cpu**: Disabling the assign CPUID control register value manually will no longer auto-assigns unique control register value. This option will always be turned on with default

Figure 7-35 No errors or warnings

23. Click **Generate** and then pop a window as shown in Figure 7-36. Click **Save** and at the time a windows as Figure 7-37 will appear, please input the name and save, then the compilation start. If there are no errors in the generation, the window will show successful as shown in Figure 7-38.

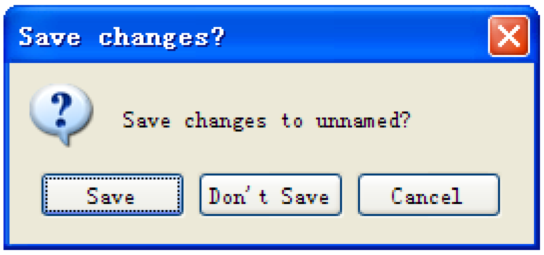


Figure 7-36 Generate SOPC

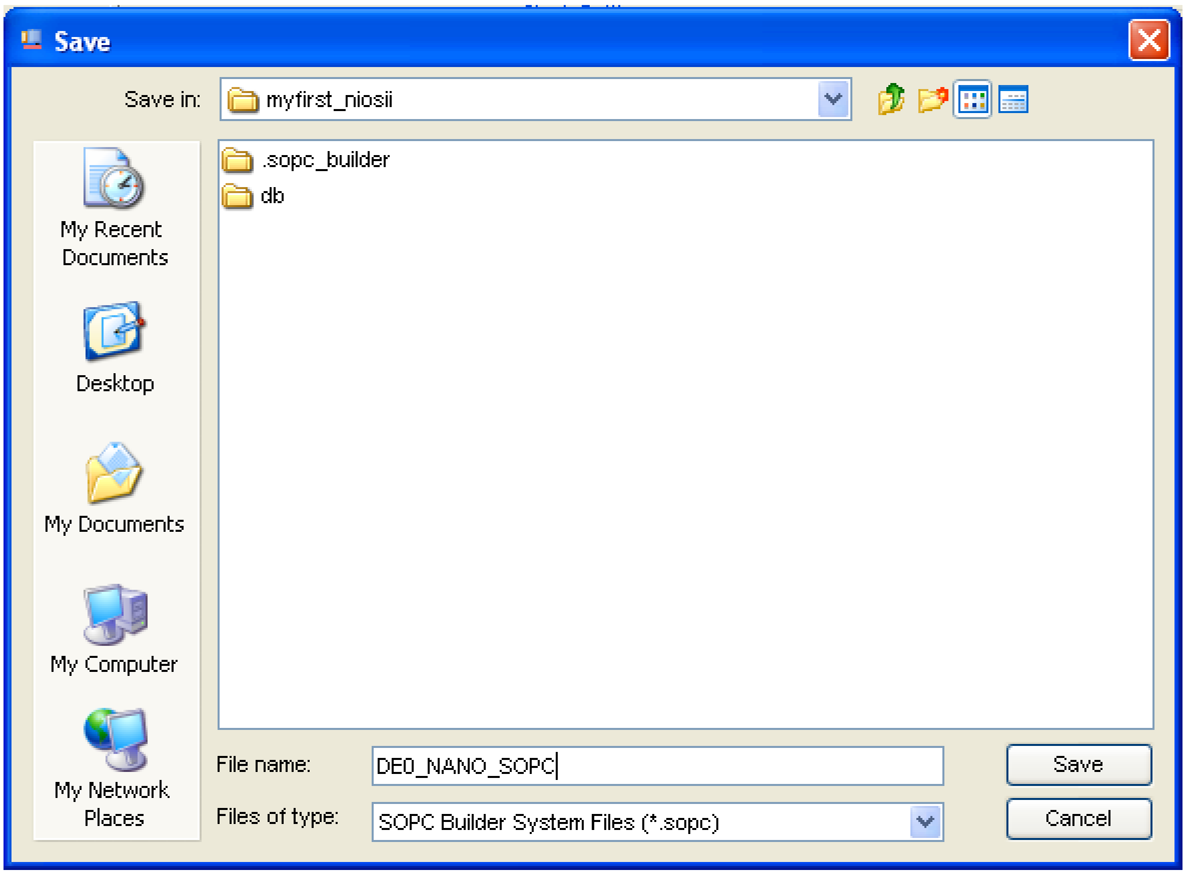
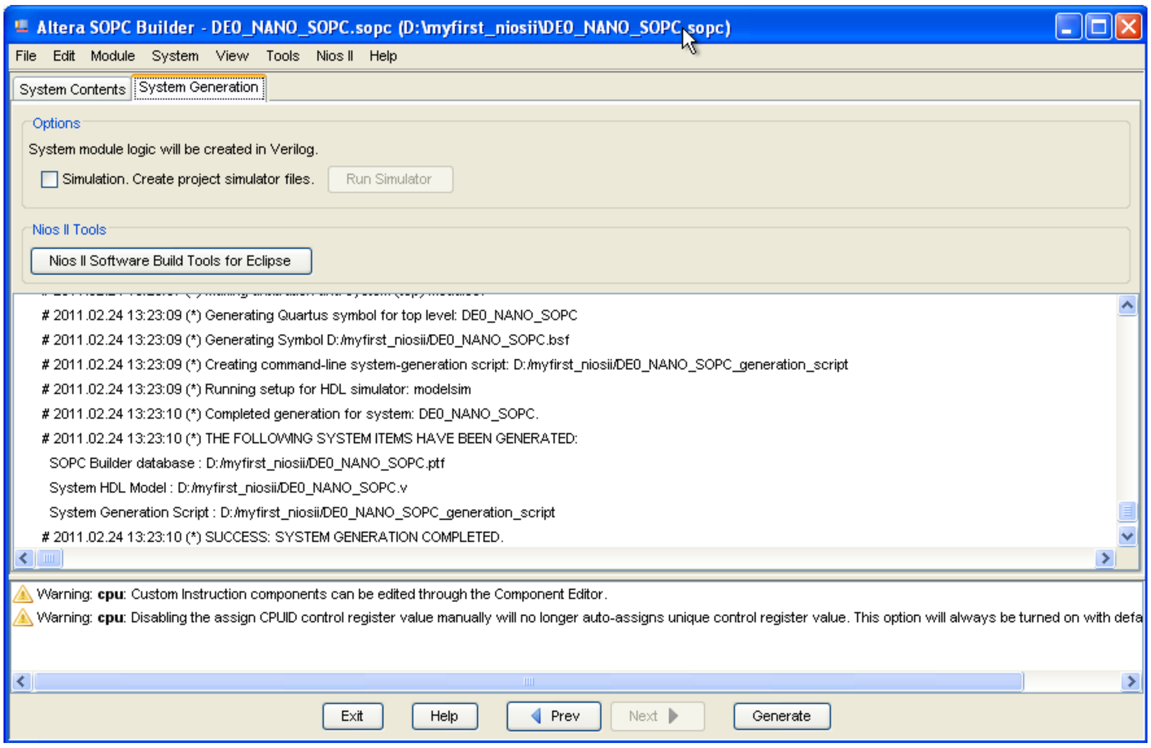

Save in: myfirst_niosii
.sopc_builder
db
File name: DE0_NANO_SOPC
Files of type: SOPC Builder System Files (*.sopc)

Figure 7-37 Generate SOPC


Altera SOPC Builder - DE0_NANO_SOPC.sopc (D:\myfirst_niosii\DE0_NANO_SOPC.sopc)
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Options
Run Simulator
Nios II Tools
Nios II Software Build Tools for Eclipse
Exit
Help
Prev
Next
Generate

System module logic will be created in Verilog.

☐ Simulation. Create project simulator files.

```
# 2011.02.24 13:23:09 (*) Generating Quartus symbol for top level: DE0_NANO_SOPC
# 2011.02.24 13:23:09 (*) Generating Symbol D:/myfirst_niosii/DE0_NANO_SOPC.bsf
# 2011.02.24 13:23:09 (*) Creating command-line system-generation script: D:/myfirst_niosii/DE0_NANO_SOPC_generation_script
# 2011.02.24 13:23:09 (*) Running setup for HDL simulator: modelsim
# 2011.02.24 13:23:10 (*) Completed generation for system: DE0_NANO_SOPC.
# 2011.02.24 13:23:10 (*) THE FOLLOWING SYSTEM ITEMS HAVE BEEN GENERATED:
  SOPC Builder database : D:/myfirst_niosii/DE0_NANO_SOPC.ptf
  System HDL Model : D:/myfirst_niosii/DE0_NANO_SOPC.v
  System Generation Script : D:/myfirst_niosii/DE0_NANO_SOPC_generation_script
# 2011.02.24 13:23:10 (*) SUCCESS: SYSTEM GENERATION COMPLETED.
```

```
Warning: cpu: Custom Instruction components can be edited through the Component Editor.
Warning: cpu: Disabling the assign CPUID control register value manually will no longer auto-assigns unique control register value. This option will always be turned on with defa
```

Figure 7-38 Generate SOPC Completely

24. Click **Exit** to exit the SOPC Builder and return to the window as shown in Figure 7-39.

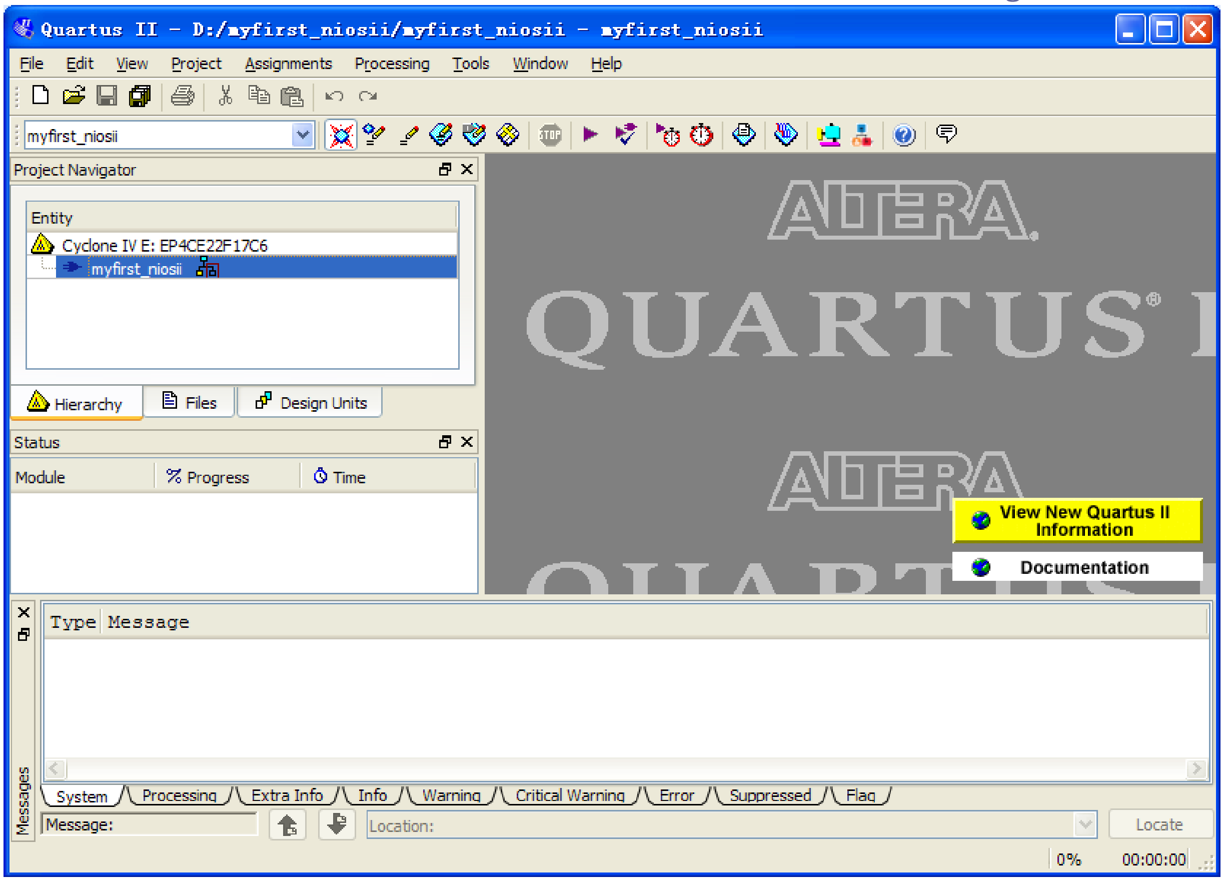


Figure 7-39 Exit SOPC

25. Choose **File** > **New** to open new files wizard. See Figure 7-40 and Figure 7-41.

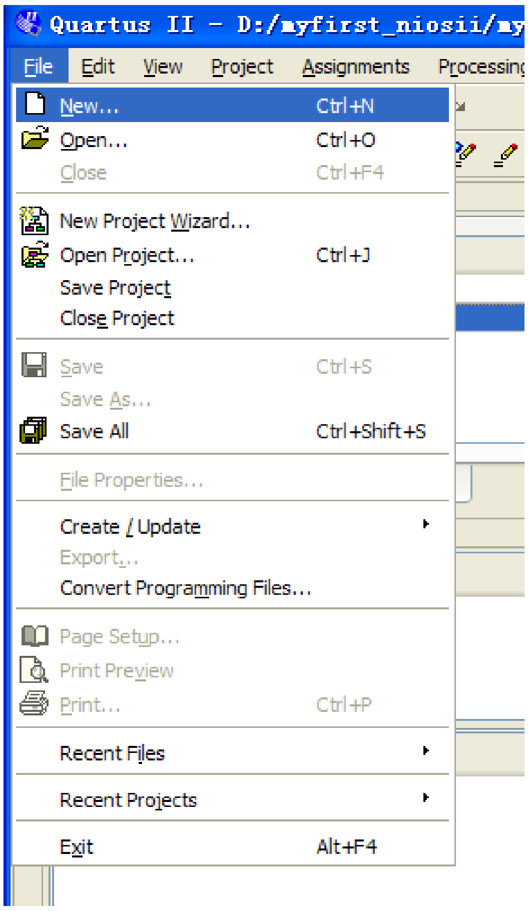


Figure 7-40 New Verilog file

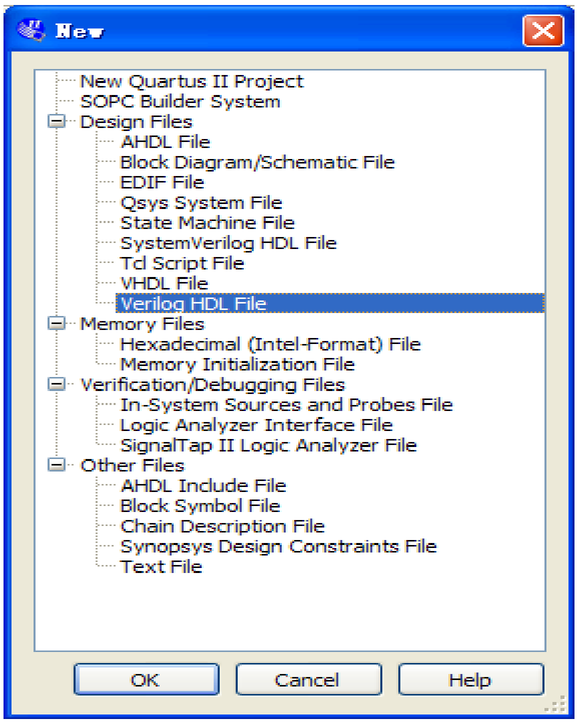


Figure 7-41 New Verilog File

33. Choose **Verilog HDL File** and click **OK** to return to the window as shown in Figure 7-42.

Figure 7-42 show a blank verilog file.

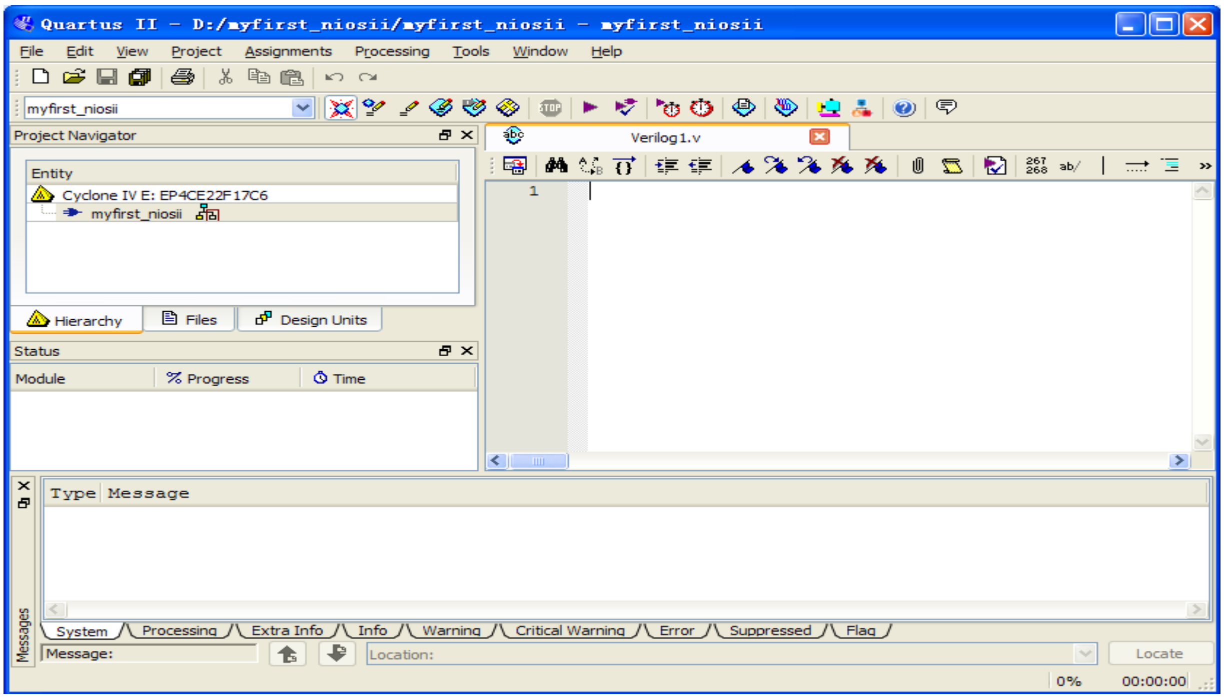


Figure 7-42 A blank verilog file

34. Type verilog the following script as shown in Figure 7-43. The module **DE0_NANO_SOPC** of the code is from **DE0_NANO_SOPC.v** of the project. See

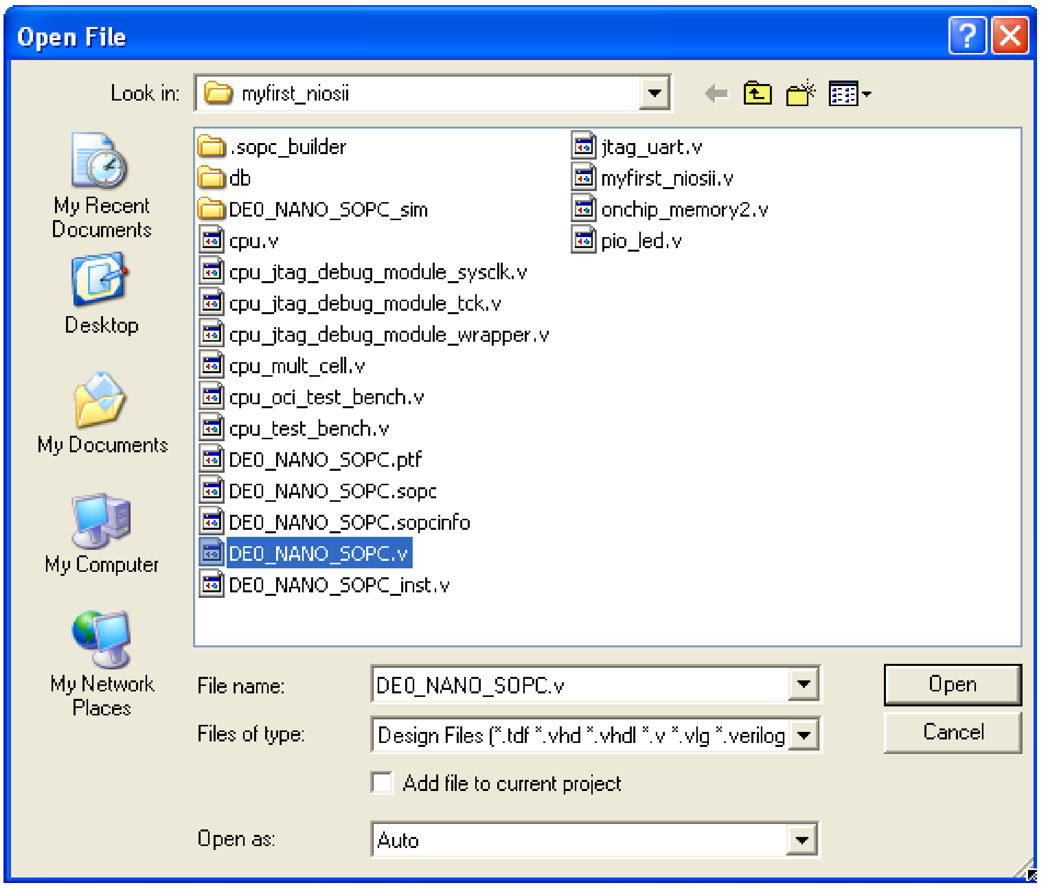
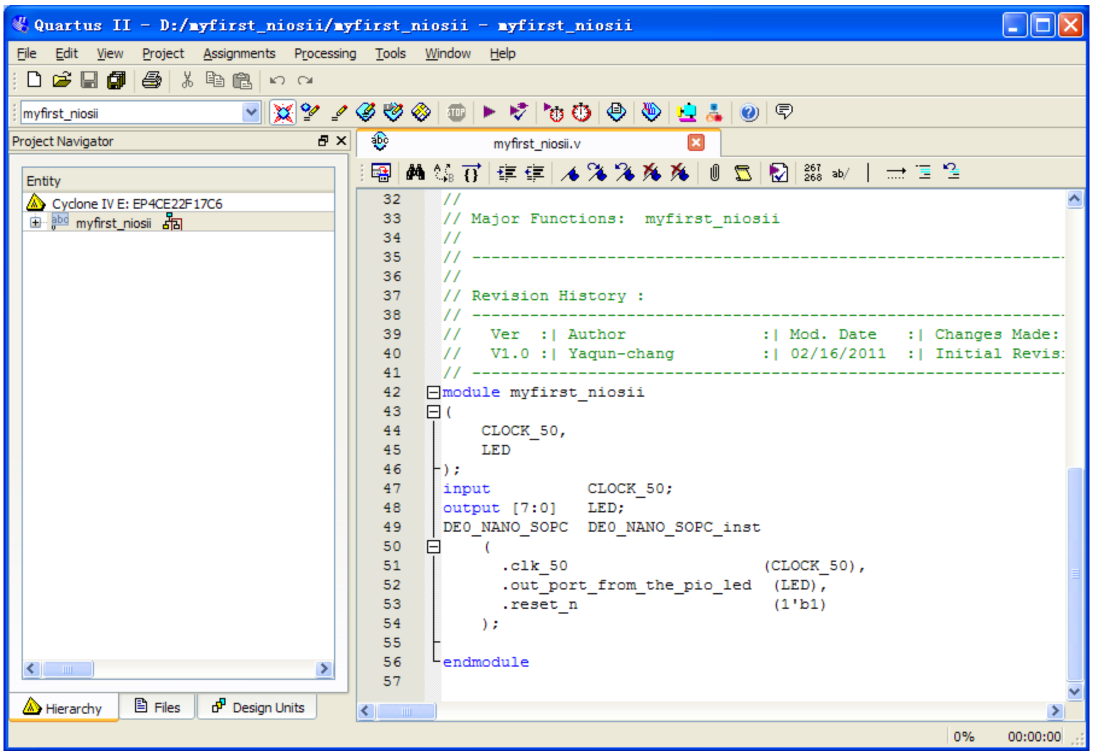


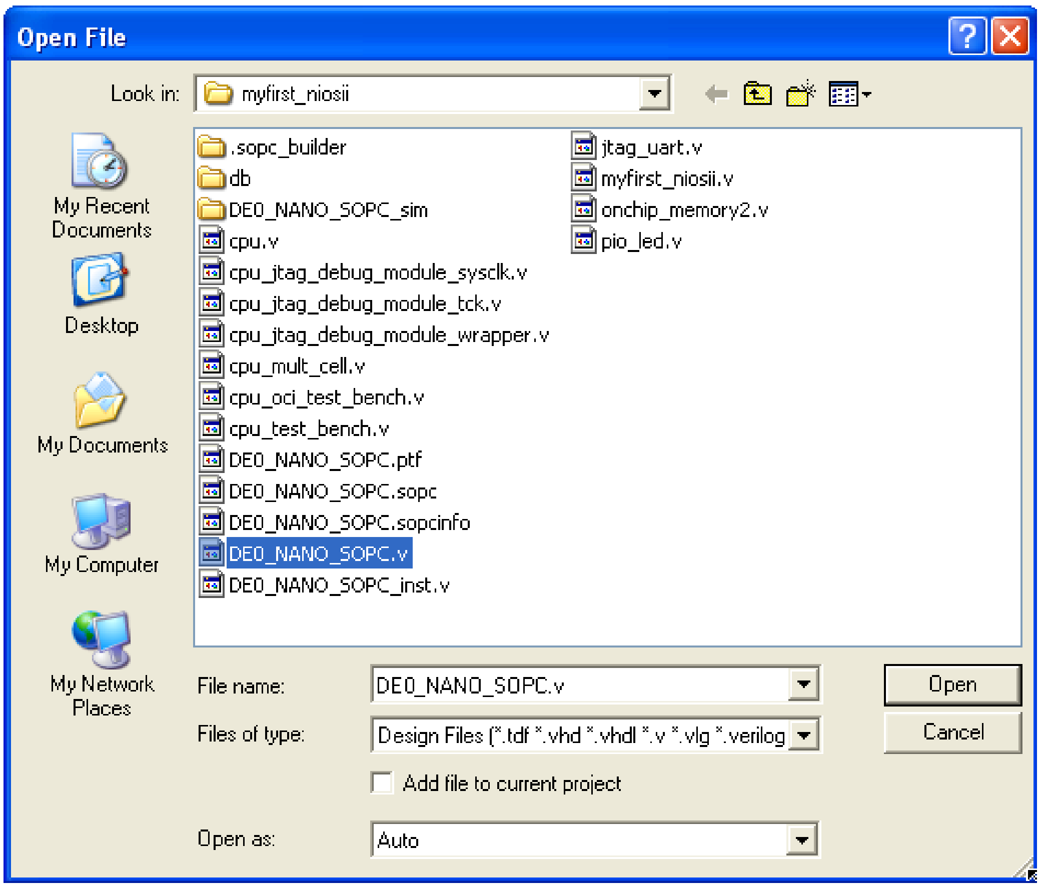
Figure 7-44 and **Figure 7-45**.

```
module myfirst_niosii
(
        CLOCK_50,
        LED
);
input                   CLOCK_50;
output [7:0]      LED;
DE0_NANO_SOPC    DE0_NANO_SOPC_inst
        (
            .clk_50                                  (CLOCK_50),
            .out_port_from_the_pio_led    (LED),
            .reset_n                                 (1'b1)
        );

endmodule
```


Quartus II - D:/myfirst_niosii/myfirst_niosii - myfirst_niosii
File Edit View Project Assignments Processing Tools Window Help
myfirst_niosii
Project Navigator
Entity
Cyclone IV E: EP4CE22F17C6
myfirst_niosii
Hierarchy
Files
Design Units
myfirst_niosii.v
0%
00:00:00

```
//
// Major Functions:  myfirst_niosii
//
// --------------------------------------------------------------------
//
// Revision History :
// --------------------------------------------------------------------
//   Ver  :| Author              :| Mod. Date    :| Changes Made:
//   V1.0 :| Yaqun-chang         :| 02/16/2011   :| Initial Revis:
// --------------------------------------------------------------------
module myfirst_niosii
(
    CLOCK_50,
    LED
);
input           CLOCK_50;
output  [7:0]   LED;
DE0_NANO_SOPC   DE0_NANO_SOPC_inst
    (
      .clk_50                        (CLOCK_50),
      .out_port_from_the_pio_led     (LED),
      .reset_n                       (1'b1)
    );

endmodule
```

Figure 7-43 Input verilog Text



.sopc_builder
db
DE0_NANO_SOPC_sim
cpu.v
cpu_jtag_debug_module_sysclk.v
cpu_jtag_debug_module_tck.v
cpu_jtag_debug_module_wrapper.v
cpu_mult_cell.v
cpu_oci_test_bench.v
cpu_test_bench.v
DE0_NANO_SOPC.ptf
DE0_NANO_SOPC.sopc
DE0_NANO_SOPC.sopcinfo
DE0_NANO_SOPC.v
DE0_NANO_SOPC_inst.v
jtag_uart.v
myfirst_niosii.v
onchip_memory2.v
pio_led.v

File name: DE0_NANO_SOPC.v

Files of type: Design Files (*.tdf *.vhd *.vhdl *.v *.vlg *.verilog

☐ Add file to current project

Open as: Auto

Figure 7-44 Open DE0_NANO_SOPC.v

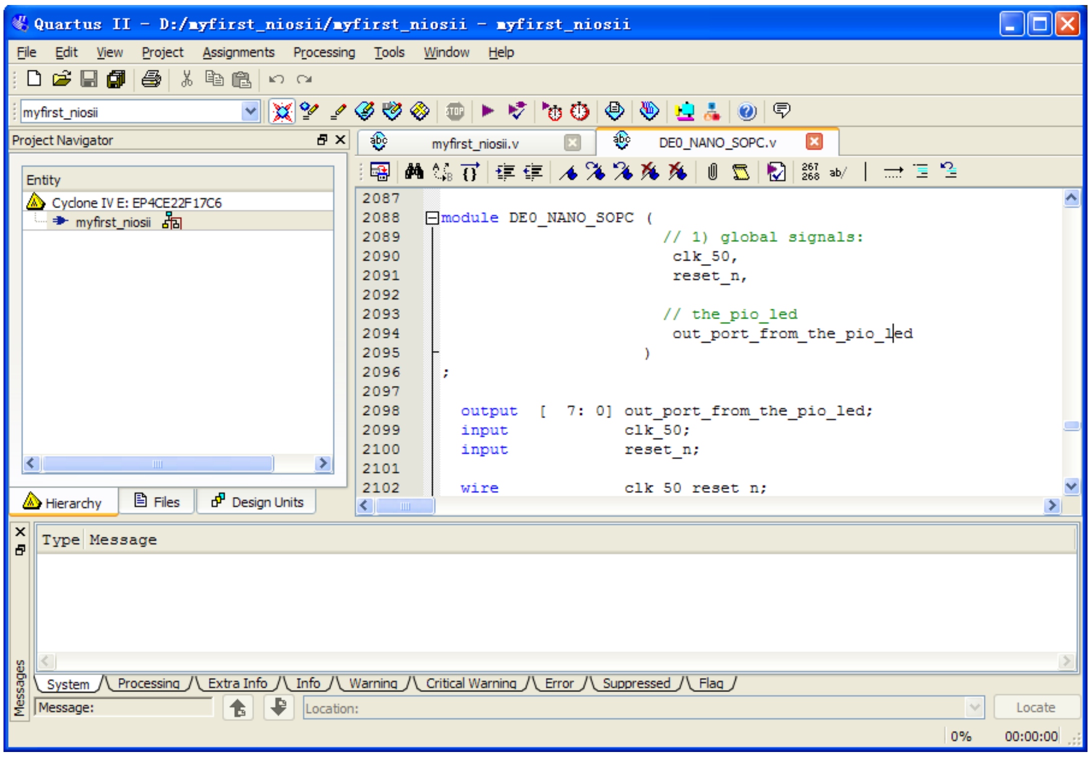
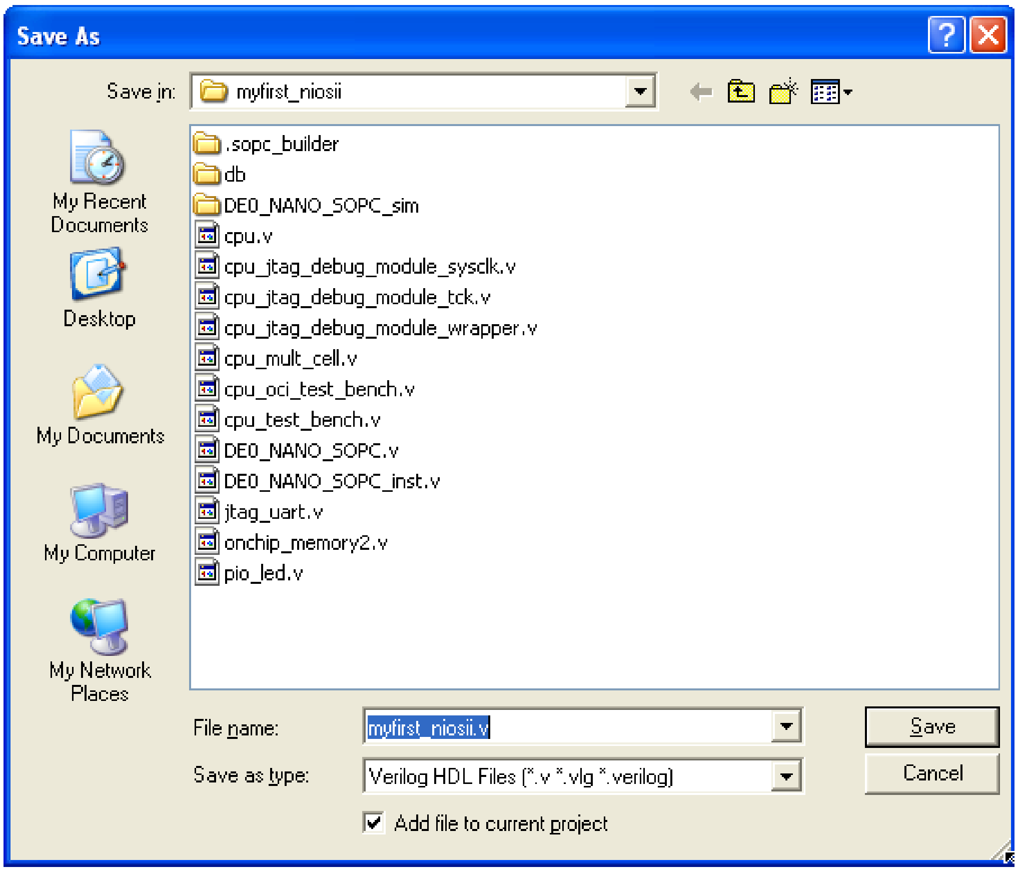

```
module DE0_NANO_SOPC (
                      // 1) global signals:
                       clk_50,
                       reset_n,

                      // the_pio_led
                       out_port_from_the_pio_led
                    )
;

  output  [  7: 0] out_port_from_the_pio_led;
  input            clk_50;
  input            reset_n;

  wire             clk_50_reset_n;
```

Figure 7-45 DE0_NANO_SOPC module

35. Choose **Save** Icon in the tool bar. There will appear a window as shown in Figure 7-46. Click **Save.**


Save As

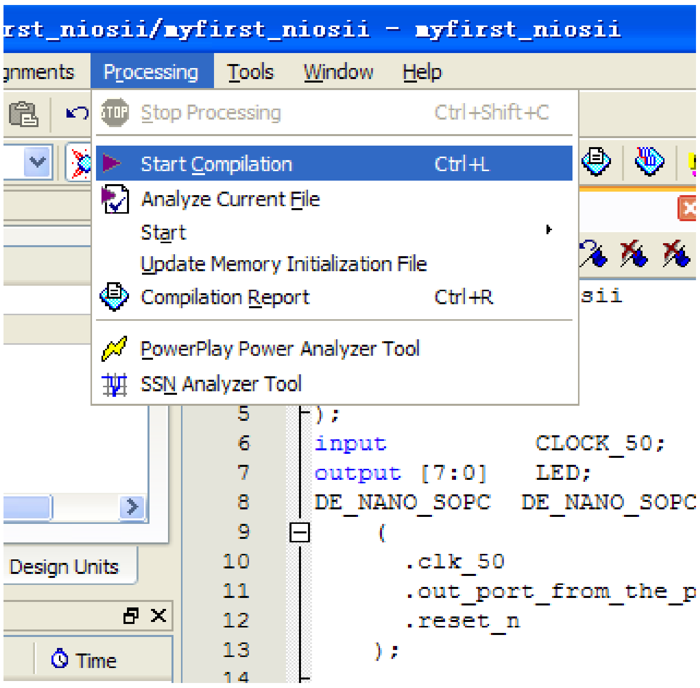
Save in: myfirst_niosii

.sopc_builder
db
DE0_NANO_SOPC_sim
cpu.v
cpu_jtag_debug_module_sysclk.v
cpu_jtag_debug_module_tck.v
cpu_jtag_debug_module_wrapper.v
cpu_mult_cell.v
cpu_oci_test_bench.v
cpu_test_bench.v
DE0_NANO_SOPC.v
DE0_NANO_SOPC_inst.v
jtag_uart.v
onchip_memory2.v
pio_led.v

File name: myfirst_niosii.v
Save as type: Verilog HDL Files (*.v *.vlg *.verilog)
☑ Add file to current project

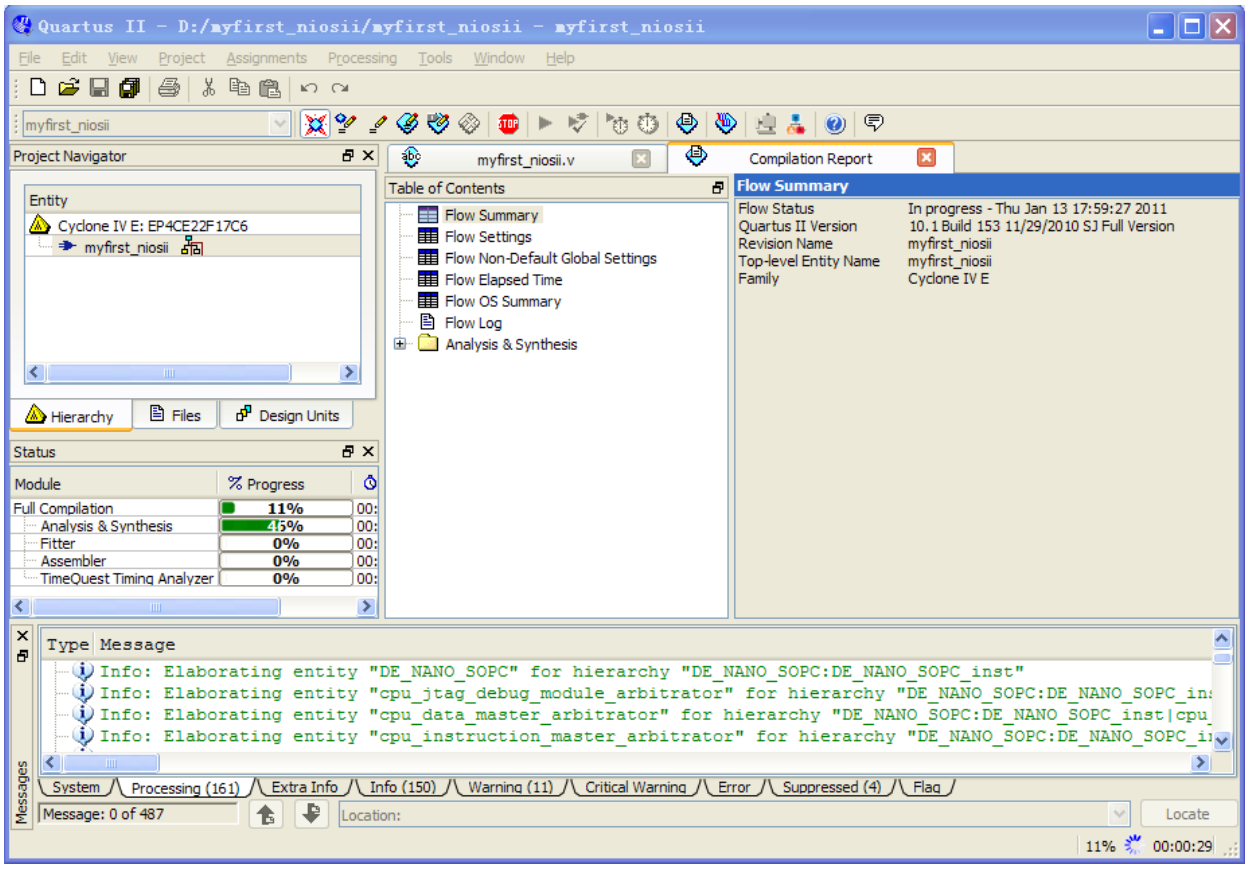
Figure 7-46 Save Verilog file

36. Choose **Processing** > **Start Compilation** as shown in Figure 7-47. Figure 7-48 shows the compilation process.


rst_niosii/myfirst_niosii - myfirst_niosii
gnments
Processing
Tools
Window
Help
Stop Processing Ctrl+Shift+C
Start Compilation Ctrl+L
Analyze Current File
Start
Update Memory Initialization File
Compilation Report Ctrl+R
PowerPlay Power Analyzer Tool
SSN Analyzer Tool
Design Units
Time

```
);
input           CLOCK_50;
output [7:0]    LED;
DE_NANO_SOPC   DE_NANO_SOPC
    (
        .clk_50
        .out_port_from_the_p
        .reset_n
    );
```

Figure 7-47 Start Compilation



Flow Summary

Flow Status	In progress - Thu Jan 13 17:59:27 2011
Quartus II Version	10.1 Build 153 11/29/2010 SJ Full Version
Revision Name	myfirst_niosii
Top-level Entity Name	myfirst_niosii
Family	Cyclone IV E

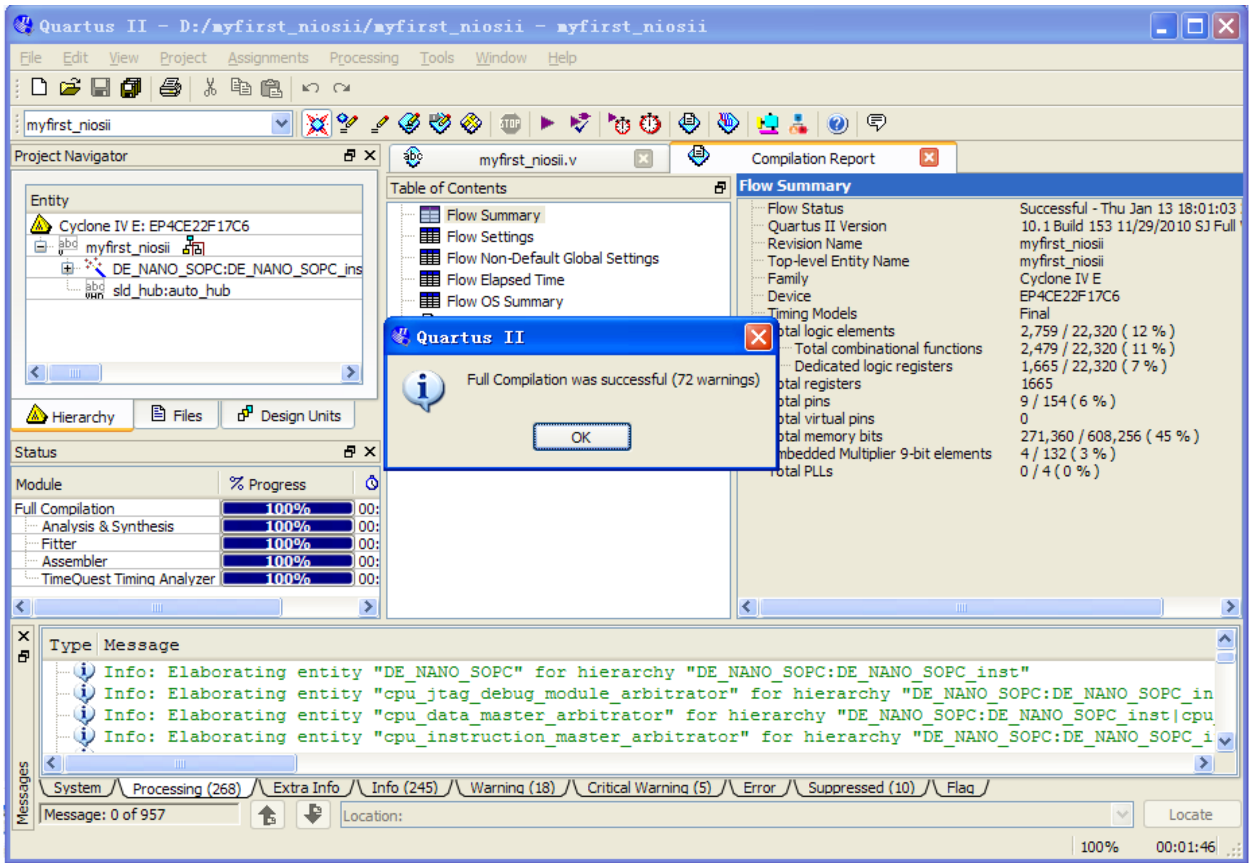
Status

Module	Progress	
Full Compilation	11%	00:
Analysis & Synthesis	45%	00:
Fitter	0%	00:
Assembler	0%	00:
TimeQuest Timing Analyzer	0%	00:

```
Type Message
Info: Elaborating entity "DE_NANO_SOPC" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_inst"
Info: Elaborating entity "cpu_jtag_debug_module_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_in
Info: Elaborating entity "cpu_data_master_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_inst|cpu
Info: Elaborating entity "cpu_instruction_master_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_in
```

Figure 7-48 Execute Compile

37. A window that shows successfully will appear as shown in **Figure 7-49**.


Quartus II - D:/myfirst_niosii/myfirst_niosii - myfirst_niosii

Quartus II

Full Compilation was successful (72 warnings)

OK

Flow Summary

Flow Status	Successful - Thu Jan 13 18:01:03
Quartus II Version	10.1 Build 153 11/29/2010 SJ Full
Revision Name	myfirst_niosii
Top-level Entity Name	myfirst_niosii
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
otal logic elements	2,759 / 22,320 (12 %)
Total combinational functions	2,479 / 22,320 (11 %)
Dedicated logic registers	1,665 / 22,320 (7 %)
otal registers	1665
otal pins	9 / 154 (6 %)
otal virtual pins	0
otal memory bits	271,360 / 608,256 (45 %)
mbedded Multiplier 9-bit elements	4 / 132 (3 %)
otal PLLs	0 / 4 (0 %)

Module	Progress
Full Compilation	100%
Analysis & Synthesis	100%
Fitter	100%
Assembler	100%
TimeQuest Timing Analyzer	100%

```
Type Message
Info: Elaborating entity "DE_NANO_SOPC" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_inst"
Info: Elaborating entity "cpu_jtag_debug_module_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_in
Info: Elaborating entity "cpu_data_master_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_inst|cpu
Info: Elaborating entity "cpu_instruction_master_arbitrator" for hierarchy "DE_NANO_SOPC:DE_NANO_SOPC_i
```

Message: 0 of 957

Figure 7-49 Compile project completely

45. Choose **Assignments** > **Pins** to open pin planner as shown in **Figure 7-50**. **Figure 7-51** show blank pins.

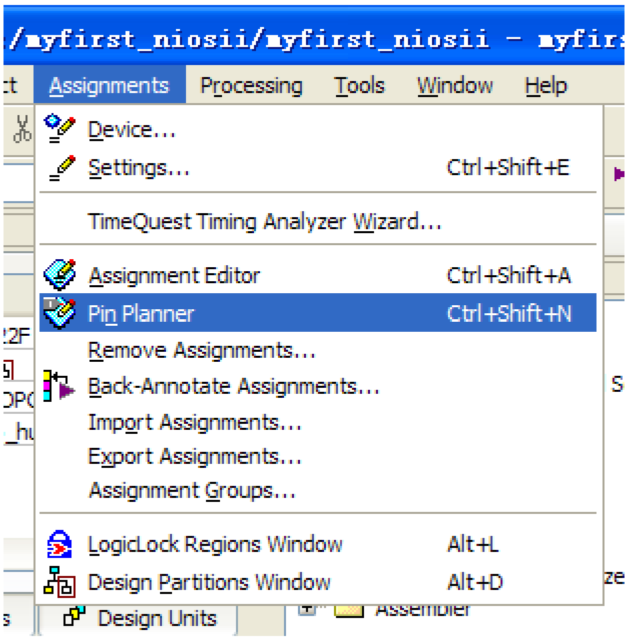
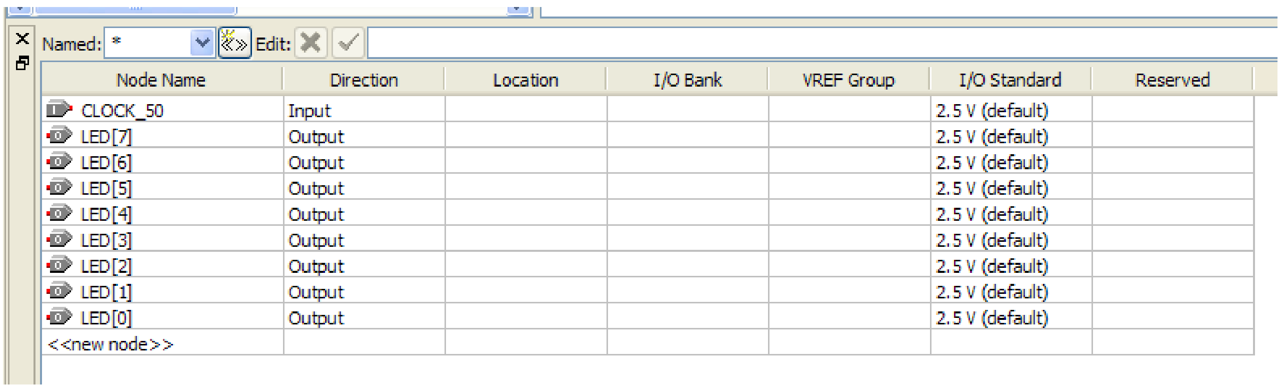

/myfirst_niosii/myfirst_niosii - myfir
Assignments
Processing
Tools
Window
Help
Device...
Settings... Ctrl+Shift+E
TimeQuest Timing Analyzer Wizard...
Assignment Editor Ctrl+Shift+A
Pin Planner Ctrl+Shift+N
Remove Assignments...
Back-Annotate Assignments...
Import Assignments...
Export Assignments...
Assignment Groups...
LogicLock Regions Window Alt+L
Design Partitions Window Alt+D
Design Units

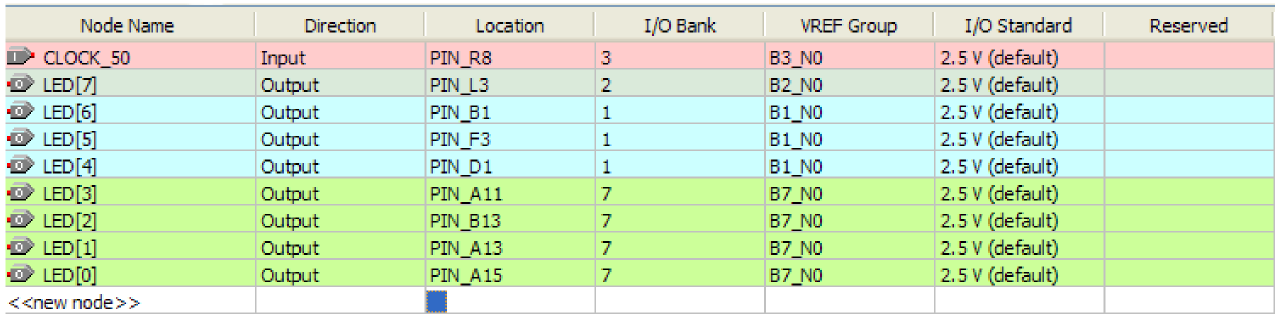
Figure 7-50 Pins menu



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
CLOCK_50	Input				2.5 V (default)	
LED[7]	Output				2.5 V (default)	
LED[6]	Output				2.5 V (default)	
LED[5]	Output				2.5 V (default)	
LED[4]	Output				2.5 V (default)	
LED[3]	Output				2.5 V (default)	
LED[2]	Output				2.5 V (default)	
LED[1]	Output				2.5 V (default)	
LED[0]	Output				2.5 V (default)	
<<new node>>						

Figure 7-51 Blank Pins

46. Input Location value as shown in **Figure 7-52**.



Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
CLOCK_50	Input	PIN_R8	3	B3_N0	2.5 V (default)	
LED[7]	Output	PIN_L3	2	B2_N0	2.5 V (default)	
LED[6]	Output	PIN_B1	1	B1_N0	2.5 V (default)	
LED[5]	Output	PIN_F3	1	B1_N0	2.5 V (default)	
LED[4]	Output	PIN_D1	1	B1_N0	2.5 V (default)	
LED[3]	Output	PIN_A11	7	B7_N0	2.5 V (default)	
LED[2]	Output	PIN_B13	7	B7_N0	2.5 V (default)	
LED[1]	Output	PIN_A13	7	B7_N0	2.5 V (default)	
LED[0]	Output	PIN_A15	7	B7_N0	2.5 V (default)	
<<new node>>						

Figure 7-52 Set Pins

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

7.3 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.

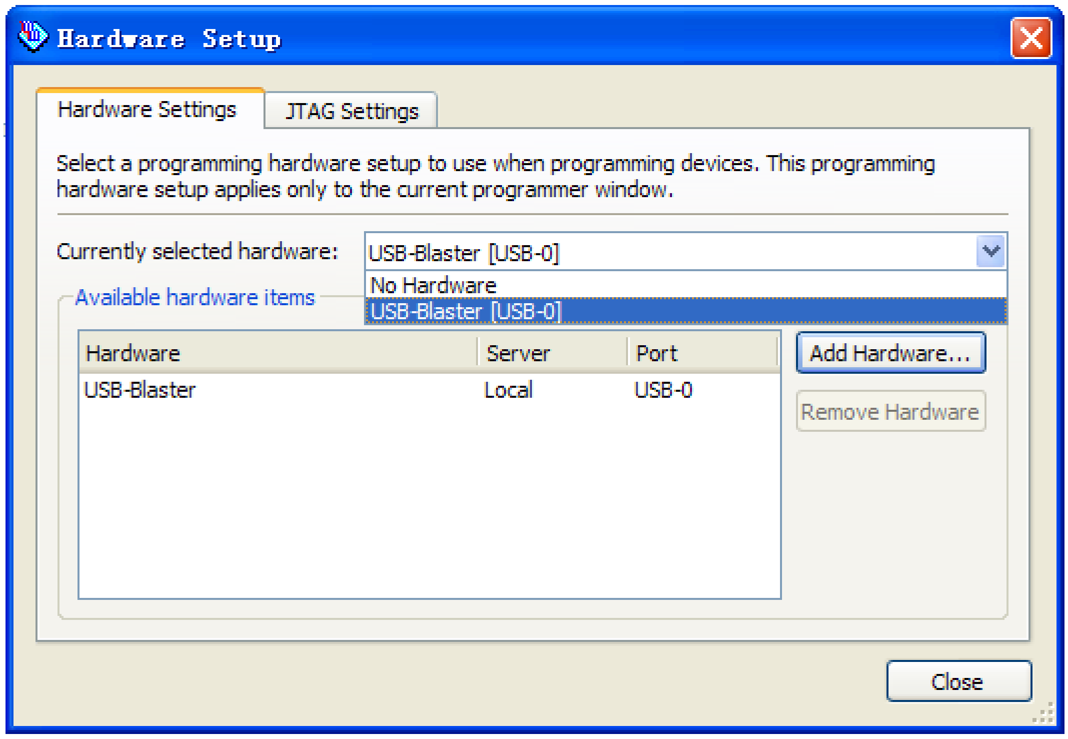
10. Select the programming file (myfirst_niosii.sof) for your board.

11. Click **OK**.

12. Click **Hardware Setup** in the top, left comer 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 7-53.



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

No Hardware
USB-Blaster [USB-0]

Available hardware items

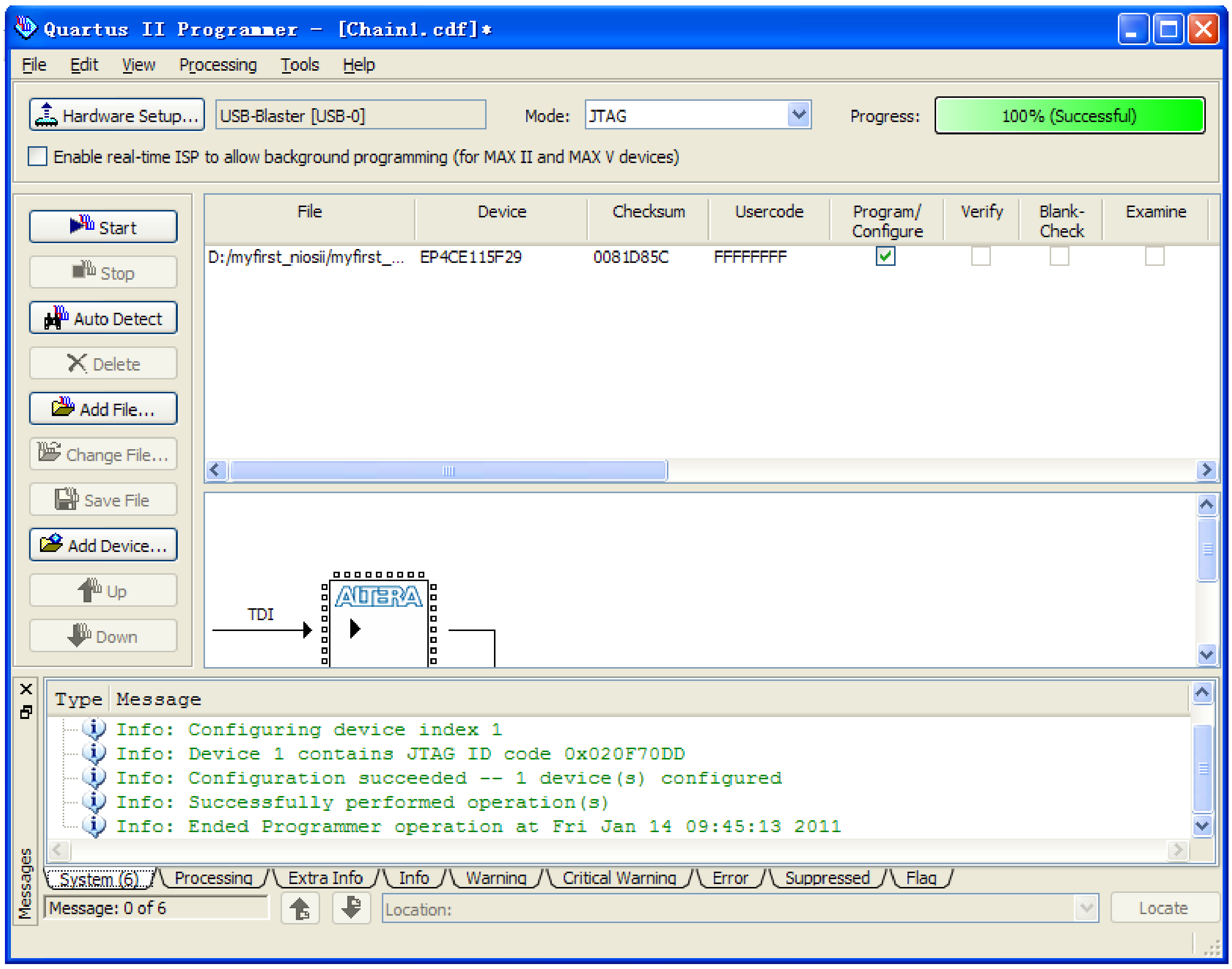
Hardware	Server	Port
USB-Blaster	Local	USB-0

Figure 7-53 HardWare Setup Window

14. Click **Close**.

15. Turn on the **Program/Configure** option for the programming file (see Figure 7-54 for an example).

16. Click **Start**.



USB-Blaster [USB-0]   Mode: JTAG   Progress: 100% (Successful)

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
D:/myfirst_niosii/myfirst_...	EP4CE115F29	0081D85C	FFFFFFFF	✓	☐	☐	☐

```
Type  Message
      Info: Configuring device index 1
      Info: Device 1 contains JTAG ID code 0x020F70DD
      Info: Configuration succeeded -- 1 device(s) configured
      Info: Successfully performed operation(s)
      Info: Ended Programmer operation at Fri Jan 14 09:45:13 2011
```

System (6) / Processing / Extra Info / Info / Warning / Critical Warning / Error / Suppressed / Flag

Message: 0 of 6   Location:

Figure 7-54 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.

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.

7.4 Create a 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.

Noet: 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 7-55 and Figure 7-56.

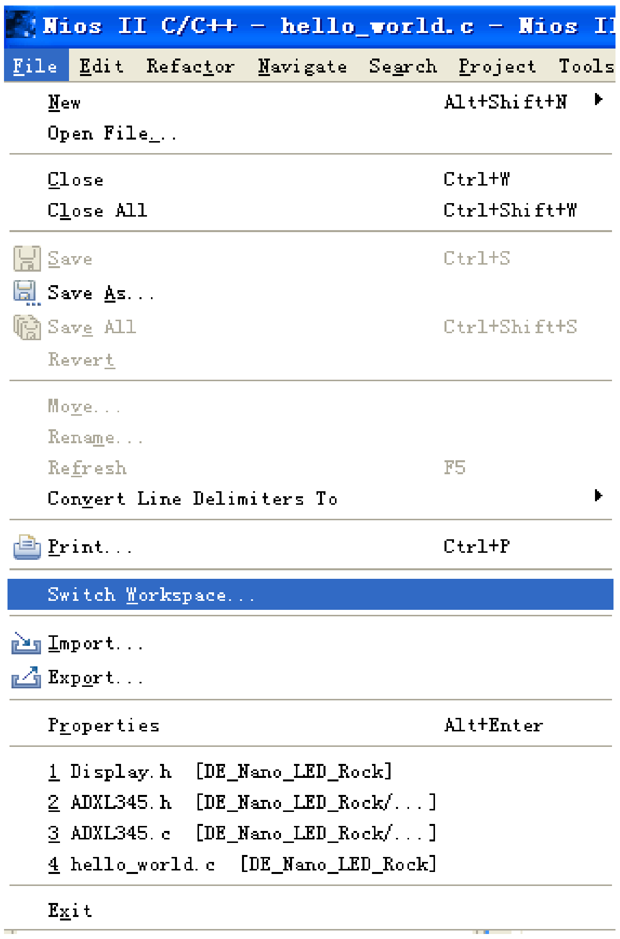
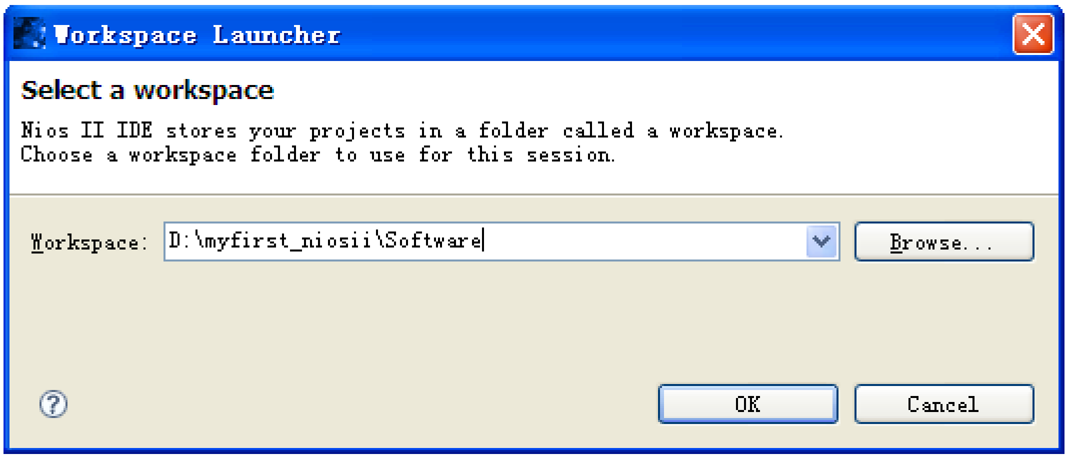

Nios II C/C++ - hello_world.c - Nios II
File Edit Refactor Navigate Search Project Tools
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename...
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace...
Import...
Export...
Properties Alt+Enter
1 Display.h [DE_Nano_LED_Rock]
2 ADXL345.h [DE_Nano_LED_Rock/...]
3 ADXL345.c [DE_Nano_LED_Rock/...]
4 hello_world.c [DE_Nano_LED_Rock]
Exit

Figure 7-55 Swtich Workspace (1)



Select a workspace

Nios II IDE stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\myfirst_niosii\Software

Figure 7-56 Switch Workspace (2)

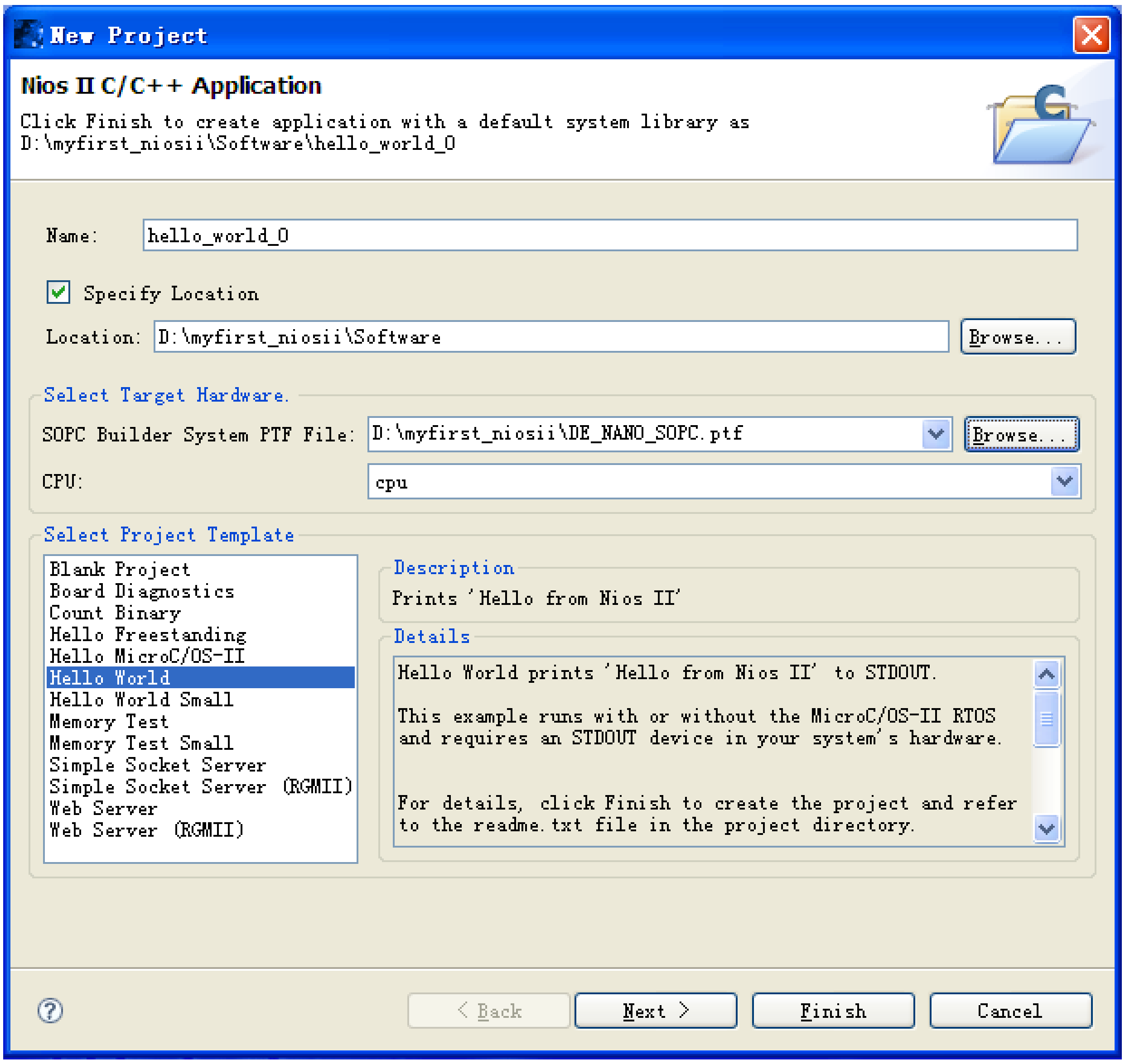
3. Choose **File->New->NIOS II C/C++ Application** to open the New Project Wizard.

4. In the New Project wizard, make sure the following things:

a.Select the **Hello World** project template.

b.Give the project a name. (**hello_world_0** is default name)

c.Select the target hardware system PTF file that locates in where the previously created hardware project resides as shown in Figure 7-57.



Nios II C/C++ Application

Click Finish to create application with a default system library as D:\myfirst_niosii\Software\hello_world_0

Name: hello_world_0

[✓] Specify Location

Location: D:\myfirst_niosii\Software

Select Target Hardware.

SOPC Builder System PTF File: D:\myfirst_niosii\DE_NANO_SOPC.ptf

CPU: cpu

Select Project Template

Blank Project
Board Diagnostics
Count Binary
Hello Freestanding
Hello MicroC/OS-II
Hello World
Hello World Small
Memory Test
Memory Test Small
Simple Socket Server
Simple Socket Server (RGMII)
Web Server
Web Server (RGMII)

Description

Prints 'Hello from Nios II'

Details

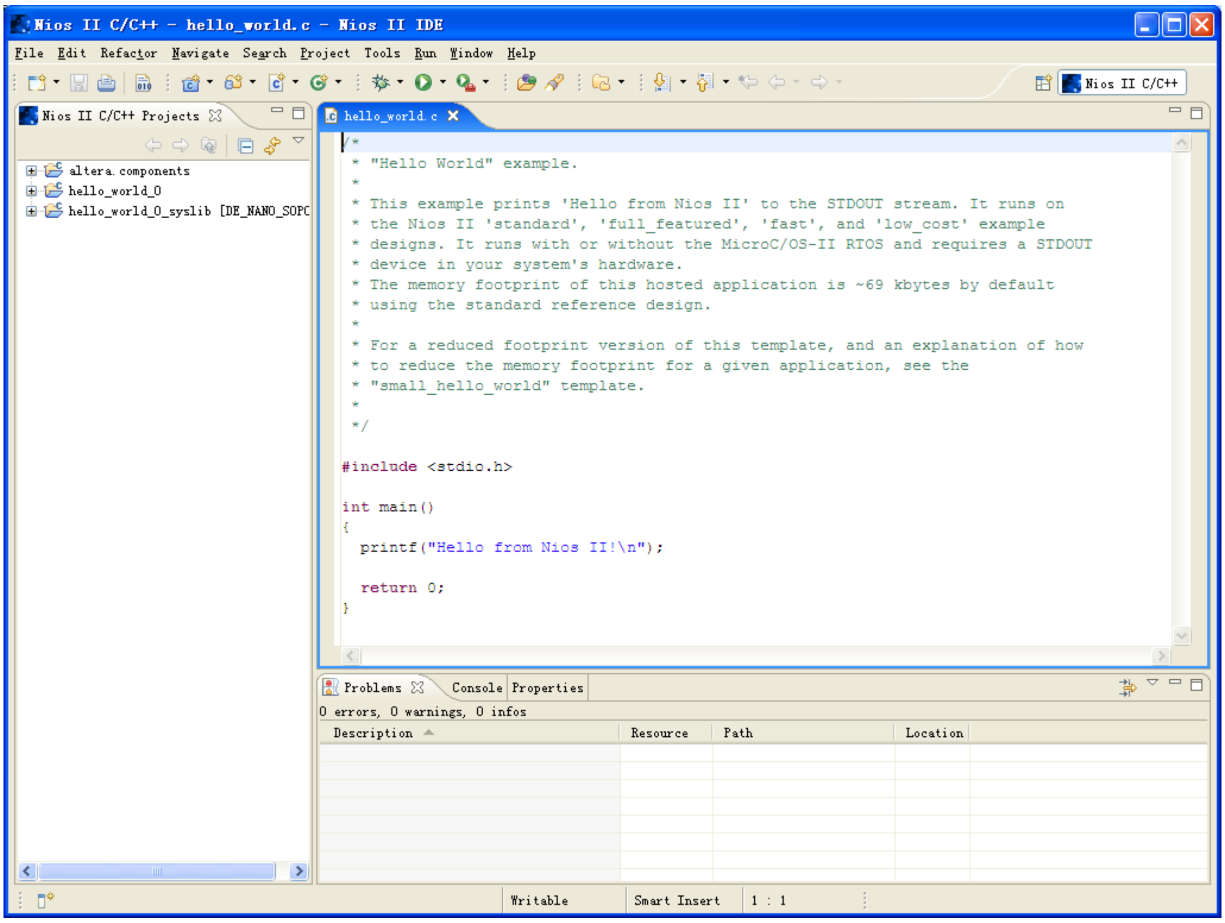
Hello World prints 'Hello from Nios II' to STDOUT.

This example runs with or without the MicroC/OS-II RTOS and requires an STDOUT device in your system's hardware.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

Figure 7-57 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 7-58.


Nios II C/C++ - hello_world.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++
Nios II C/C++ Projects
altera.components
hello_world_0
hello_world_0_syslib [DE_NANO_SOPC
hello_world.c
Problems
Console
Properties
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Location
Writable
Smart Insert
1 : 1

```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#include <stdio.h>

int main()
{
  printf("Hello from Nios II!\n");

  return 0;
}
```

Figure 7-58 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 realtime operating system (RTOS).

● **system.h**, which is a header file that encapsulates your hardware system.

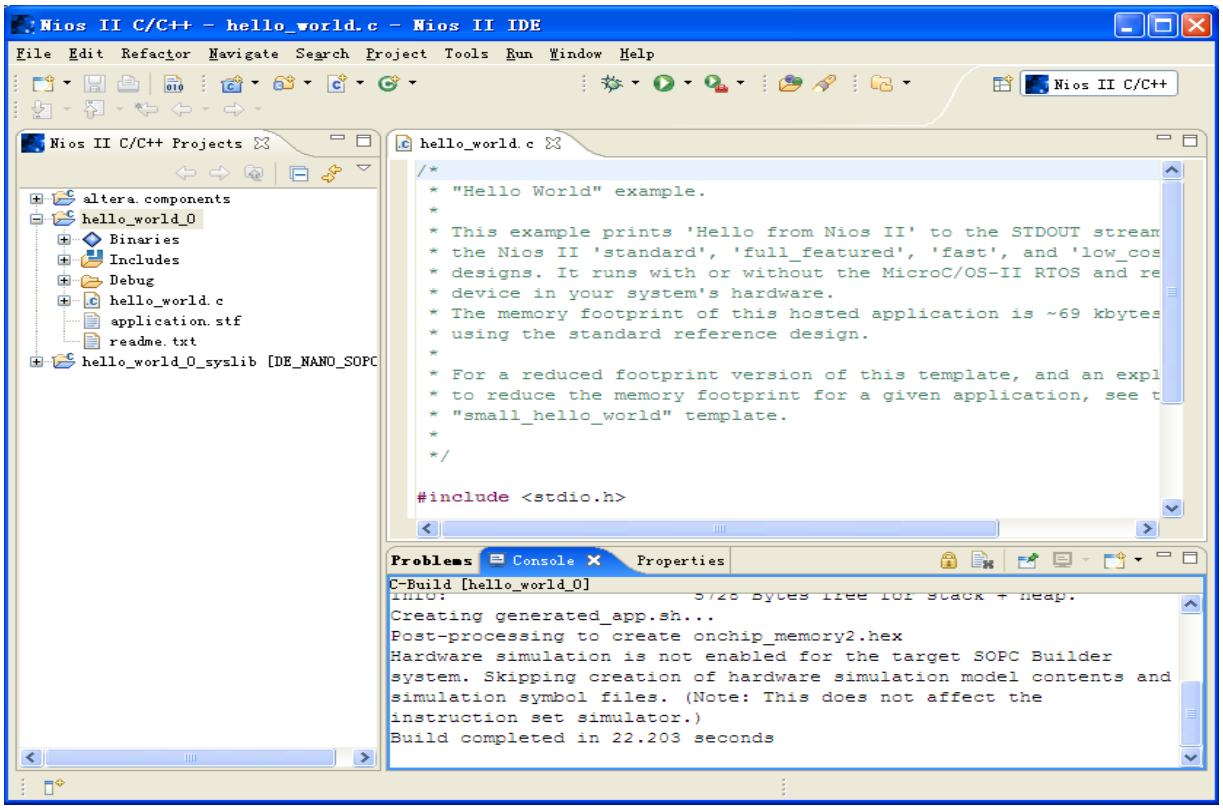
● **alt_sys_init.c,** which is an initialization file that initializes the devices in the system.

● **Hello_world_0.elf**, which is an executable and linked format file for the application located in hello_world_0 folder under Debug.

7.5 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 7-59** for an example.



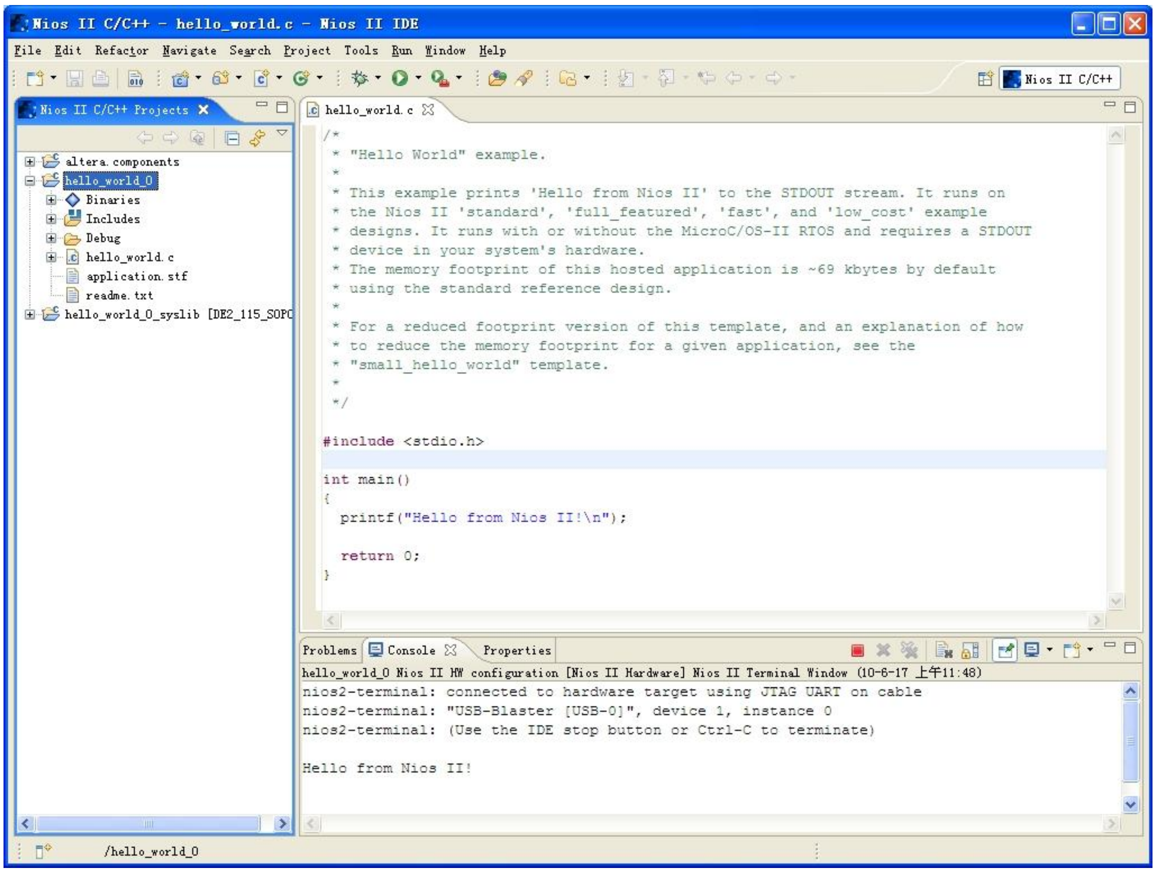
```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cos
 * designs. It runs with or without the MicroC/OS-II RTOS and re
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an expl
 * to reduce the memory footprint for a given application, see t
 * "small_hello_world" template.
 *
 */

#include <stdio.h>
```

```
C-Build [hello_world_0]
Creating generated_app.sh...
Post-processing to create onchip_memory2.hex
Hardware simulation is not enabled for the target SOPC Builder
system. Skipping creation of hardware simulation model contents and
simulation symbol files. (Note: This does not affect the
instruction set simulator.)
Build completed in 22.203 seconds
```

Figure 7-59 Nios II IDE hello_world_0 Build Completed

*Note: If there appears the error which is "**region onchip_memory2 is full(hello_world_0.elf section .text). Region needs to be XXX bytes larger.**" , please right-click **hello_world_0** ,choose **System Library Properties** menu, then pop a window. In the **System Library Properties** window, choose **Small C Library**, then click **OK** to close the window. Rebuild the project.*

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 7-60 for an example.



```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#include <stdio.h>

int main()
{
  printf("Hello from Nios II!\n");

  return 0;
}
```

```
hello_world_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (10-6-17 上午11:48)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello from Nios II!
```

Figure 7-60 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.

7.6 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 LED 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.

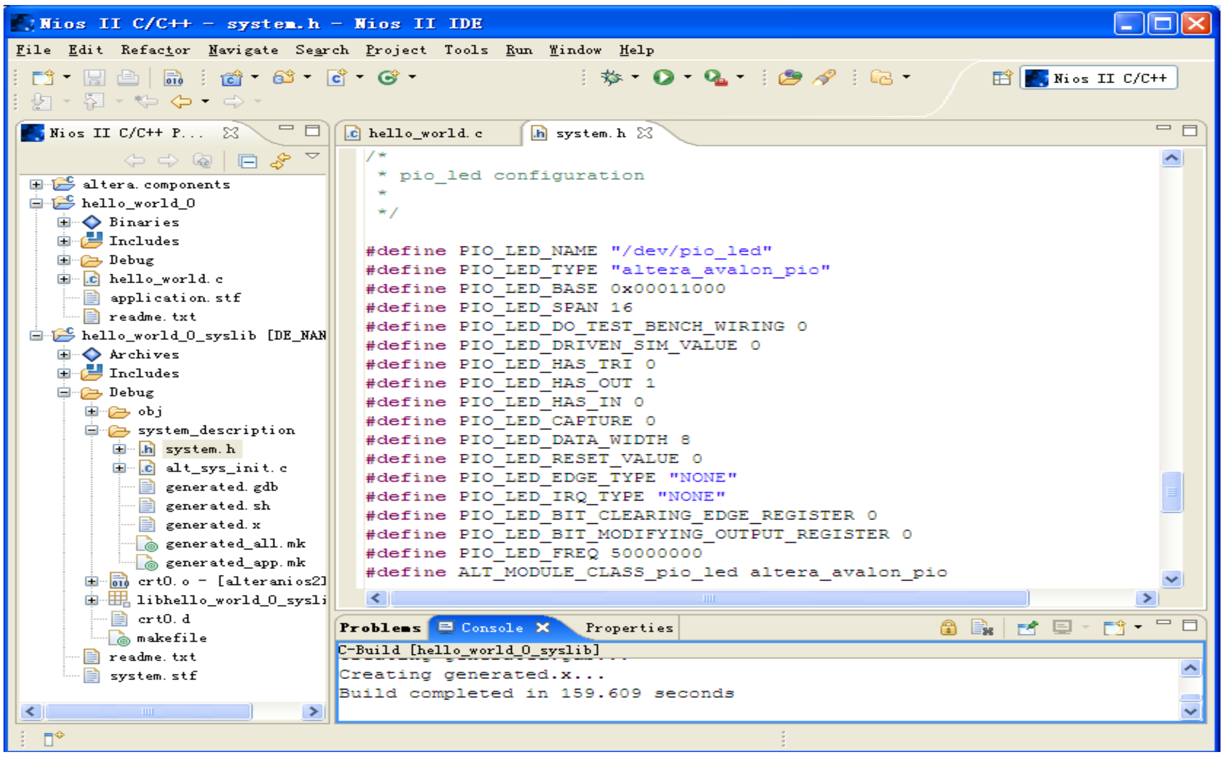
3. Recompile the file by right-clicking **hello_world_0** in the NIOS II C/C++ Projects tab and choosing **Run** > **Run As** > **Nios II Hardware**.

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.

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

7.7 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 7-61.



```
/*
 * pio_led configuration
 *
 */

#define PIO_LED_NAME "/dev/pio_led"
#define PIO_LED_TYPE "altera_avalon_pio"
#define PIO_LED_BASE 0x00011000
#define PIO_LED_SPAN 16
#define PIO_LED_DO_TEST_BENCH_WIRING 0
#define PIO_LED_DRIVEN_SIM_VALUE 0
#define PIO_LED_HAS_TRI 0
#define PIO_LED_HAS_OUT 1
#define PIO_LED_HAS_IN 0
#define PIO_LED_CAPTURE 0
#define PIO_LED_DATA_WIDTH 8
#define PIO_LED_RESET_VALUE 0
#define PIO_LED_EDGE_TYPE "NONE"
#define PIO_LED_IRQ_TYPE "NONE"
#define PIO_LED_BIT_CLEARING_EDGE_REGISTER 0
#define PIO_LED_BIT_MODIFYING_OUTPUT_REGISTER 0
#define PIO_LED_FREQ 50000000
#define ALT_MODULE_CLASS_pio_led altera_avalon_pio
```

```
C-Build [hello_world_0_syslib]
Creating generated.x...
Build completed in 159.609 seconds
```

Figure 7-61 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, interruptmask, and edgecapture**. 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)

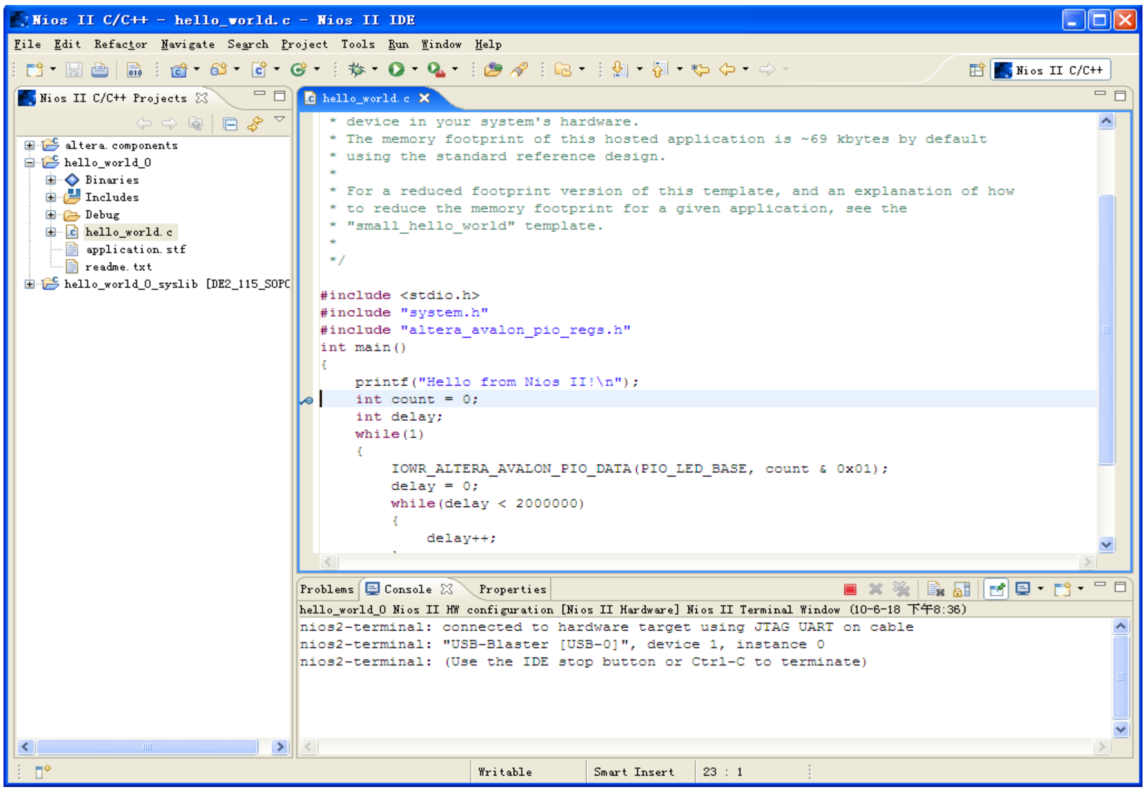
can write to the PIO data register, turning the LED on and off. The PIO is just one of many SOPC peripherals that you can use in a system. To learn about the PIO core and other embedded peripheral cores, refer to Quartus II Version <version> Handbook Volume 5: Embedded Peripherals.

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.

7.8 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 7-62**.



```
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#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++;
```

```
hello_world_0 Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (10-6-18 下午8:36)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)
```

Figure 7-62 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.

7.9 Configure System Library

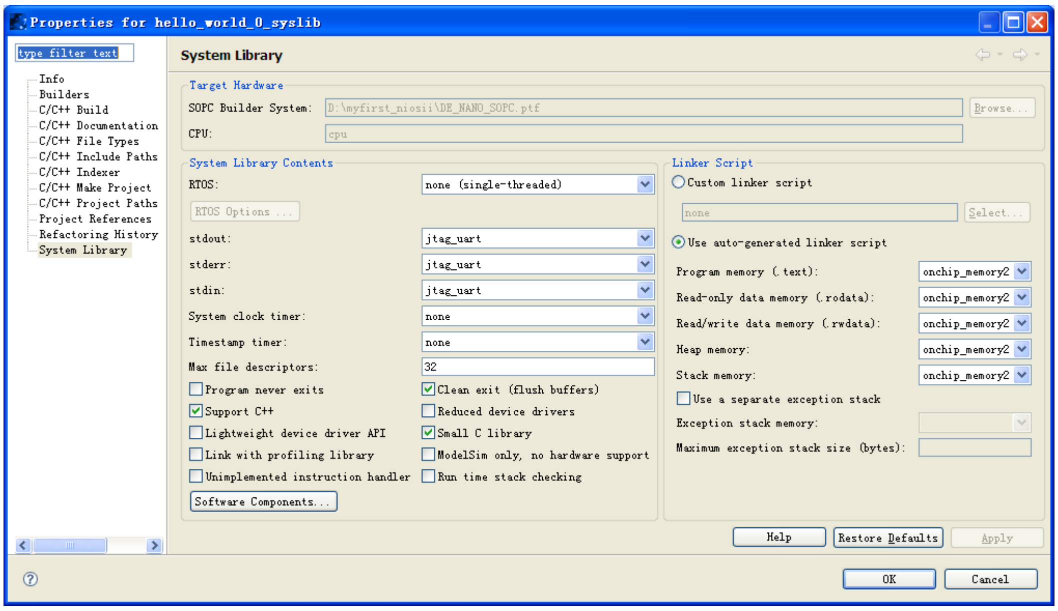
In this section you will learn how to configure some advanced options about the target memory or other things. By performing the following steps, you can charge all the available settings:

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(.rwdata)**, **Heap memory, and Stack memory**, see Figure 7-63. 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 7-63** for an example.


Properties for hello_world_0_syslib

type filter text
- Info
- Builders
- C/C++ Build
- C/C++ Documentation
- C/C++ File Types
- C/C++ Include Paths
- C/C++ Indexer
- C/C++ Make Project
- C/C++ Project Paths
- Project References
- Refactoring History
- System Library

System Library

Target Hardware
SOPC Builder System: D:\myfirst_niosii\DE_NANO_SOPC.ptf   Browse...
CPU: cpu

System Library Contents
RTOS: none (single-threaded)
RTOS Options ...
stdout: jtag_uart
stderr: jtag_uart
stdin: jtag_uart
System clock timer: none
Timestamp timer: none
Max file descriptors: 32
- [ ] Program never exits
- [x] Support C++
- [ ] Lightweight device driver API
- [ ] Link with profiling library
- [ ] Unimplemented instruction handler
- [x] Clean exit (flush buffers)
- [ ] Reduced device drivers
- [x] Small C library
- [ ] ModelSim only, no hardware support
- [ ] Run time stack checking

Software Components...

Linker Script
○ Custom linker script
none   Select...
● Use auto-generated linker script
Program memory (.text): onchip_memory2
Read-only data memory (.rodata): onchip_memory2
Read/write data memory (.rwdata): onchip_memory2
Heap memory: onchip_memory2
Stack memory: onchip_memory2
- [ ] Use a separate exception stack
Exception stack memory:
Maximum exception stack size (bytes):

Help   Restore Defaults   Apply

OK   Cancel

Figure 7-63 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 8

DE0-Nano Demonstrations

8.1 System Requirements

Make sure Quartus II and NIOS II are installed on your PC.

8.2 Breathing LEDs

In this demonstration, it will show how to use the FPGA to control the luminance of the LEDs by means of PWM scheme .There are two groups of LEDs, while one group dims, the other group lights up and vice versa. This behavior acts like a 'human breath'. Users can change the PWM wave's duty ratio and frequency to control the LED luminance and repetition rate.

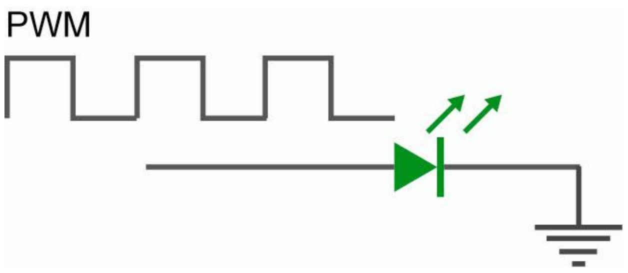


Figure 8-1 Shows a diagram of PWM signals to drive LED.

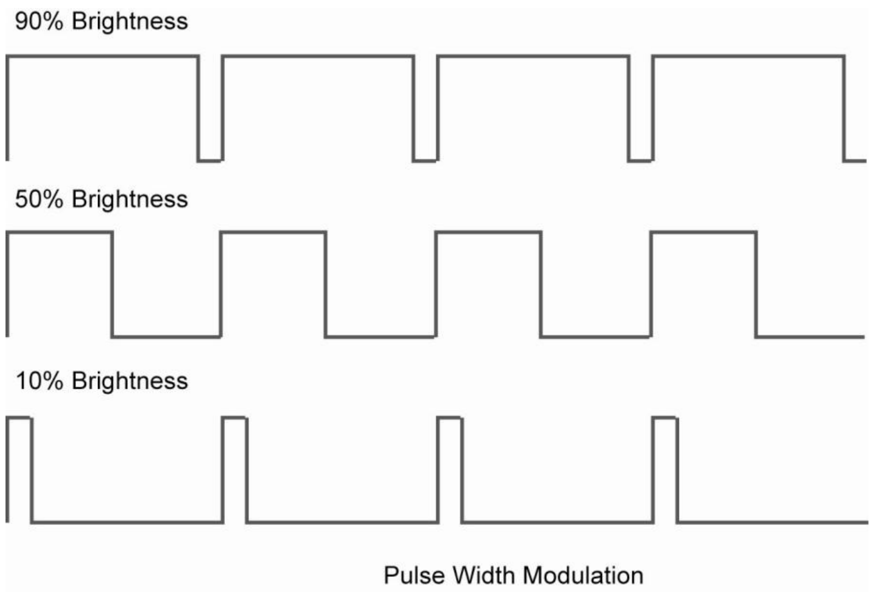


Figure 8-2 Pulse Width Modulation

Figure 8-2 shows the relationship between duty cycle and LED luminance.

Demonstration Source Code

- Project directory: DE0_NANO_Default
- Bit stream used: DE0_NANO.sof

Demonstration Batch File

Demo Batch File Folder: DE0_NANO_Default\demo_batch

The demo batch file includes the following files:

- FPGA Configure File: DE0_NANO.sof

Demonstration Setup

- Make sure Quartus II and Nios II are installed on your PC.
- Connect USB cable to the DE0-Nano board and install USB Blaster driver if necessary.
- Execute the demo batch file "DE0_NANO.bat" under the batch file folder, *DE0_NANO_ Default \demo_batch.*

8.3 ADC Reading

This section illustrates the steps to evaluate the performance of the 8-channel 12-bit A/D Converter. A DC 3.3V on the 2x13 header is used to drive the analog signal and by using a trimmer potentiometer, we are able to adjust voltage range from 0~3.3V. The 12-bit voltage measurements are indicated on the 8 LEDs. Since there are only 8 LEDs, only bit-4 through bit-11 are represented on the LEDs.

■ Design Concept

This section describes the design concepts for this demo. **Figure 8-3** shows the block diagram.

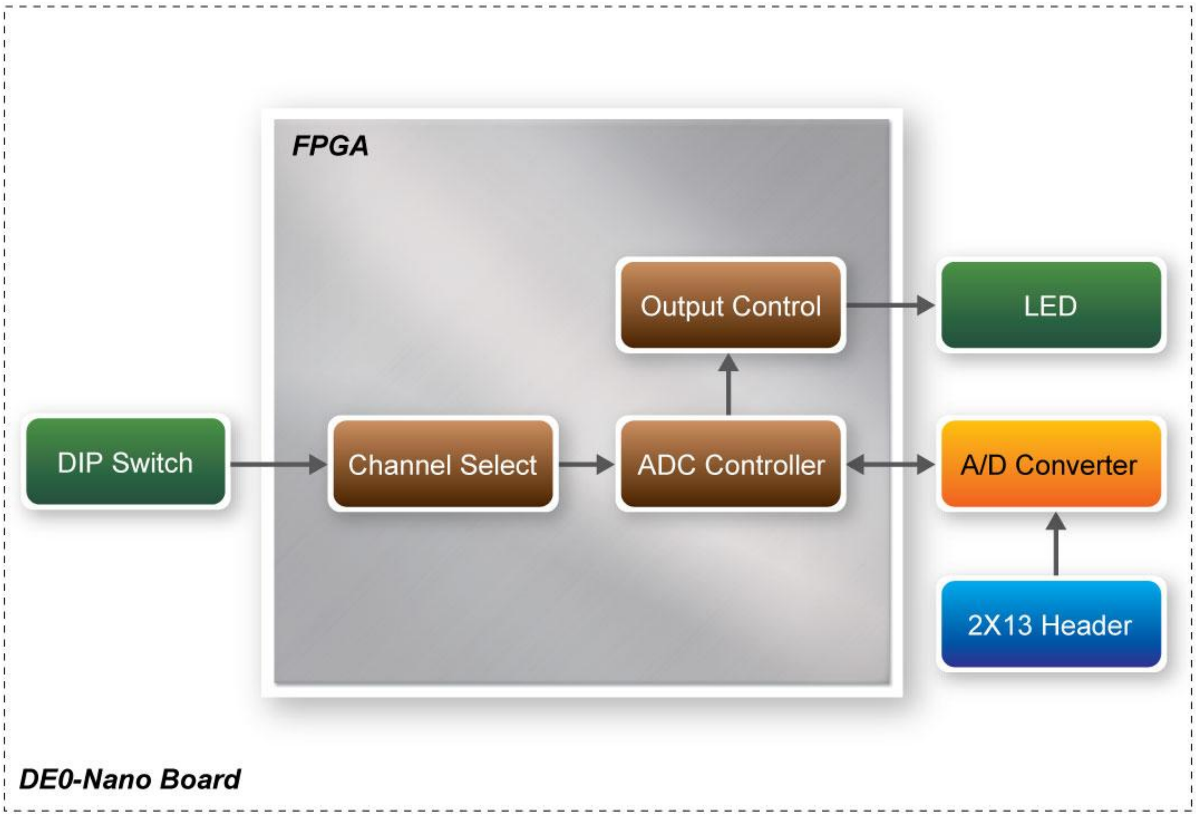


Figure 8-3 ADC Reading Block Diagram

The ADC Controller reads the voltage from the A/D converter through the serial interface and displays its measurement on the LEDs. The on-board dip-switch determines which channel to read from. **Table 8–1** lists the DIP Switch settings and its corresponding ADC channel.

Table 8–1 DIP Switch Settings

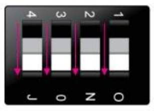
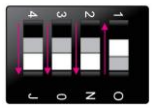
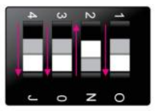
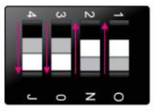
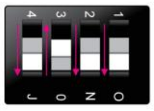
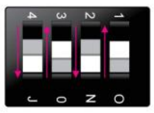
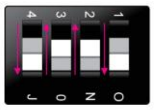
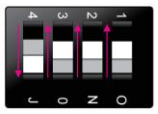
DIP Switch (SW1)	*Setting*	*ADC Channel*
	**0000**	**Analog_In0**
	**0001**	**Analog_In1**
	**0010**	**Analog_In2**
	**0011**	**Analog_In3**
	**0100**	**Analog_In4**
	**0101**	**Analog_In5**
	**0110**	**Analog_In6**
	**0111**	**Analog_In7**

Figure 8-4 depicts the pin distribution of the 2X13 header. Connect the trimmer to the ADC channel to read from (Analog_In0 ~ Analog_In7).

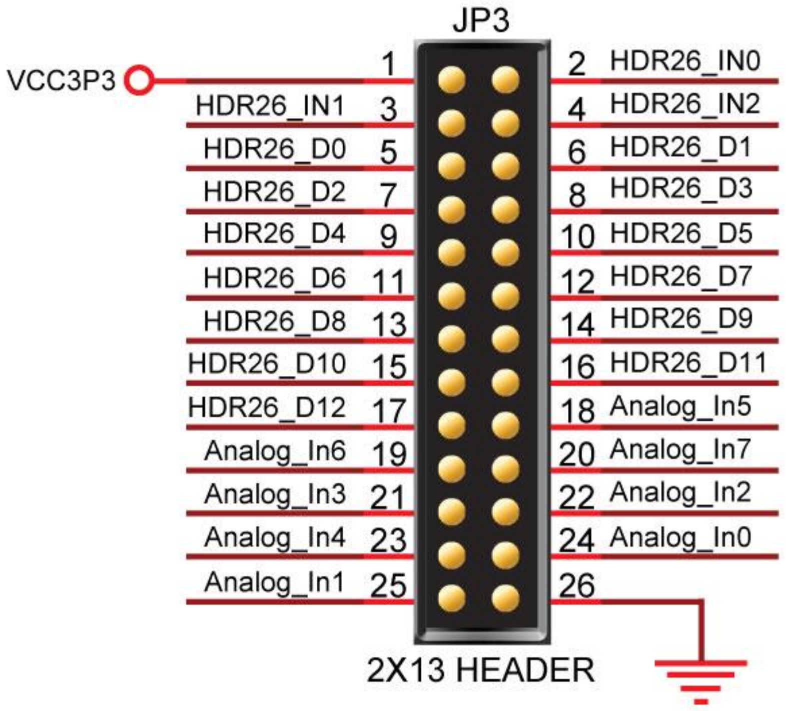


Figure 8-4 2X13 Header

■ System Requirements

The following items are required for the ADC Reading demonstration

- DE0-Nano board x1
- Trimmer Potentiometer x1
- Wire Strip x3

■ Hardware Setup

Figure 8-5 shows the hardware setup for the ADC Reading demonstration.

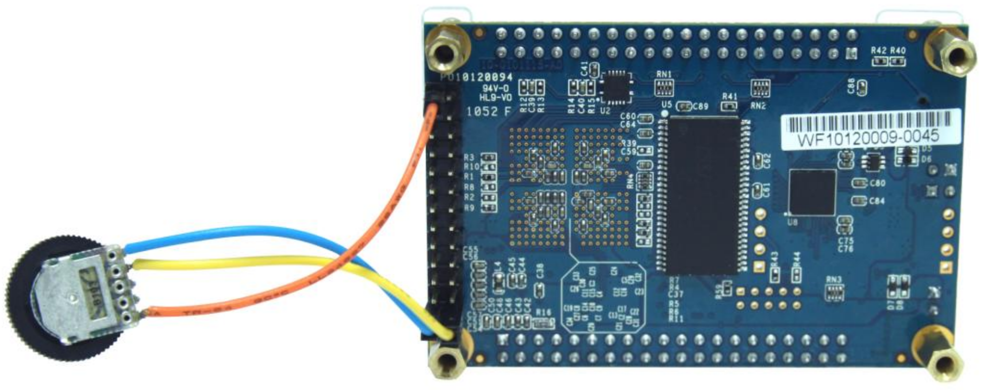


Figure 8-5 ADC Reading hardware setup

Note: the setup shown above is connected ADC channel 1.

Demonstration Source Code

- Project directory: DE0_NANO_ADC
- Bit stream used: DE0_NANO.sof

Demonstration Batch File

Demo Batch File Folder: DE0_NANO_ADC\demo_batch

The demo batch file includes the following files:

- FPGA Configure File: DE0_NANO.sof

Demonstration Setup

- Make sure Quartus II is installed on your PC.
- Connect the trimmer to corresponding ADC channel to read from, as well as the +3.3V and GND signals.
- Adjust the DIP switch according to the ADC channel connected
- Connect USB cable to the DE0-Nano board and install USB Blaster driver if necessary.
- Execute the demo batch file "DE0_NANO_ADC.bat" under the batch file folder, *DE0_NANO_ADC\demo_batch*.
- Adjust the voltage using the trimmer and observe the measurements on the LEDs. Note a fully lit LED bar indicates the voltage is 3.3V and similarly no LED lit indicates 0V.

8.4 SOPC Demo

This section illustrates how to use the SOPC Builder to establish a SOPC System to provide the following functions:

- Control accelerometer through 3-wire SPI interface
- Control analog to digital conversion through 4-wire SPI interface
- Access EEPROM memory through I2C interface
- Access EPCS memory

■ System Block Diagram

This section describes the SOPC System Block Diagram of this demo. Figure 8-6 shows the block diagram.

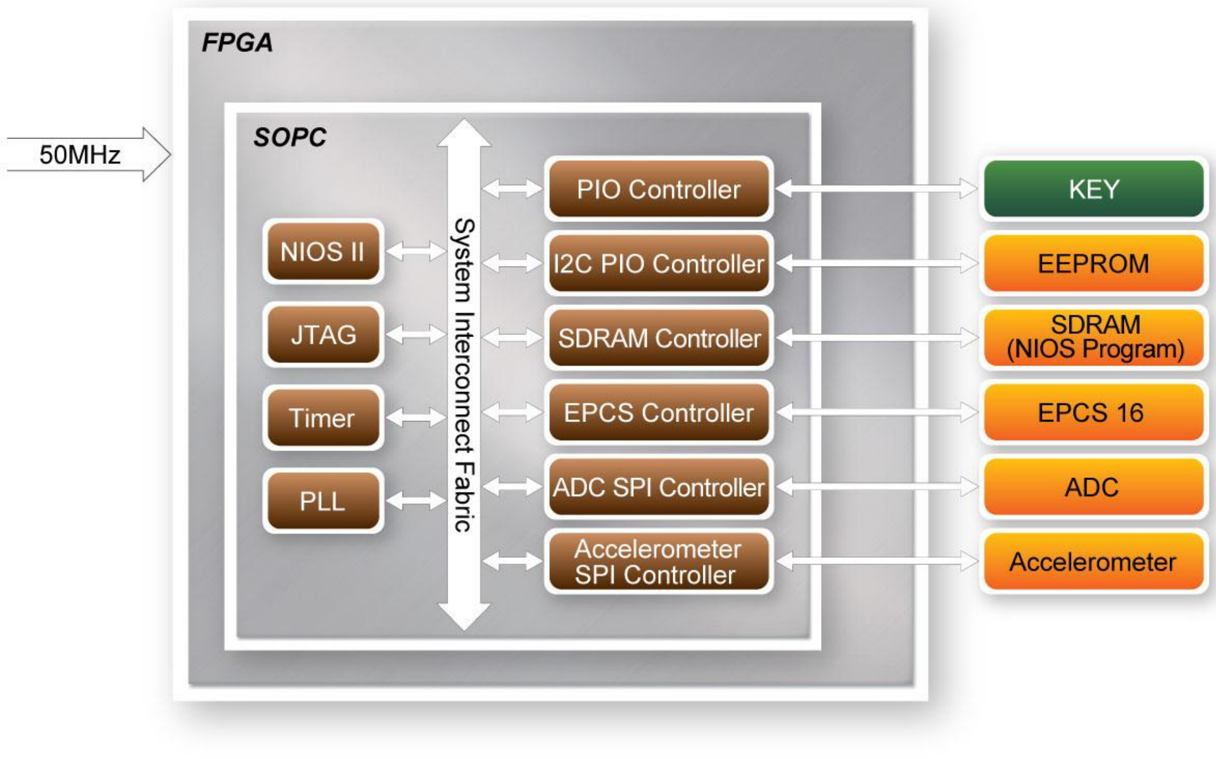


Figure 8-6 SOPC Block Diagram

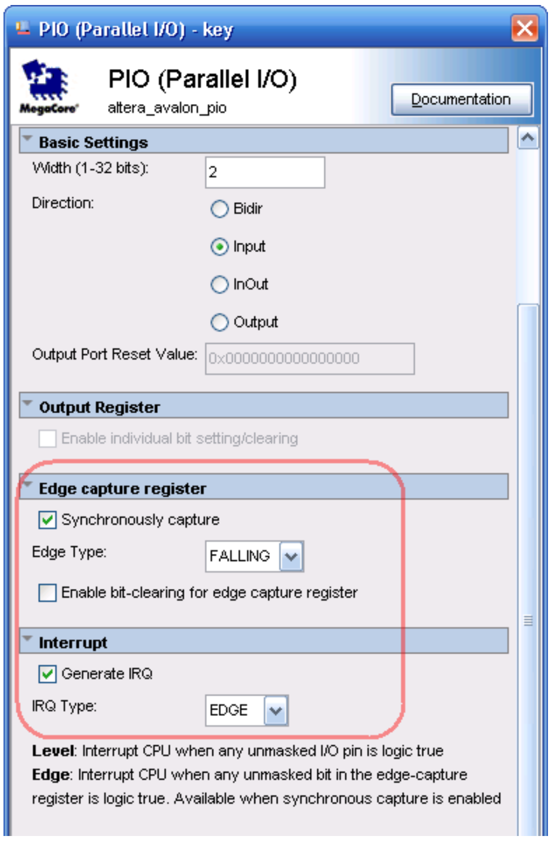
A 50 MHz Clock is required for the SOPC System. A NIOS II processor is included in the system for flow control. The PLL is used to generate clocks, including 100 MHz, 10 MHz and 2MHz. The NIOS II Processor and SDRAM are running at 100 MHZ. The SDRAM is used to store the NIOS II Program. The ADC SPI Controller is running at 2 MHz. Other peripheral controller is running at 10

MHz. The ADC SPI Controller and the Accelerometer SPI Controller are custom SOPC component. The source code is located in the “ip” folder under this Quartus project. The other SOPC components are SOPC Builder build-in components.

■ KEY

The KEY button is driven by PIO Controller with interrupt enabled. It is design to generate an interrupt event when users click KEY0 or KEY1. The interrupt event is used to terminate accelerometer and analog to digital conversion process in this demo.

For default, the interrupt is disabled in PIO Controller. Users can enable it with the parameter setting as shown in below **Figure 8-7**.


PIO (Parallel I/O) - key

PIO (Parallel I/O)
altera_avalon_pio

Basic Settings
Width (1-32 bits): 2
Direction: Bidir [✓ Input] InOut Output
Output Port Reset Value: 0x0000000000000000

Output Register
Enable individual bit setting/clearing

Edge capture register
✓ Synchronously capture
Edge Type: FALLING
Enable bit-clearing for edge capture register

Interrupt
✓ Generate IRQ
IRQ Type: EDGE

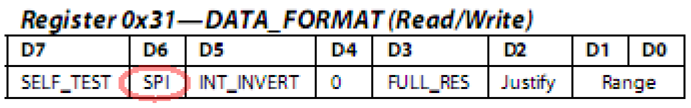
Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Figure 8-7 PIO Controller

■ Accelerometer Control

The accelerometer controller is a custom SOPC component developed by Terasic. The source code is available under the folder \DE0_NANO_SOPC_DEMO\ip\TARASIC_SPI_3WIRE.

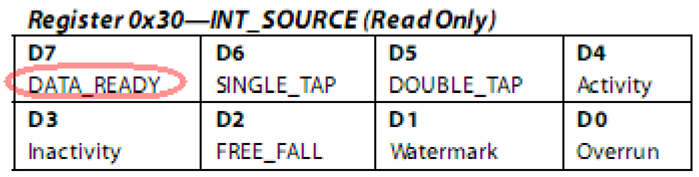
In this demo, the accelerometer is controlled through a 3-wire SPI. Note by default the SPI is a 4-wire interface. Before reading any data from the accelerometer, master should set 1 on the SPI bit in the Register 0x31 – DATA_FORMAT register, as shown in below **Figure 8-8**, to set the device to 3-wire SPI mode.


Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Figure 8-8 DATA_FORMAT Register

The data format is configured as 10 bits, right-justify, +/- 2g mode. The output data rate is configured as 400 HZ. The X/Y/Z value is read using polling mode. Before reading X/Y/Z, the master needs to make sure data is ready by reading the register 0x30-INT_SOURCE, as shown below **Figure 8-9**, and checking the DATA_READY bit. In the demo, multiple-byte read of six bytes X/Y/Z, register from 0x32 to 0x37, is performed to prevent a change in data between reads of sequential register. Note, the output data is twos complement with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z.


Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	D6 SINGLE_TAP	D5 DOUBLE_TAP	D4 Activity
D3 Inactivity	D2 FREE_FALL	D1 Watermark	D0 Overrun

Figure 8-9 Register 0x30

The SPI timing scheme follows clock polarity (CPOL)=1 and clock phase (CPHA)=1. (CPOL)=1 means the clock is high in idle. (CPHA)=1 means data is captured on clock's rising edge and data is propagated on a falling edge. The timing diagram of 3-wire SPI is shown below **Figure 8-10**:

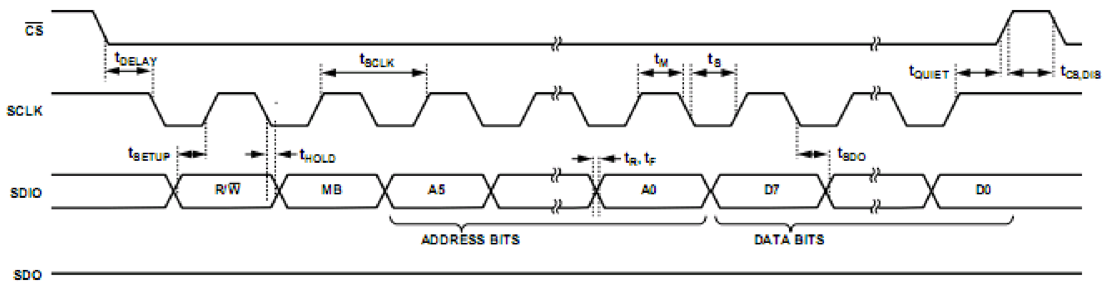


Figure 8-10 3-wire SPI Timing Diagram

■ ADC Control

The Analog to Digital Conversion is controller through a 4-wire SPI interface with the timing dialog given below **Figure 8-11**. Note, the DIN signal is used to specify the channel (IN0~IN7) for the next data conversion. The DOUT signal is used to read the data conversion result whose channel is specified in previous transaction. The first conversion result after power-up will be on IN0. The output format of conversion result is straight binary.

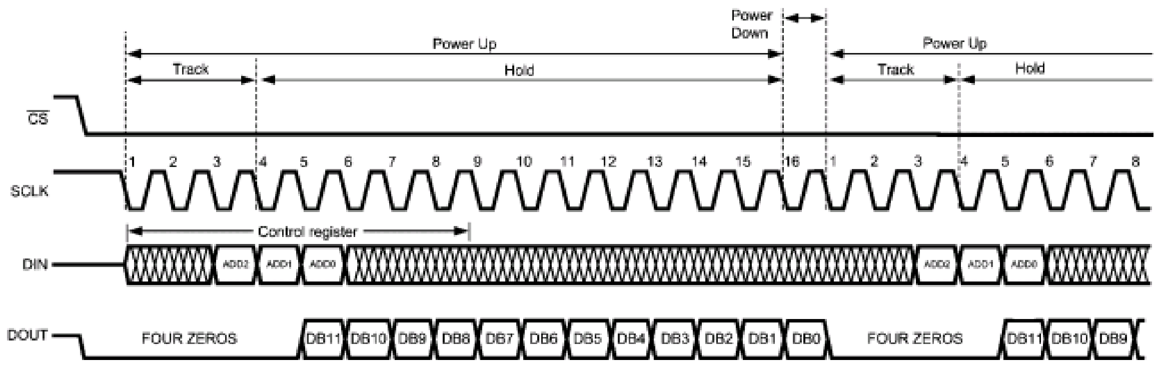


Figure 8-11 4-wire SPI Timing Diagram

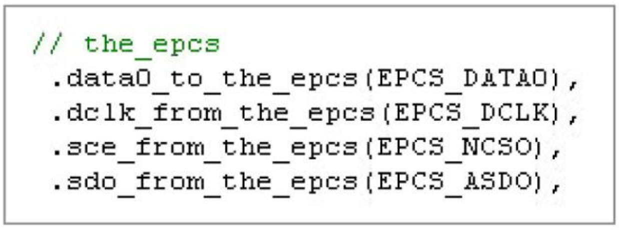
■ EEPROM Control

EEPROM is accessed through the I2C interface. In this demo, I2C signal is toggle by NIOS II through the PIO controller. The I2C clock signal is driver by an OUTPUT PIO Controller and the I2C data signal is driver by a BIDIRECTION PIO Controller. The I2C C code is located in:

DE0_NANO_SOPC_DEMO\software\DE0_NANO\terasic_lib\I2C.c

■ EPCS Control

EPCS16 is accessed through the EPCS interface. In Quartus 10.0 or later, the EPCS pin assignment is required and should be connected the pins to EPCS Controller as shown below **Figure 8-12**:



```
// the_epcs
  .data0_to_the_epcs(EPCS_DATA0),
  .dclk_from_the_epcs(EPCS_DCLK),
  .sce_from_the_epcs(EPCS_NCSO),
  .sdo_from_the_epcs(EPCS_ASDO),
```

Figure 8-12 EPCS interface connection

For the EPCS access functions, users can refer to:

DE0_NANO_SOPC_DEMO\software\DE0_NANO\terasic_lib\Flash.c

Demonstration Source Code

- Project directory: DE0_NANO_SOPC_DEMO
- Bit stream used: DE0_NANO.sof
- NIOS II elf file: DE0_NANO.elf

Demonstration Batch File

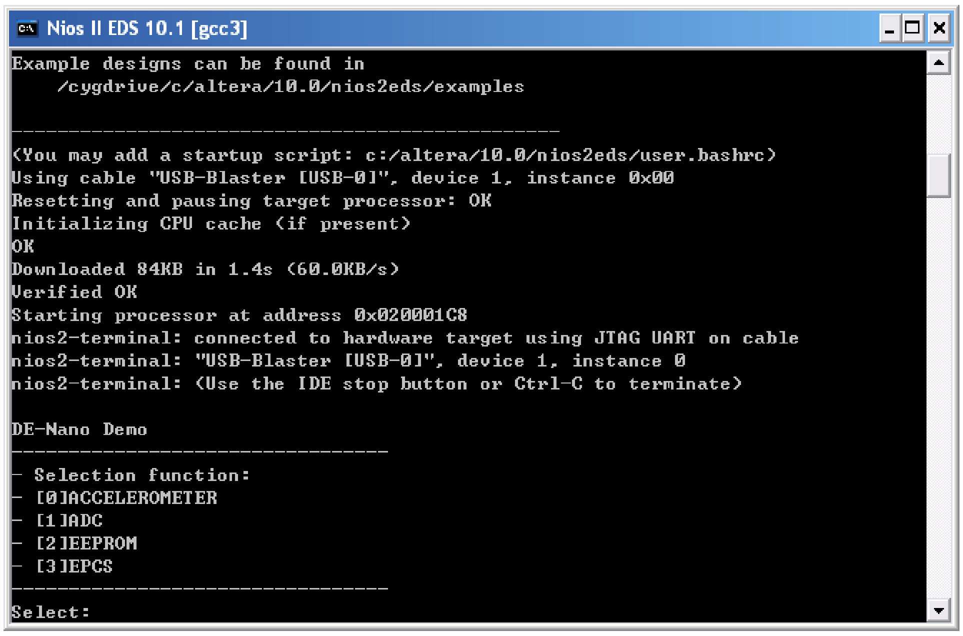
- Demo Batch File Folder: DE0_NANO_SOPC_DEMO\demo_batch

The demo batch file includes the file:

- Batch File: test.bat and test_bashrc
- FPGA Configure File: DE0_NANO.sof
- Nios II Program: DE0_NANO.elf

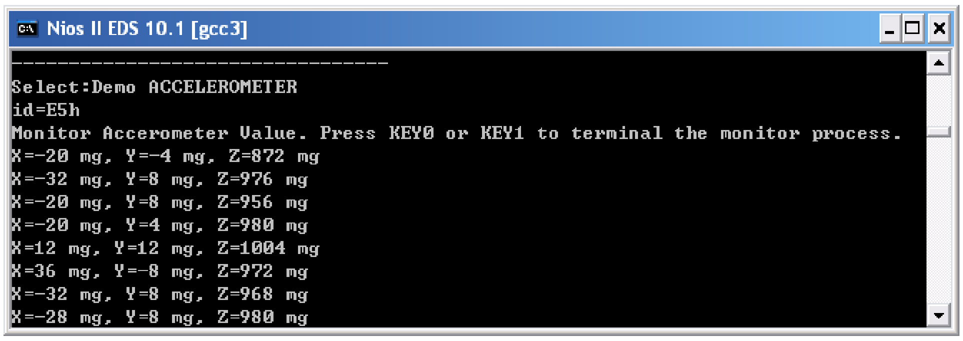
Demonstration Setup

- Make sure Quartus II and Nios II are installed on your PC.
- Connect a USB cable to the DE0-Nano board and install USB Blaster driver if necessary.
- Execute the demo batch file “test.bat” under the batch file folder, *DE0_NANO_SOPC_DEMO\demo_batch*.
- A menu dialog appears as follows:



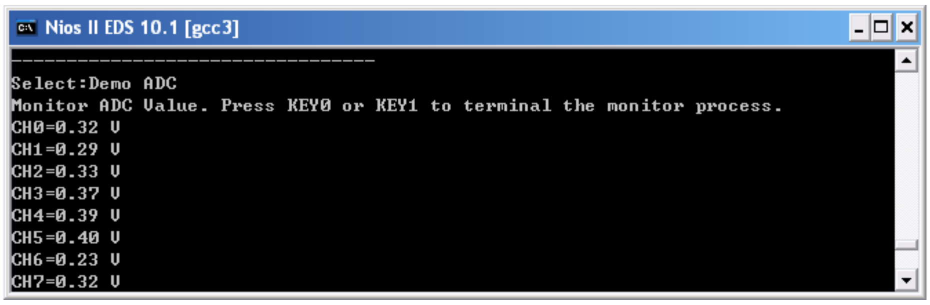
```
Example designs can be found in
    /cygdrive/c/altera/10.0/nios2eds/examples

------------------------------------------------
(You may add a startup script: c:/altera/10.0/nios2eds/user.bashrc)
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 84KB in 1.4s (60.0KB/s)
Verified OK
Starting processor at address 0x020001C8
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

DE-Nano Demo
---------------------------------
- Selection function:
- [0]ACCELEROMETER
- [1]ADC
- [2]EEPROM
- [3]EPCS
---------------------------------
Select:
```

- Input “0” to start Accelerometer demo. In the demo, the accelerometer chip ID is shown, and then X/Y/Z value is displayed every 1.0 second. To terminate the process, just click KEY0 or KEY1 in the DE0-Nano board.



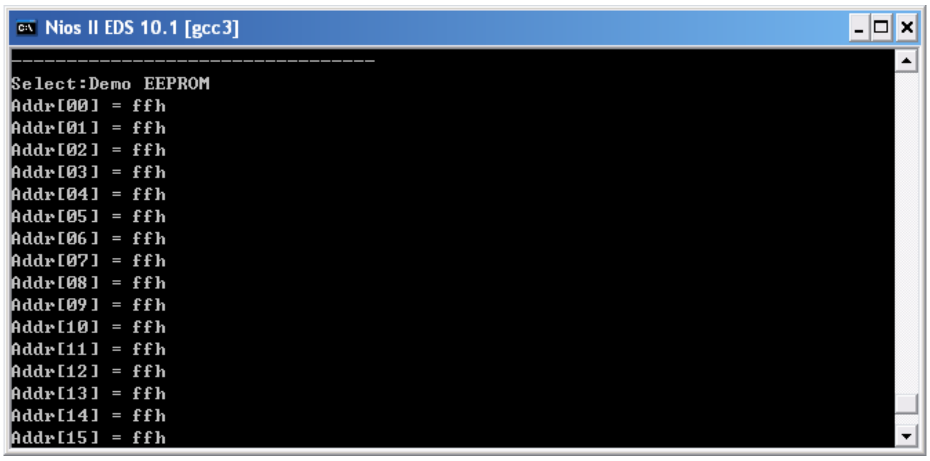
```
---------------------------------
Select:Demo ACCELEROMETER
id=E5h
Monitor Accerometer Value. Press KEY0 or KEY1 to terminal the monitor process.
X=-20 mg, Y=-4 mg, Z=872 mg
X=-32 mg, Y=8 mg, Z=976 mg
X=-20 mg, Y=8 mg, Z=956 mg
X=-20 mg, Y=4 mg, Z=980 mg
X=12 mg, Y=12 mg, Z=1004 mg
X=36 mg, Y=-8 mg, Z=972 mg
X=-32 mg, Y=8 mg, Z=968 mg
X=-28 mg, Y=8 mg, Z=980 mg
```

- Menu dialog appears again. Input “1” to start Analog to Digital Conversion Process. In the demo, the voltage of eight channels is shown repeatedly. To terminate the process, just click KEY0 or KEY1 in the DE0-Nano board.



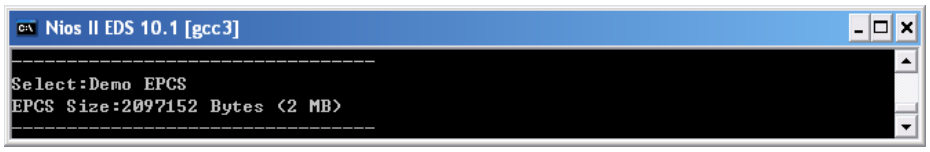
```
---------------------------------
Select:Demo ADC
Monitor ADC Value. Press KEY0 or KEY1 to terminal the monitor process.
CH0=0.32 V
CH1=0.29 V
CH2=0.33 V
CH3=0.37 V
CH4=0.39 V
CH5=0.40 V
CH6=0.23 V
CH7=0.32 V
```

- Menu dialog appears again. Input “2” to start EEPROM Content Dump process. In the demo, the value of first 16 bytes is dumped. Then, return to the menu dialog.



```
----------------------------------
Select:Demo EEPROM
Addr[00] = ffh
Addr[01] = ffh
Addr[02] = ffh
Addr[03] = ffh
Addr[04] = ffh
Addr[05] = ffh
Addr[06] = ffh
Addr[07] = ffh
Addr[08] = ffh
Addr[09] = ffh
Addr[10] = ffh
Addr[11] = ffh
Addr[12] = ffh
Addr[13] = ffh
Addr[14] = ffh
Addr[15] = ffh
```

- In the Menu dialog, input “3” to start EPCS demo process. In the demo, the memory size of EPCS is shown as follows.



```
----------------------------------
Select:Demo EPCS
EPCS Size:2097152 Bytes (2 MB)
----------------------------------
```

8.5 G-Sensor

In this demonstration, we illustrate the steps to evaluate the digital accelerometer on the DE0-Nano board to measure the static acceleration of gravity in tilt-sensing applications. The board can be tilted from left to right as the digital accelerometer detects any tilting movement and displays it on the LEDs.

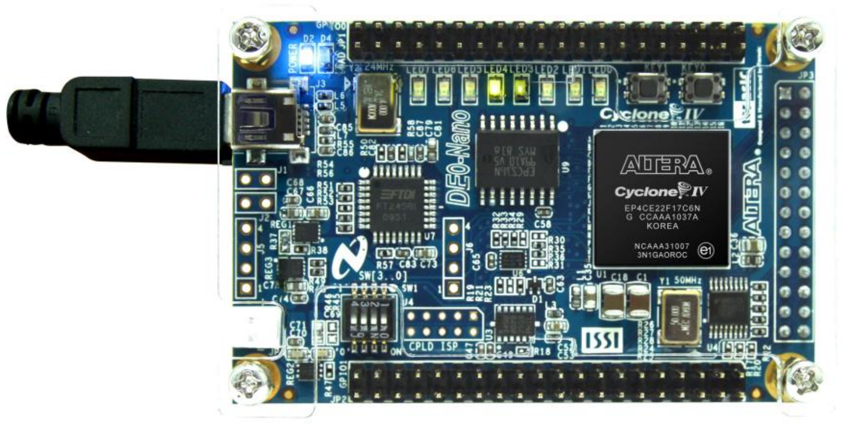


Figure 8-13 DE0-Nano on level surface

Design Concept

This section describes the design concepts for this demo. Figure 8-14 shows the block diagram.

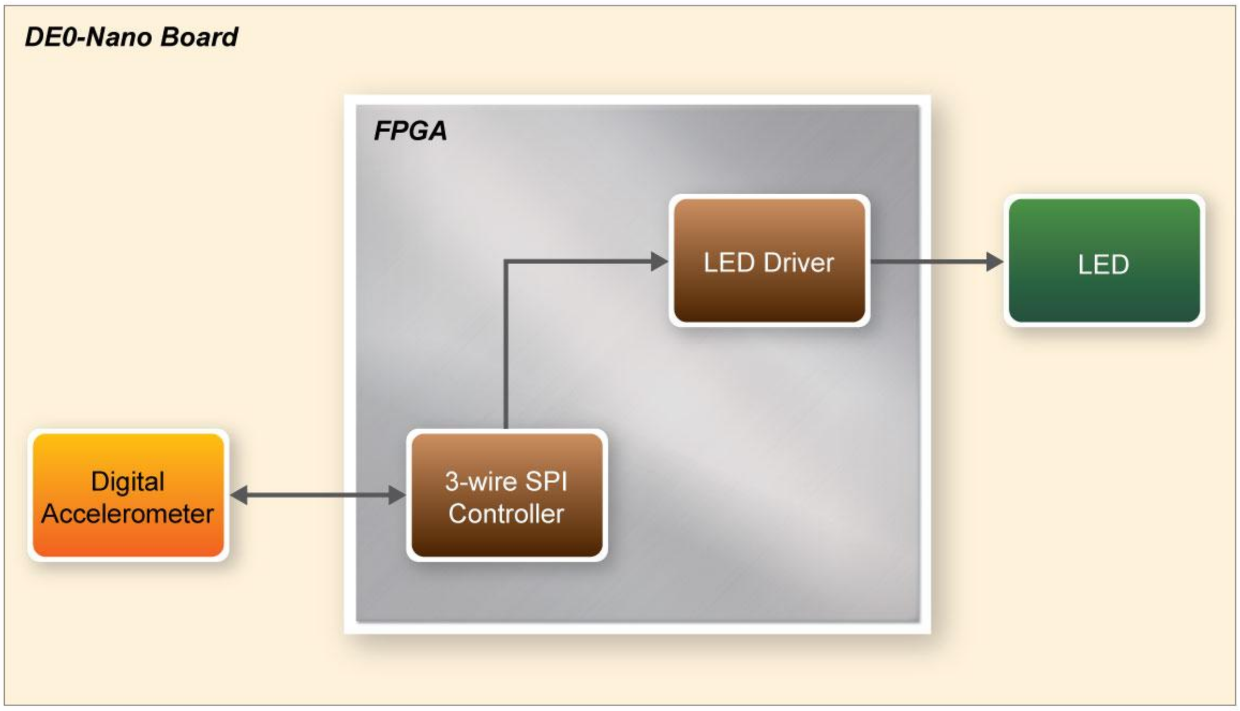


Figure 8-14 G-Sensor block diagram

In this demo, the accelerometer is controlled through a 3-wire SPI. Note by default the SPI is a 4-wire interface. Before reading any data from the accelerometer, master should set 1 on the SPI bit in the Register 0x31 – DATA_FORMAT register. The 3-wire SPI Controller block reads the digital accelerometer X-axis value and displays the degree scale on the LEDs.

Demonstration Source Code

- Project directory: DE0_NANO_GSensor
- Bit stream used: DE0_NANO_G_Sensor.sof

Demonstration Batch File

Demo Batch File Folder: DE0_NANO_GSensor\demo_batch

The demo batch file includes the following files:

- FPGA Configure File: DE0_NANO_G_Sensor.sof

Demonstration Setup

- Make sure Quartus II is installed on your PC.
- Connect USB cable to the DE0-Nano board and install USB Blaster driver if necessary.
- Execute the demo batch file “test.bat” under the batch file folder, *DE0_NANO_GSensor\demo_batch*.
- Tilt the DE0-Nano board (side to side) and observe the result on the LEDs.

Chapter 9

Appendix

9.1 Appendix A

Programming the Serial Configuration Device

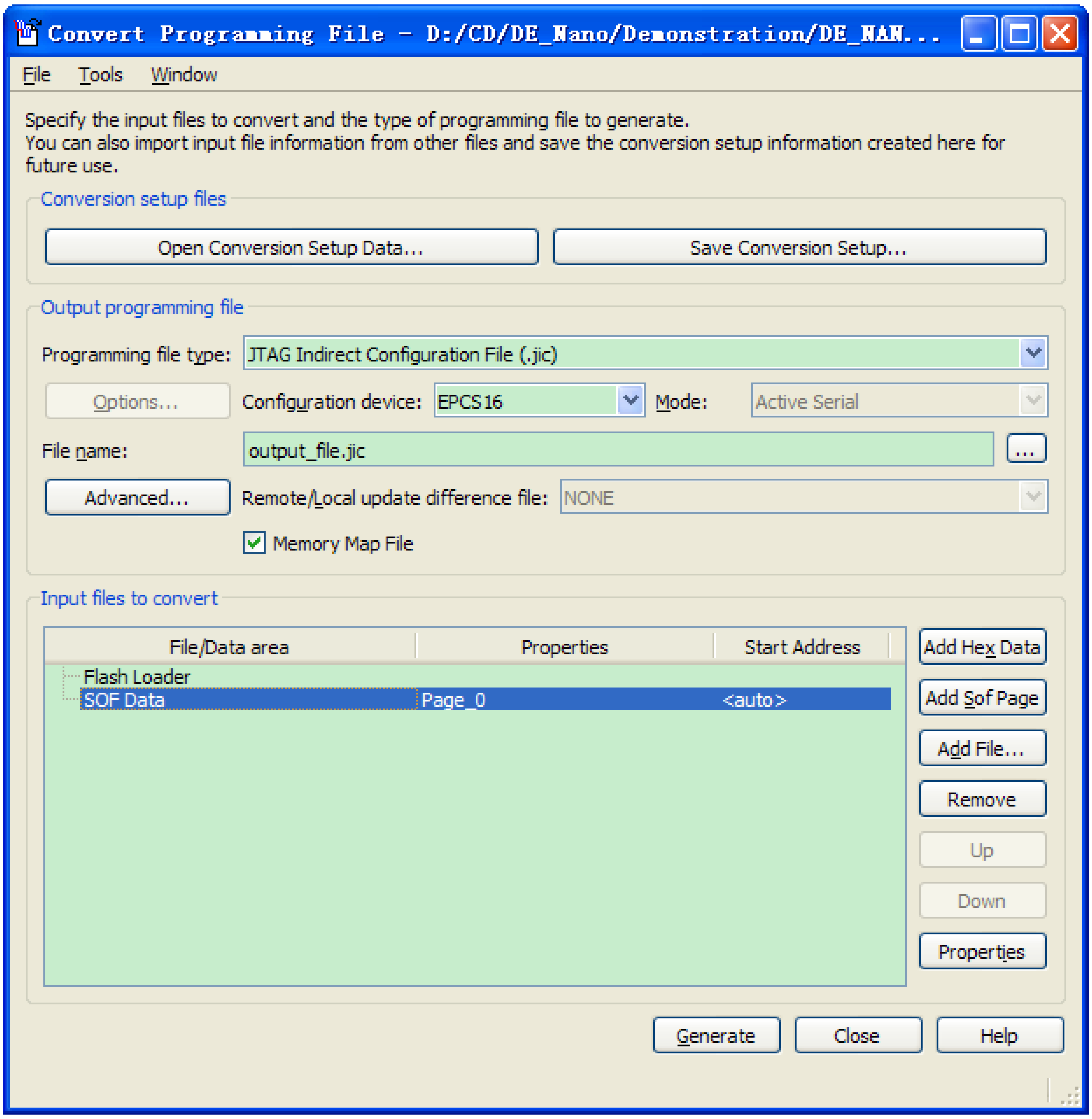
This appendix describes how to program the serial configuration device with Serial Flash Loader (SFL) function via the JTAG interface. User can program serial configuration devices with a JTAG indirect configuration (.jic) file. To generate JIC programming files with the Quartus II software, users need to generate a user-specified SRAM object file (.sof), which is the input file first. Next, users need to convert the SOF to a JIC file. To convert a SOF to a JIC file in Quartus II software, follow these steps:

- **Convert SOF to JIC**

1. Choose **Convert Programming Files** (File menu).
2. In the **Convert Programming Files** dialog box, scroll to the **JTAG Indirect Configuration File** (.jic) from the **Programming file type** field.
3. In the **Configuration device** field, specify the targeted serial configuration device (please select EPCS16).
4. In the **File name** field, browse to the target directory and specify an output file name.
5. Highlight the SOF data in the Input files to convert section. See Figure A1.
6. Click **Add File**.
7. Select the SOF that you want to convert to a JIC file.
8. Click **Open**.

9. Highlight the Flash Loader and click **Add Device**. See Figure A2.

10. Click **OK**. The Select Devices page displays.


Convert Programming File - D:/CD/DE_Nano/Demonstration/DE_NAN...

File Tools Window

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Options... | Configuration device: EPCS16 | Mode: Active Serial

File name: output_file.jic

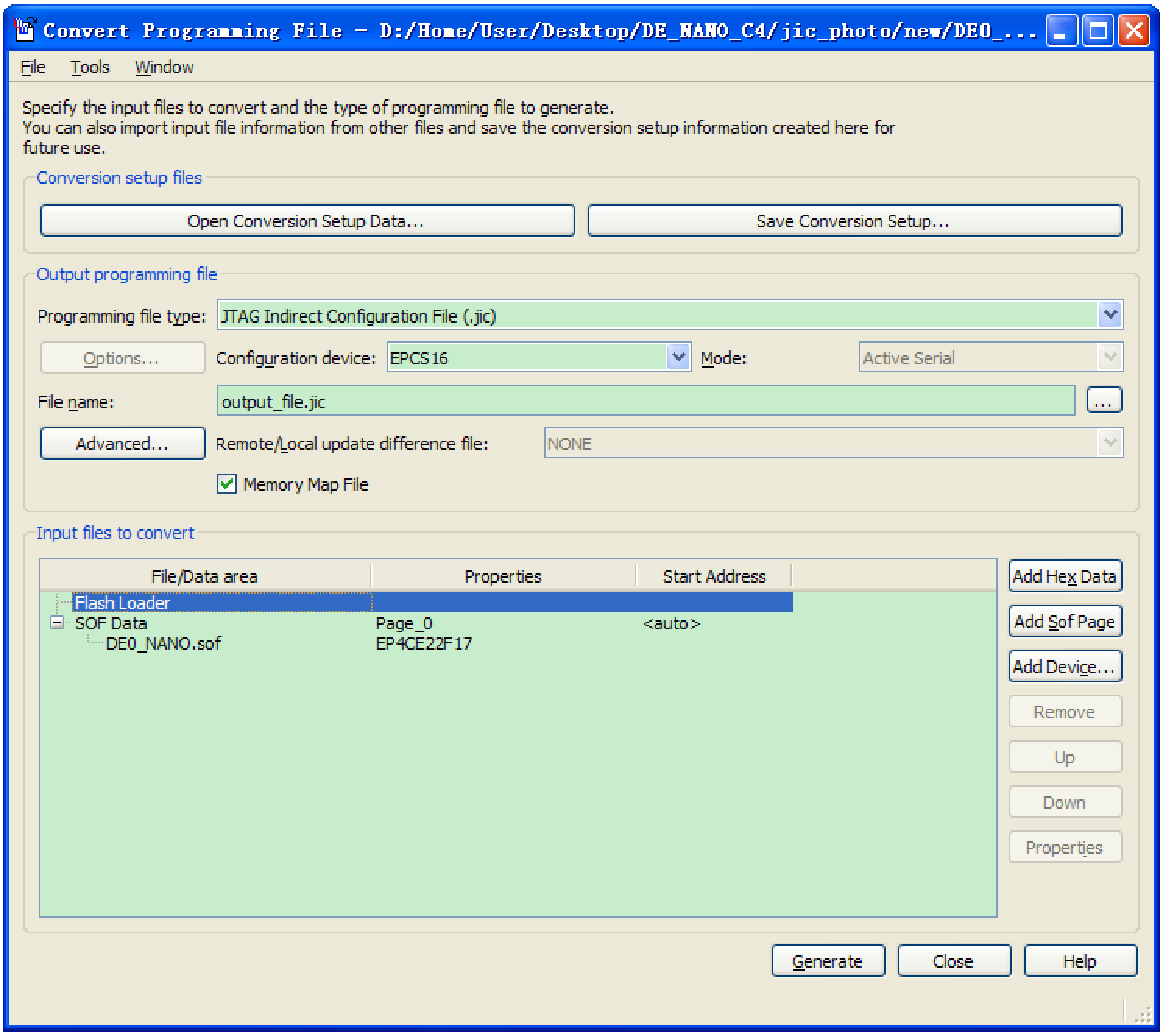
Advanced... | Remote/Local update difference file: NONE

☑ Memory Map File

Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>

Figure A1 Convert Programming Files Dialog Box


Convert Programming File - D:/Home/User/Desktop/DE_NANO_C4/jic_photo/new/DE0_...

File Tools Window

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Options... Configuration device: EPCS16 Mode: Active Serial

File name: output_file.jic

Advanced... Remote/Local update difference file: NONE

✓ Memory Map File

Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>
DE0_NANO.sof	EP4CE22F17	

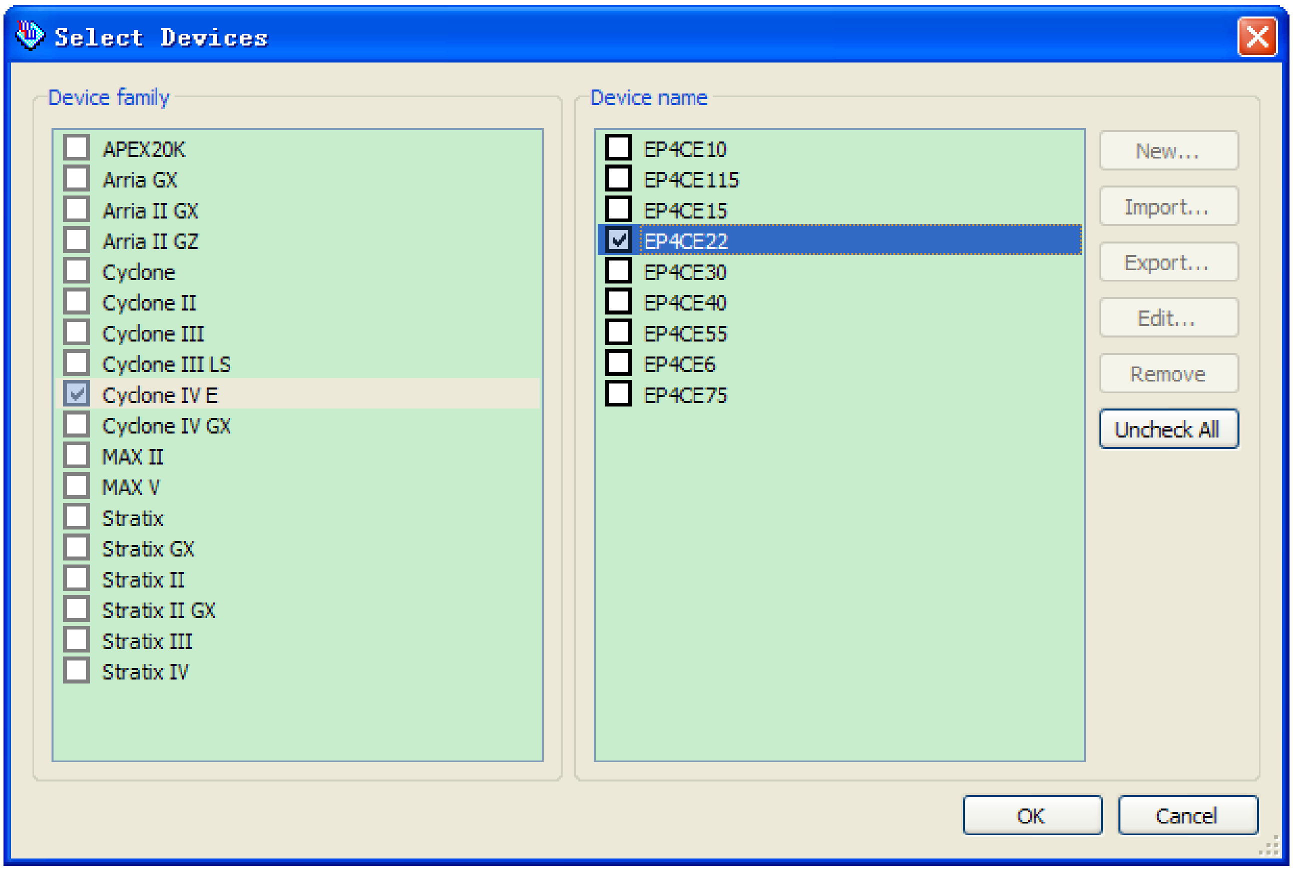
Figure A2 Highlight Flash Loader

11. Select the targeted FPGA that you are using to program the serial configuration device. See Figure A3.

12. Click OK. The **Convert Programming Files** page displays. See Figure A4.

13. Select the .sof file, and Click the **Properties**. Select Compression, click **OK** See Figure A5.

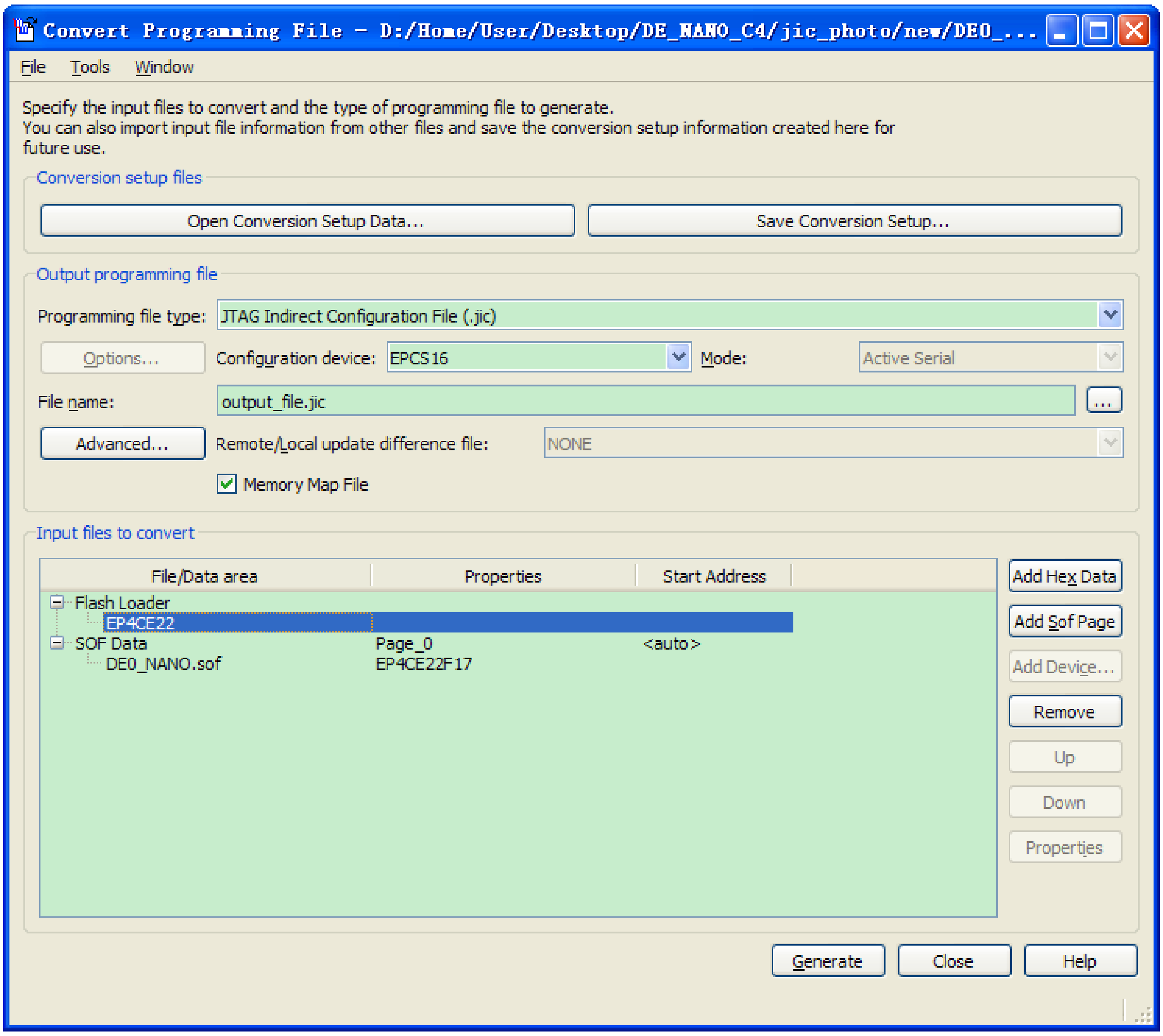
14. Click **Generate**.



Device family:
- [ ] APEX20K
- [ ] Arria GX
- [ ] Arria II GX
- [ ] Arria II GZ
- [ ] Cyclone
- [ ] Cyclone II
- [ ] Cyclone III
- [ ] Cyclone III LS
- [✓] Cyclone IV E
- [ ] Cyclone IV GX
- [ ] MAX II
- [ ] MAX V
- [ ] Stratix
- [ ] Stratix GX
- [ ] Stratix II
- [ ] Stratix II GX
- [ ] Stratix III
- [ ] Stratix IV

Device name:
- [ ] EP4CE10
- [ ] EP4CE115
- [ ] EP4CE15
- [✓] EP4CE22
- [ ] EP4CE30
- [ ] EP4CE40
- [ ] EP4CE55
- [ ] EP4CE6
- [ ] EP4CE75

Figure A3 Select Devices Page


Convert Programming File - D:/Home/User/Desktop/DE_NANO_C4/jic_photo/new/DE0_...

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Configuration device: EPCS16

Mode: Active Serial

File name: output_file.jic

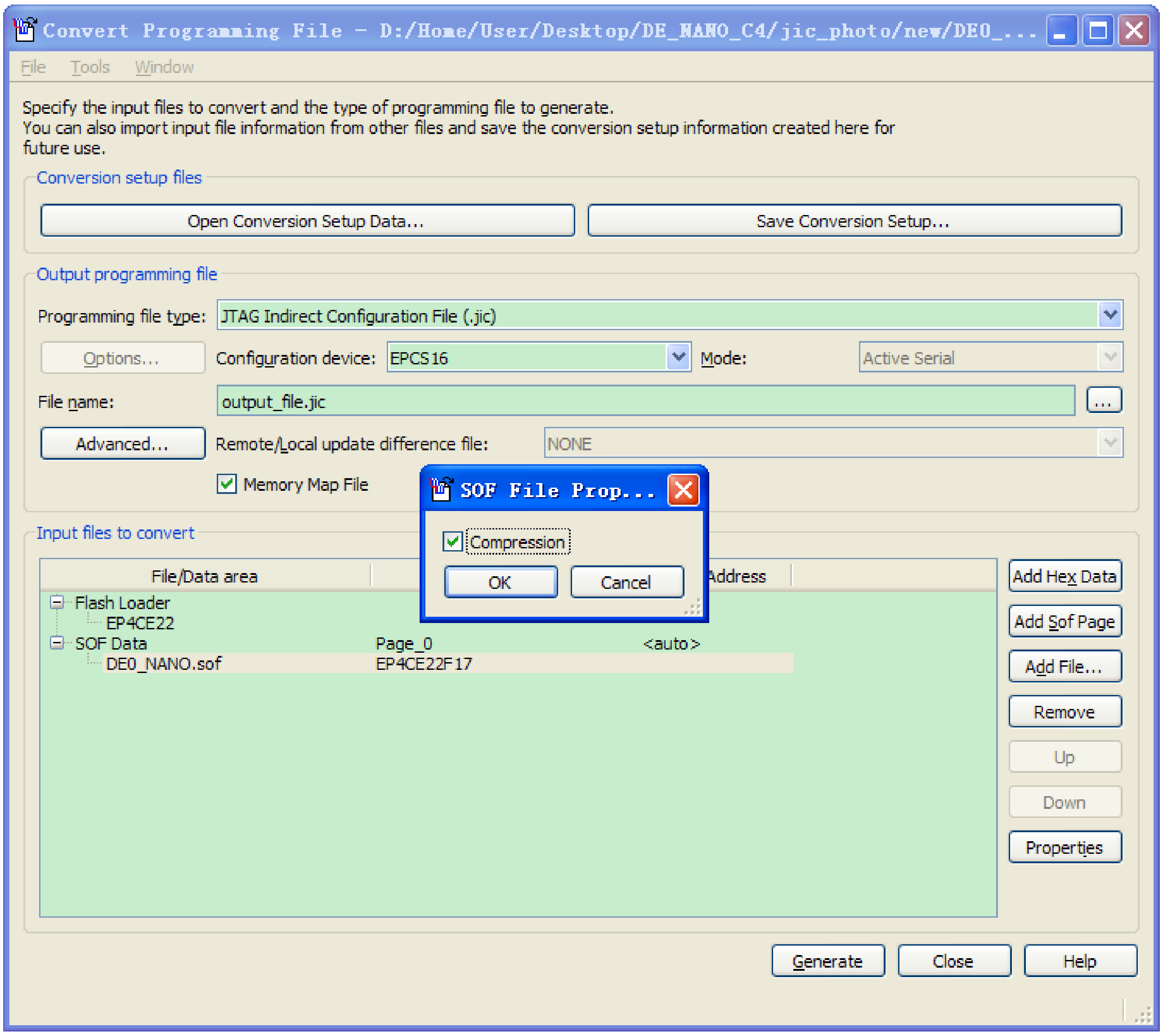
Remote/Local update difference file: NONE

✓ Memory Map File

Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
EP4CE22		
SOF Data	Page_0	<auto>
DE0_NANO.sof	EP4CE22F17	

Figure A4. Convert Programming Files Page


Convert Programming File - D:/Home/User/Desktop/DE_NANO_C4/jic_photo/new/DE0_...

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Open Conversion Setup Data... | Save Conversion Setup...

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Configuration device: EPCS16 Mode: Active Serial

File name: output_file.jic

Remote/Local update difference file: NONE

☑ Memory Map File

SOF File Prop...

☑ Compression

Input files to convert

File/Data area		Address
Flash Loader		
EP4CE22		
SOF Data	Page_0	<auto>
DE0_NANO.sof	EP4CE22F17	

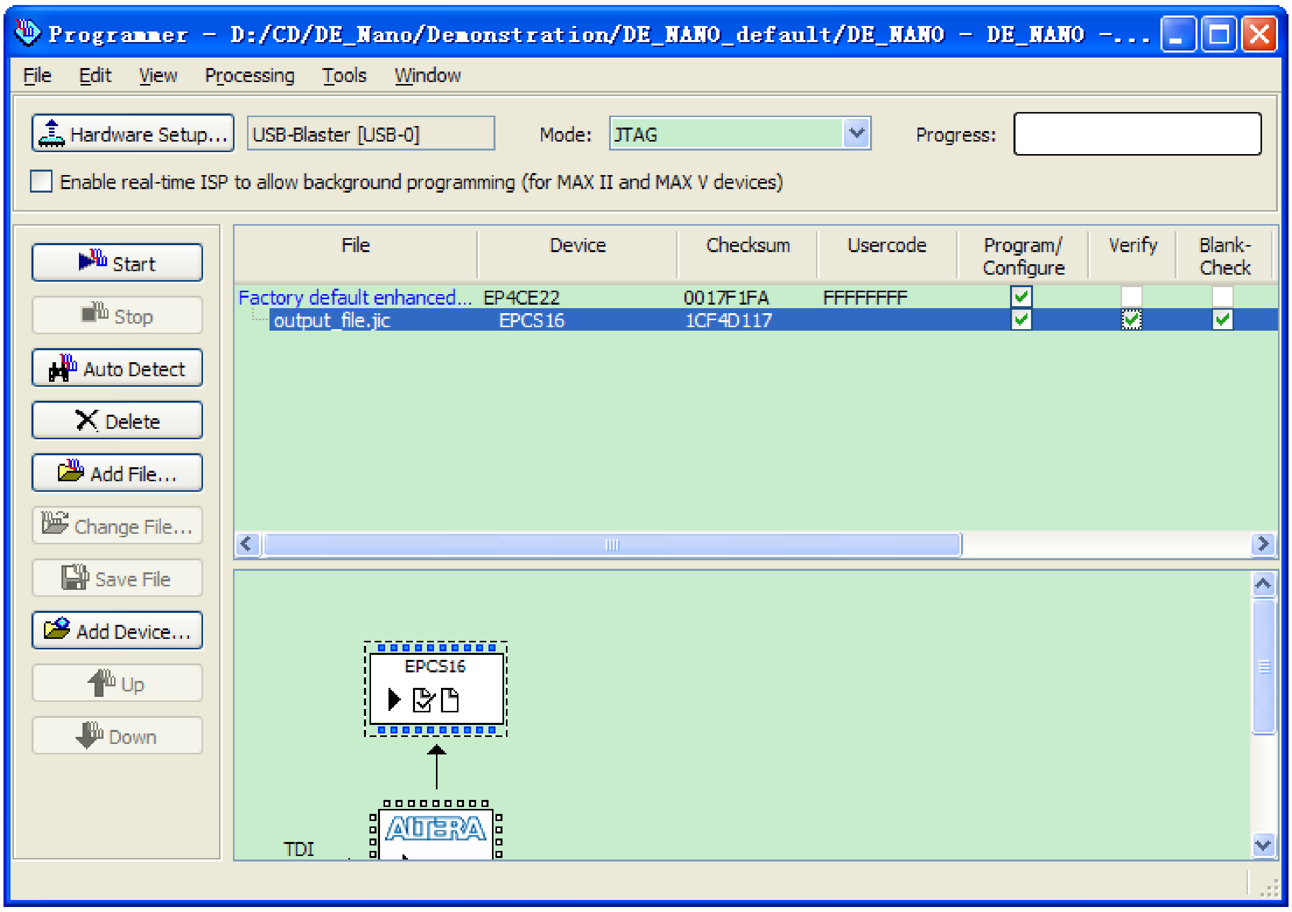
Figure A5 Compression the sof file

Write JIC File into Serial Configuration Device

To program the serial configuration device with the JIC file that you just created, add the file to the Quartus II Programmer window and follow the steps:

1. When the SOF-to-JIC file conversion is complete, add the JIC file to the Quartus II Programmer window:
   i. Choose **Programmer** (Tools menu). The **Chain1.cdf** window displays.
   ii. Click **Add File**. From the **Select Programming File** page, browse to the JIC file.
   iii. Click **Open**.

2. Program the serial configuration device by checking the corresponding **Program/Configure** box, a Factory default SFL image will be load (See Figure A6).


Programmer - D:/CD/DE_Nano/Demonstration/DE_NANO_default/DE_NANO - DE_NANO -...

File Edit View Processing Tools Window

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check
Factory default enhanced...	EP4CE22	0017F1FA	FFFFFFFF	✓		
output_file.jic	EPCS16	1CF4D117		✓	✓	✓

Figure A6. Quartus II programmer window with one JIC file

3. Click **Start** to program serial configuration device.

■ Erase the Serial Configuration Device

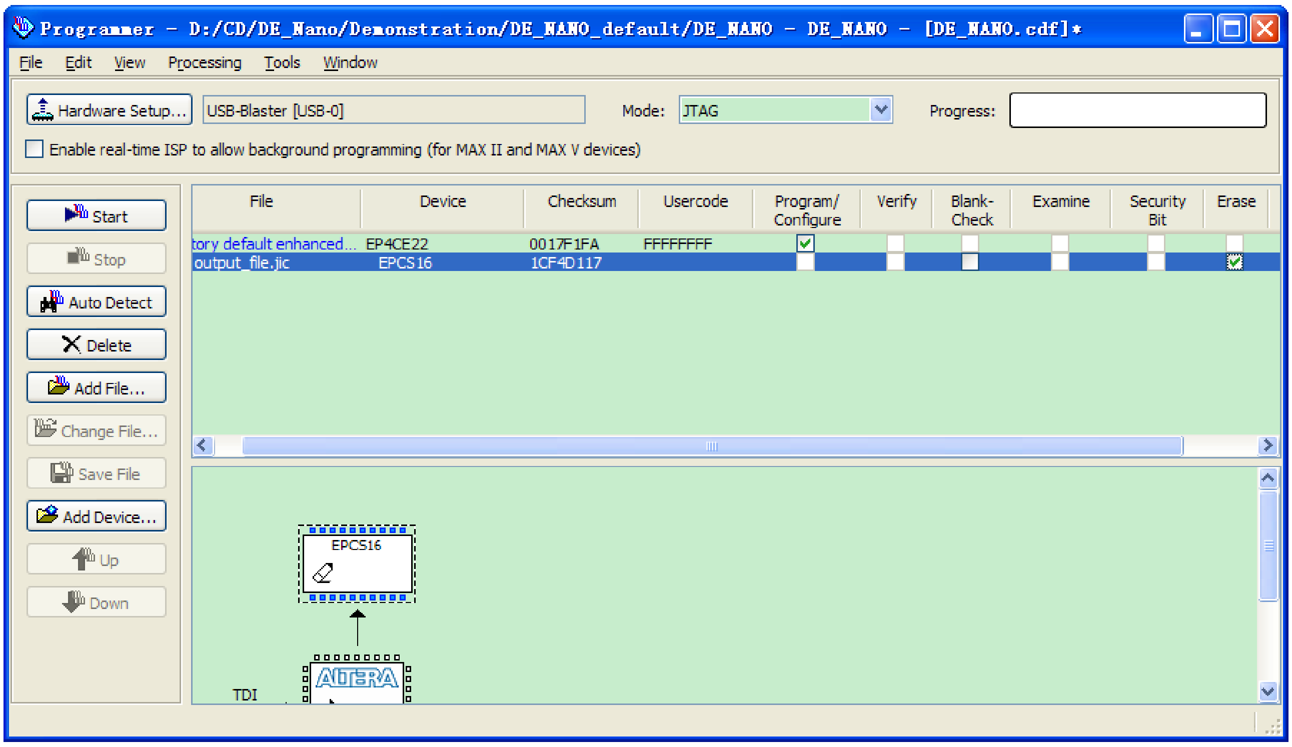
To erase the existed file in the serial configuration device, follow the steps listed below:

1. Choose **Programmer** (Tools menu). The **Chain1.cdf** window displays.

2. Click **Add File**. From the Select Programming File page, browse to a JIC file.

3. Click **Open.**

4. Erase the serial configuration device by checking the corresponding **Erase** box, a Factory default SFL image will be load (See Figure A7).


Programmer - D:/CD/DE_Nano/Demonstration/DE_NANO_default/DE_NANO - DE_NANO - [DE_NANO.cdf]*

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase
tory default enhanced...	EP4CE22	0017F1FA	FFFFFFFF	✓					
output_file.jic	EPCS16	1CF4D117							✓

Figure A7 Erasing setting in Quartus II programmer window

5. Click **Start** to erase the serial configuration device.

Chapter 10

Appendix

10.1 Revision History

Version	***Change Log***
V1.0	**Initial Version (Preliminary)**

10.2 Copyright Statement

Always visit DE0-Nano webpage for new applications.

We will be continuing providing interesting examples and labs on our DE0-Nano webpage. Please visit www.altera.com or DE0-Nano.terasic.com for more information.