Introduction to the Altera Qsys System Integration Tool

For Quartus Prime 15.1

1 Introduction

This tutorial presents an introduction to Altera's Qsys system integration tool, which is used to design digital hardware systems that contain components such as processors, memories, input/output interfaces, timers, and the like. The Qsys tool allows a designer to choose the components that are desired in the system by selecting these components in a graphical user interface. It then automatically generates the hardware system that connects all of the components together.

The hardware system development flow is illustrated by giving step-by-step instructions for using the Qsys tool in conjunction with the Quartus® Prime software to implement a simple example system. The last step in the development process involves configuring the designed hardware system in an actual FPGA device, and running an application program. To show how this is done, it is assumed that the user has access to an Altera DE-series Development and Education board connected to a computer that has Quartus Prime and Nios® II software installed. The screen captures in the tutorial were obtained using the Quartus Prime version 15.1; other versions of the software may be slightly different.

Contents:

2 Altera DE-series FPGA Boards

For this tutorial we assume that the reader has access to an Altera DE-series board, such as the one shown in Figure 1. The figure depicts the DE1-SoC board, which features an Altera Cyclone IV FPGA chip. The board provides a lot of other resources, such as memory chips, slider switches, pushbutton keys, LEDs, audio input/output, video input (NTSC/PAL decoder) and video output (VGA). It also provides several types of serial input/output connections, including a USB port for connecting the board to a personal computer. In this tutorial we will make use of only a few of the resources: the FPGA chip, slider switches, LEDs, and the USB port that connects to a computer.

Although we have chosen the DE1-SoC board as an example, the tutorial is pertinent for other DE-series boards that are described in the University Program section of Altera's website.

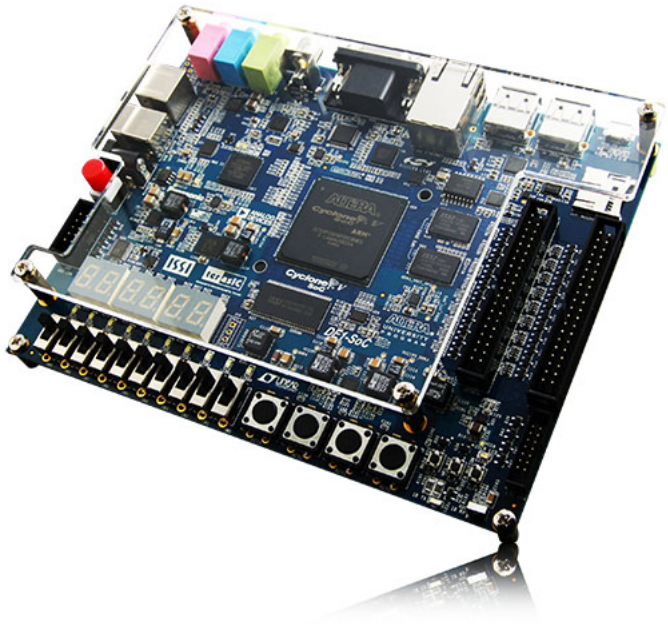

Figure 1. An Altera DE1-SoC board.

3 A Digital Hardware System Example

We will use a simple hardware system that is shown in Figure 2. It includes the Altera Nios® II embedded processor, which is a *soft processor* module defined as code in a hardware-description language. A Nios II module can be included as part of a larger system, and then that system can be implemented in an Altera FPGA chip by using the Quartus Prime software.

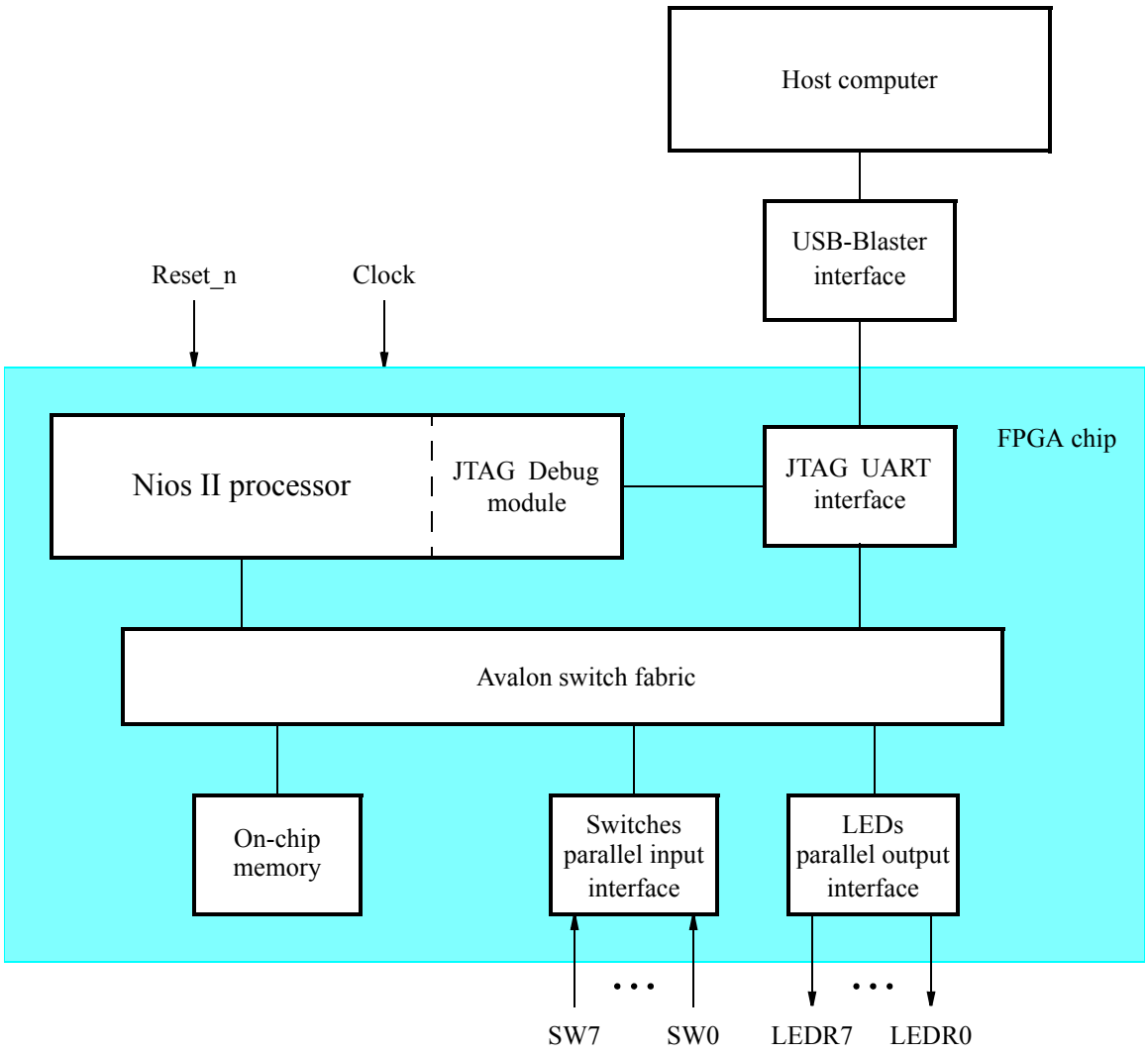

Figure 2. A simple example of a Nios II system.

As shown in Figure 2, the Nios II processor is connected to the memory and I/O interfaces by means of an interconnection network called the *Avalon switch fabric*. This interconnection network is automatically generated by the Qsys tool.

The memory component in our system will be realized by using the on-chip memory available in the FPGA chip. The I/O interfaces that connect to the slider switches and LEDs will be implemented by using the predefined modules that are available in the Qsys tool. A special JTAG UART interface is used to connect to the circuitry that provides a USB link to the host computer to which the DE-series board is connected. This circuitry and the associated software is called the *USB-Blaster*. Another module, called the JTAG Debug module, is provided to allow the host computer

to control the Nios II system. It makes it possible to perform operations such as downloading Nios II programs into memory, starting and stopping the execution of these programs, setting breakpoints, and examining the contents of memory and Nios II registers.

Since all parts of the Nios II system implemented on the FPGA chip are defined by using a hardware description language, a knowledgeable user could write such code to implement any part of the system. This would be an onerous and time consuming task. Instead, we will show how to use the Qsys tool to implement the desired system simply by choosing the required components and specifying the parameters needed to make each component fit the overall requirements of the system. Although in this tutorial we illustrate the capability of the Qsys tool by designing a very simple system, the same approach is used to design larger systems.

Our example system in Figure 2 is intended to realize a trivial task. Eight slider switches on the DE1-SoC board, $SW7-0$, are used to turn on or off eight LEDs, $LEDR7-0$. To achieve the desired operation, the eight-bit pattern corresponding to the state of the switches has to be sent to the output port to activate the LEDs. This will be done by having the Nios II processor execute a program stored in the on-chip memory. Continuous operation is required, such that as the switches are toggled the lights change accordingly.

In the next section we will use the Qsys tool to design the hardware depicted in Figure 2. After assigning the FPGA pins to realize the connections between the parallel interfaces and the switches and LEDs on the DE1-SoC board, we will compile the designed system. Finally, we will use the software tool called the *Altera Monitor Program* to download the designed circuit into the FPGA device, and download and execute a Nios II program that performs the desired task.

Doing this tutorial, the reader will learn about:

- Using the Qsys tool to design a Nios II-based system
- Integrating the designed Nios II system into a Quartus Prime project
- Implementing the designed system on the DE1-SoC board
- Running an application program on the Nios II processor

4 Altera's Qsys Tool

The Qsys tool is used in conjunction with the Quartus Prime CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters. To implement the system in Figure 2, we have to instantiate the following functional units:

- Nios II processor
- On-chip memory, which consists of the memory blocks in the FPGA chip; we will specify a 4-Kbyte memory arranged in 32-bit words
- Two parallel I/O interfaces

- JTAG UART interface for communication with the host computer

To define the desired system, start the Quartus Prime software and perform the following steps:

1. Create a new Quartus Prime project for your system. As shown in Figure 3, we stored our project in a directory called *qsys_tutorial*, and we assigned the name *lights* to both the project and its top-level design entity. You can choose a different directory or project name. Step through the screen for adding design files to the project; we will add the required files later in the tutorial. In your project, choose the FPGA device used on your DE-series board. A list of FPGA devices on the DE-series boards is given in Table 1.

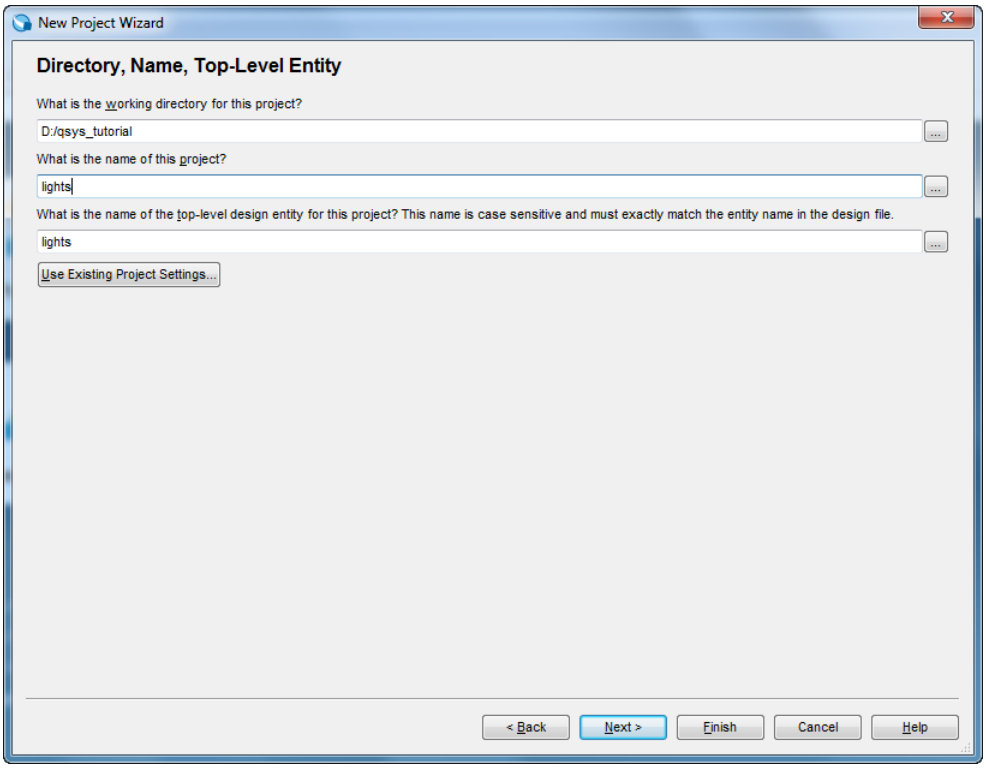

Figure 3. Create a new project.

Board	Device Name
DE0-CV	Cyclone V 5CEBA4F23C7
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE0-Nano-SoC	Cyclone V SoC 5CSEMA4U23C6
DE1-SoC	Cyclone V SoC 5CSEMA5F31C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1. DE-series FPGA device names

2. After completing the New Project Wizard to create the project, in the main Quartus Prime window select Tools > Qsys, which leads to the window in Figure 4. This is the System Contents tab of the Qsys tool, which is used to add components to the system and configure the selected components to meet the design requirements. The available components are listed on the left side of the window.

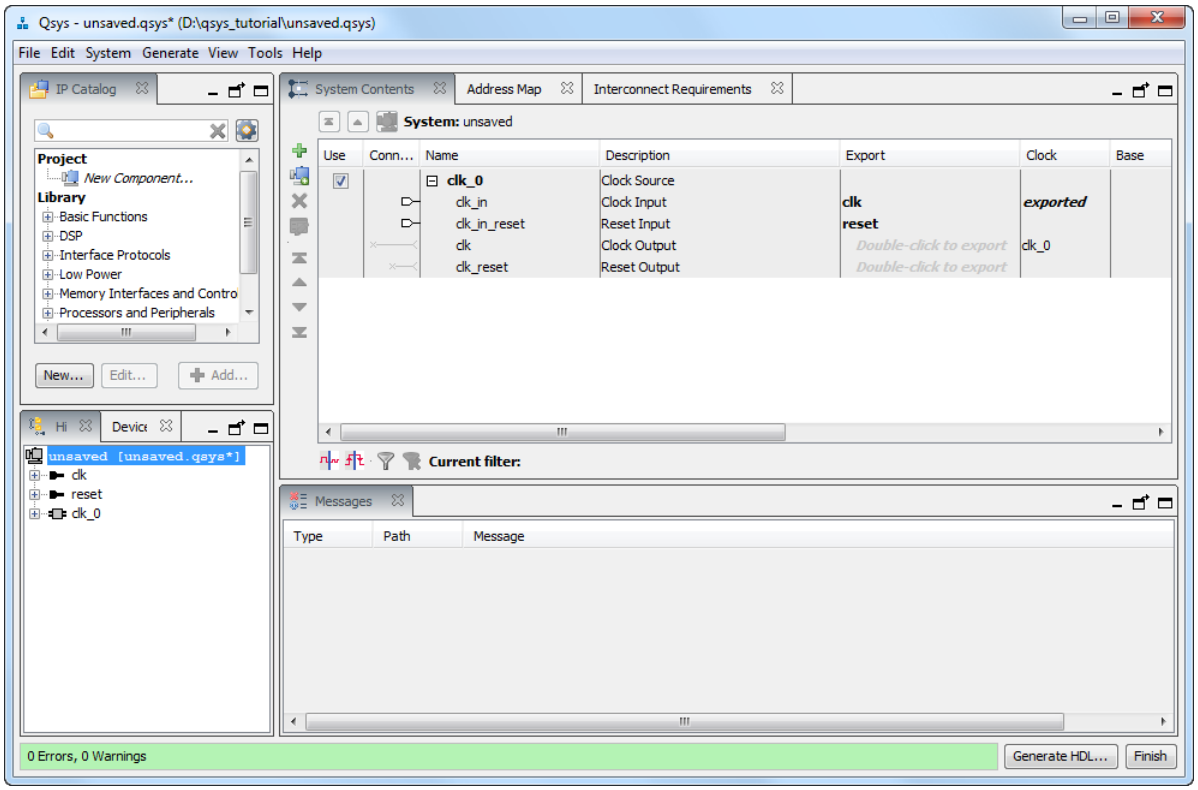

Figure 4. Create a new Nios II system.

3. The hardware system that will be generated using the Qsys tool runs under the control of a clock. For this tutorial we will make use of the 50-MHz clock that is provided on the DE1-SoC board. Your hardware system should contain a clock source called *clk_0*, whose frequency is 50-MHz. You can check that its frequency is indeed 50-MHz by double clicking the component, and checking the Clock frequency parameter of the component. If your system does not already contain *clk_0*, it is possible to add a clock source by selecting Basic Functions > Clocks; PLLs and Resets > Clock Source in the IP Catalog tab, then clicking Add....

4. Next, specify the processor as follows:

   - On the left side of the Qsys window expand Processors and Peripherals, select Embedded Processors > Nios II (Classic) Processor and click Add..., which leads to the window in Figure 5.

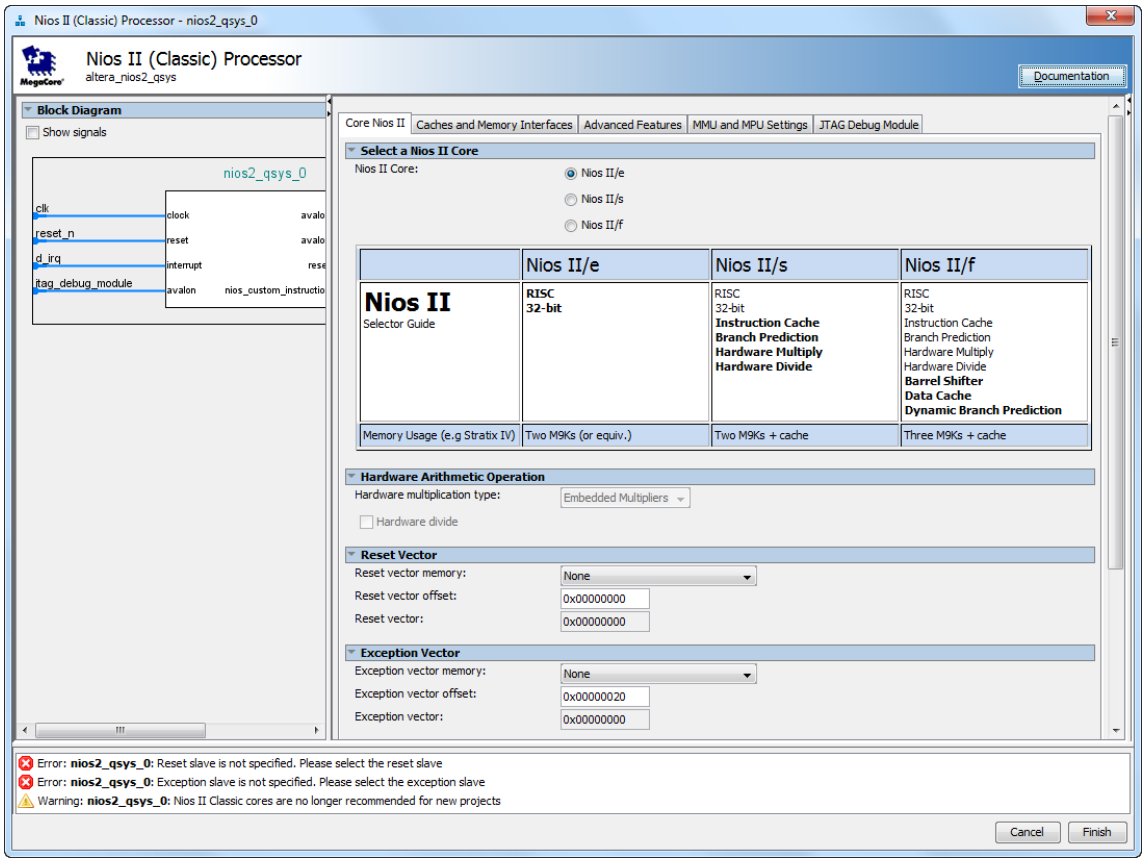

Figure 5. Create a Nios II processor.

- Choose Nios II/e which is the economy version of the processor. This version is available for use without a paid license. The Nios II processor has *reset* and *interrupt* inputs. When one of these inputs is activated, the processor starts executing the instructions stored at memory addresses known as *reset vector* and *interrupt vector*, respectively. Since we have not yet included any memory components in our design, the Qsys tool will display corresponding error messages. Ignore these messages as we will provide the necessary information later. You must disable the Include reset_req signal for OCI RAM and Multi-Cycle Custom Instructions setting in the Advanced Features tab to ensure proper operation. It is important that the checkbox is unchecked, otherwise you will experience issues when loading programs onto the system using the Altera Monitor Program. Click Finish to return to the main Qsys window, which now shows the Nios II processor specified as indicated in Figure 6.

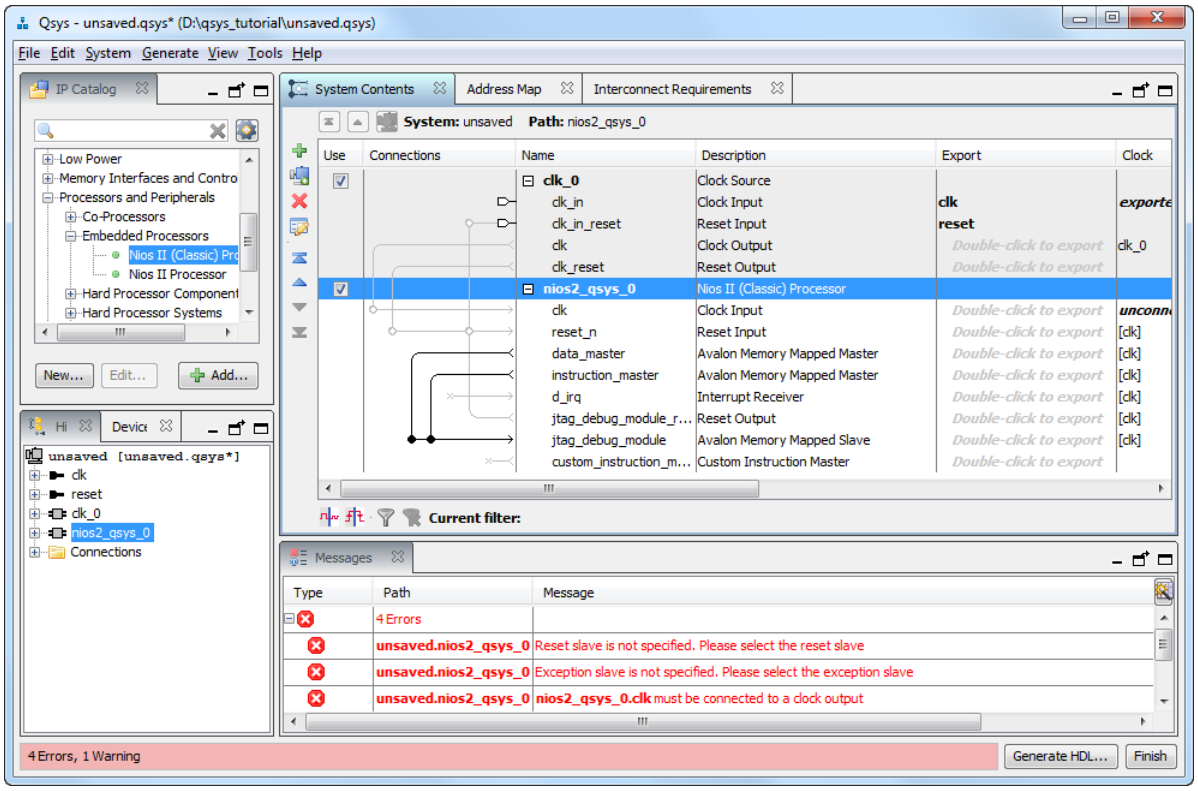

Figure 6. Inclusion of the Nios II processor in the design.

5. To specify the on-chip memory perform the following:

   - Expand the category Basic Functions, and then expand to select On Chip Memory > On-Chip Memory (RAM or ROM), and click Add
   - In the On-Chip Memory Configuration Wizard window, shown in Figure 7, ensure that the Data width is set to 32 bits and the Total memory size to 4K bytes (4096 bytes)
   - Do not change the other default settings
   - Click Finish, which returns to the System Contents tab as indicated in Figure 8

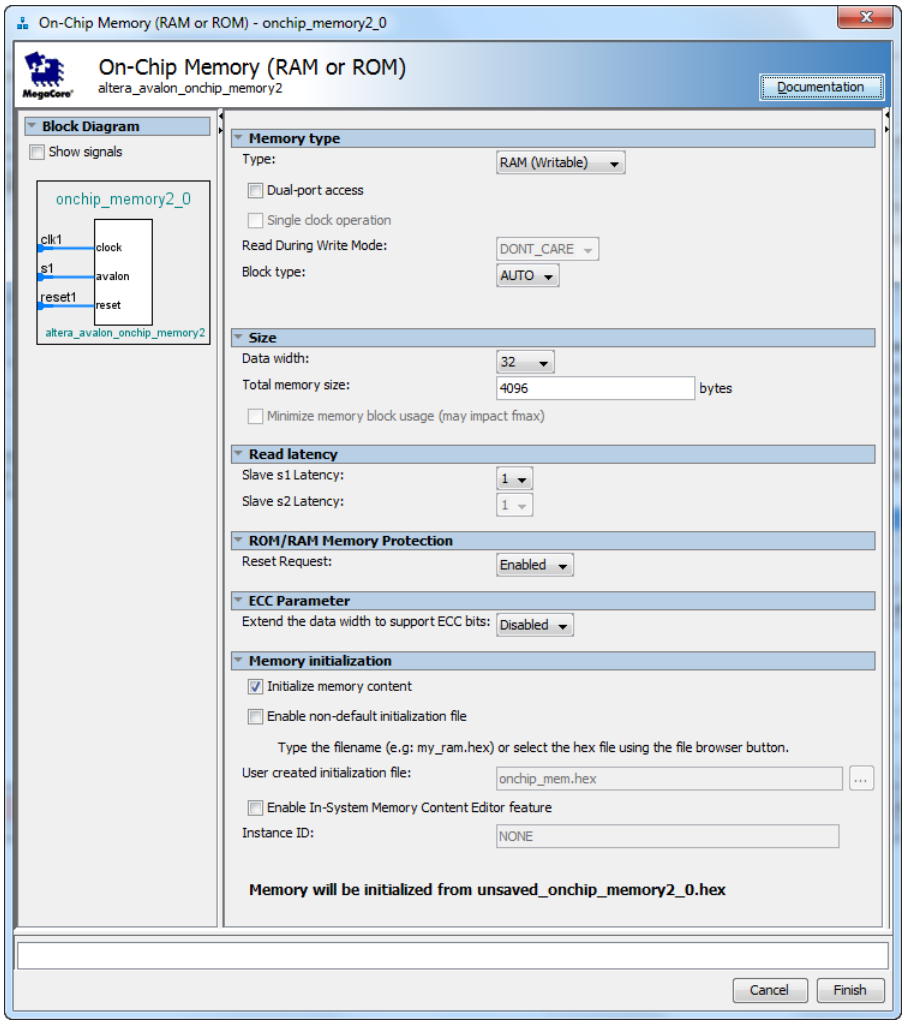

Figure 7. Define the on-chip memory.

6. Observe that while the Nios II processor and the on-chip memory have been included in the design, no connections between these components have been established. To specify the desired connections, examine the Connections area in the window in Figure 8. The connections already made are indicated by filled circles and the other possible connections by empty circles, as indicated in Figure 9.

   Clicking on an empty circle makes a connection. Clicking on a filled circle removes the connection.

   Make the following connections:

   - Clock inputs of the processor and the memory to the clock output of the clock component
   - Reset inputs of the processor and the memory to both the reset output of the clock component and the *jtag_debug_module_reset* output
   - The *s1* input of the memory to both the *data_master* and *instruction_master* outputs of the processor

   The resulting connections are shown in Figure 10.

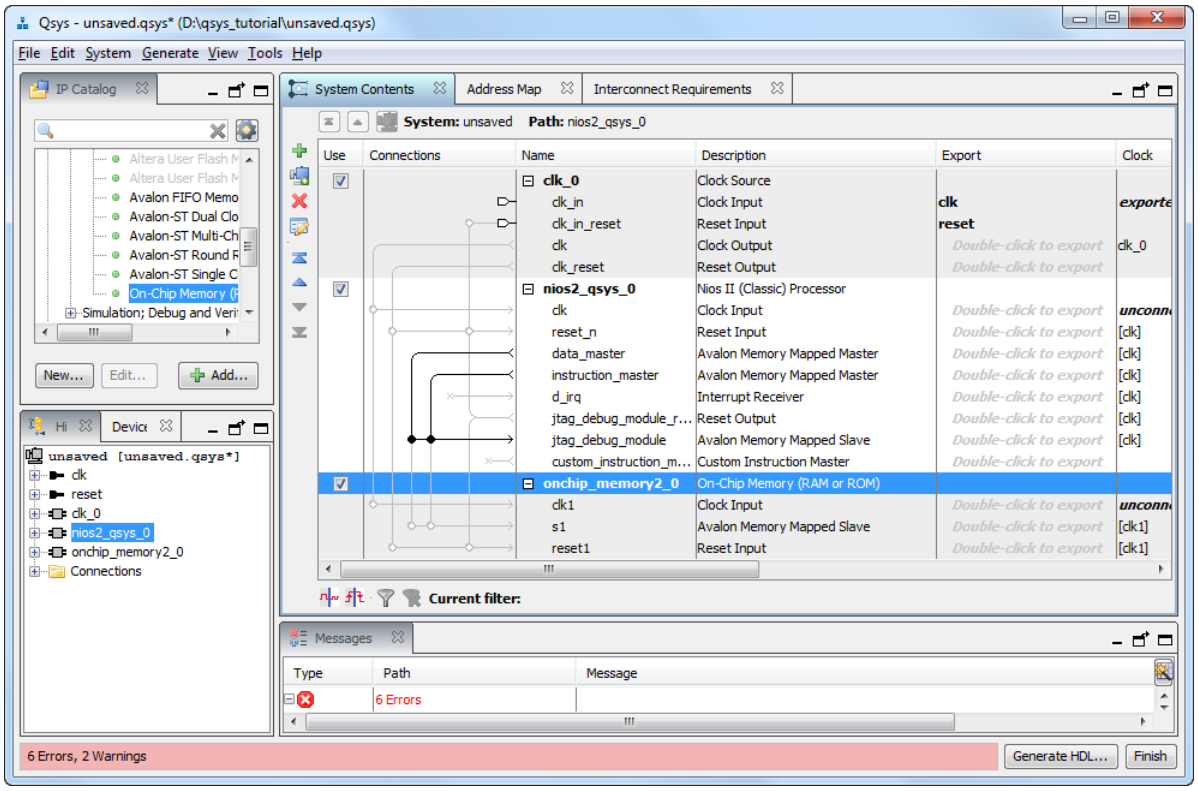

Figure 8. The on-chip memory included on a DE-series board.

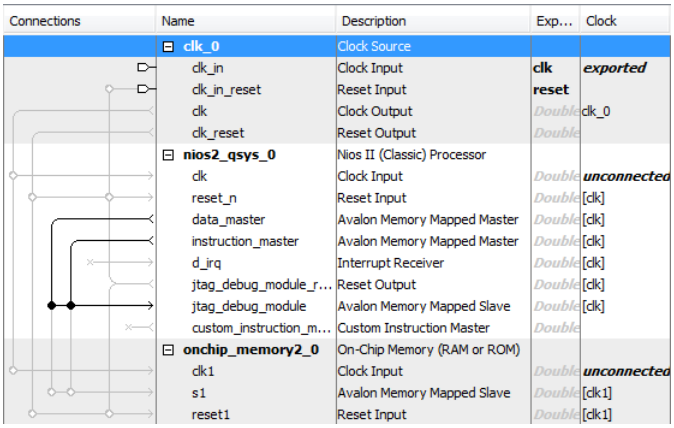

Figure 9. Connections that can be made.

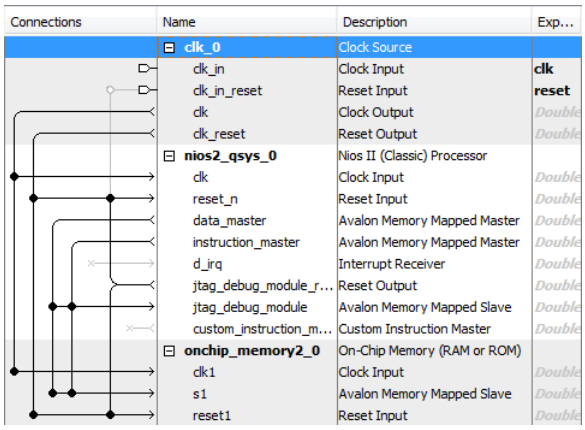

Figure 10. The connections that are now established.

7. Specify the input parallel I/O interface as follows:
   - Select Processors and Peripherals > Peripherals > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard in Figure 11
   - Specify the width of the port to be 8 bits and choose the direction of the port to be Input, as shown in the figure.
   - Click Finish.

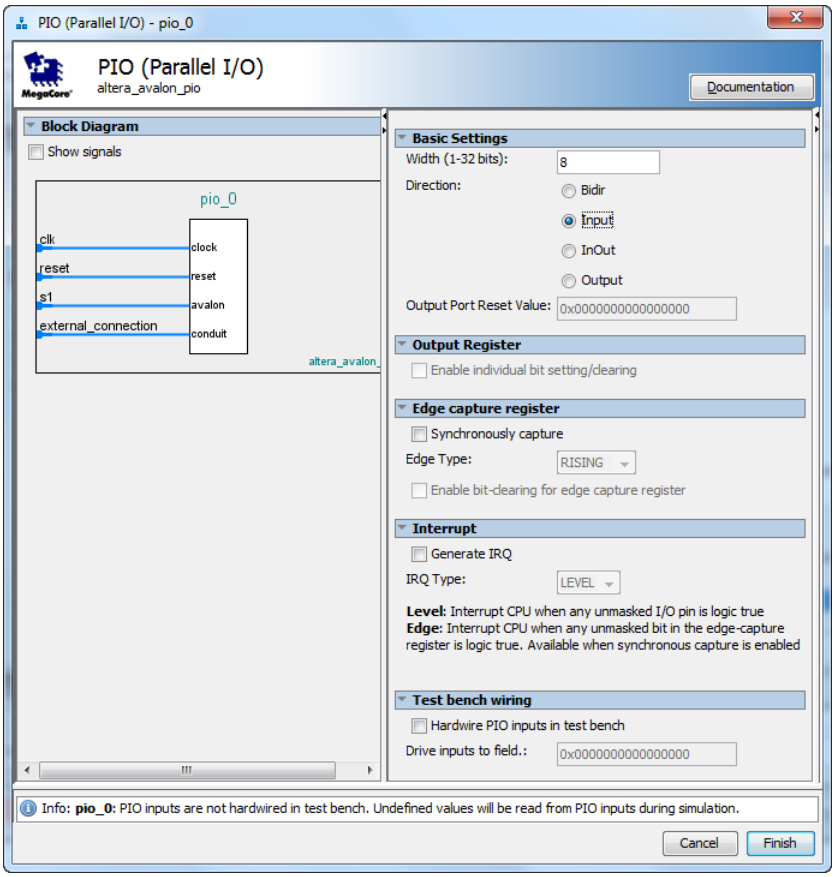

Figure 11. Define a parallel input interface.

8. In the same way, specify the output parallel I/O interface:
   - Select Processors and Peripherals > Peripherals > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard again
   - Specify the width of the port to be 8 bits and choose the direction of the port to be Output.
   - Click Finish to return to the System Contents tab
9. Specify the necessary connections for the two PIOs:
   - Clock input of the PIO to the clock output of the clock component
   - Reset input of the PIO to the reset output of the clock component and the *jtag_debug_module_reset* output
   - The *s1* input of the PIO the *data_master* output of the processor

   The resulting design is depicted in Figure 12.

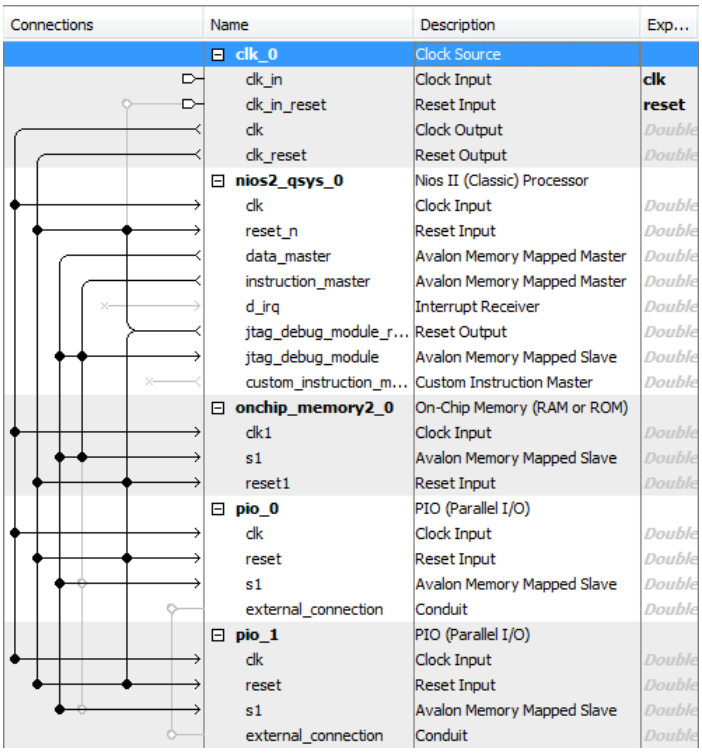

Figure 12. The system with all components and connections.

10. We wish to connect to a host computer and provide a means for communication between the Nios II system and the host computer. This can be accomplished by instantiating the JTAG UART interface as follows:

    - Select Interface Protocols > Serial > JTAG UART and click Add to reach the JTAG UART Configuration Wizard in Figure 13
    - Do not change the default settings
    - Click Finish to return to the System Contents tab

    Connect the JTAG UART to the clock, reset and data-master ports, as was done for the PIOs. Connect the Interrupt Request (IRQ) line from the JTAG UART to the Nios II processor by selecting the connection under the IRQ column, as shown in Figure 14. Once the connection is made, a box with the number 0 inside will appear on the connection. The Nios II processor has 32 interrupt ports ranging from 0 to 31, and the number in this box selects which port will be used for this IRQ. Click on the box and change it to use port 5. Make sure the irq port of JTAG UART gets automatically connected to the d_irq port of Nios II Processor.

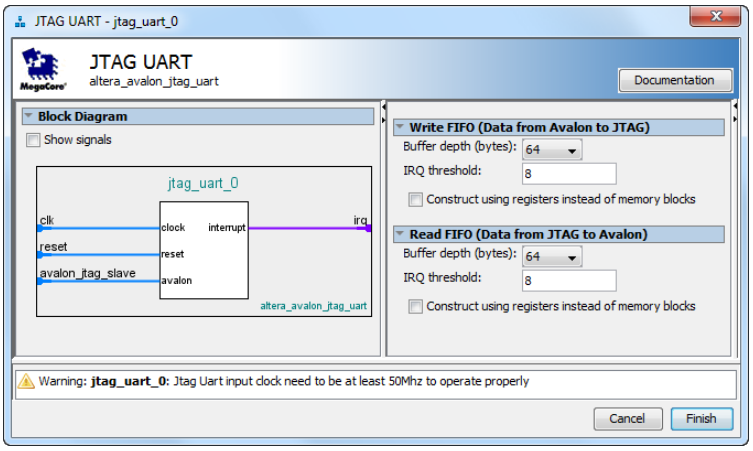

Figure 13. Define the JTAG UART interface.

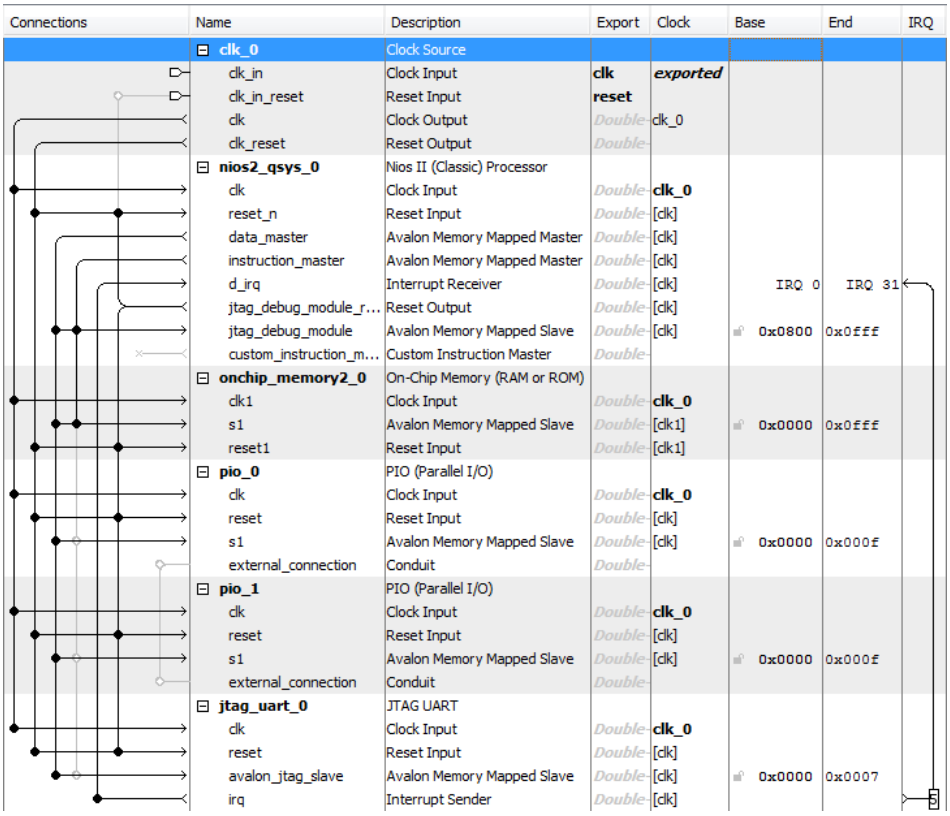

Figure 14. Connect the IRQ line from the JTAG UART to the Nios II processor.

11. Note that the Qsys tool automatically chooses names for the various components. The names are not necessarily descriptive enough to be easily associated with the target design, but they can be changed. In Figure 2,

we use the names Switches and LEDs for the parallel input and output interfaces, respectively. These names can be used in the implemented system. Right-click on the pio_0 name and then select Rename. Change the name to *switches*. Similarly, change pio_1 to *LEDs*. Figure 15 shows the system with name changes that we made for all components.

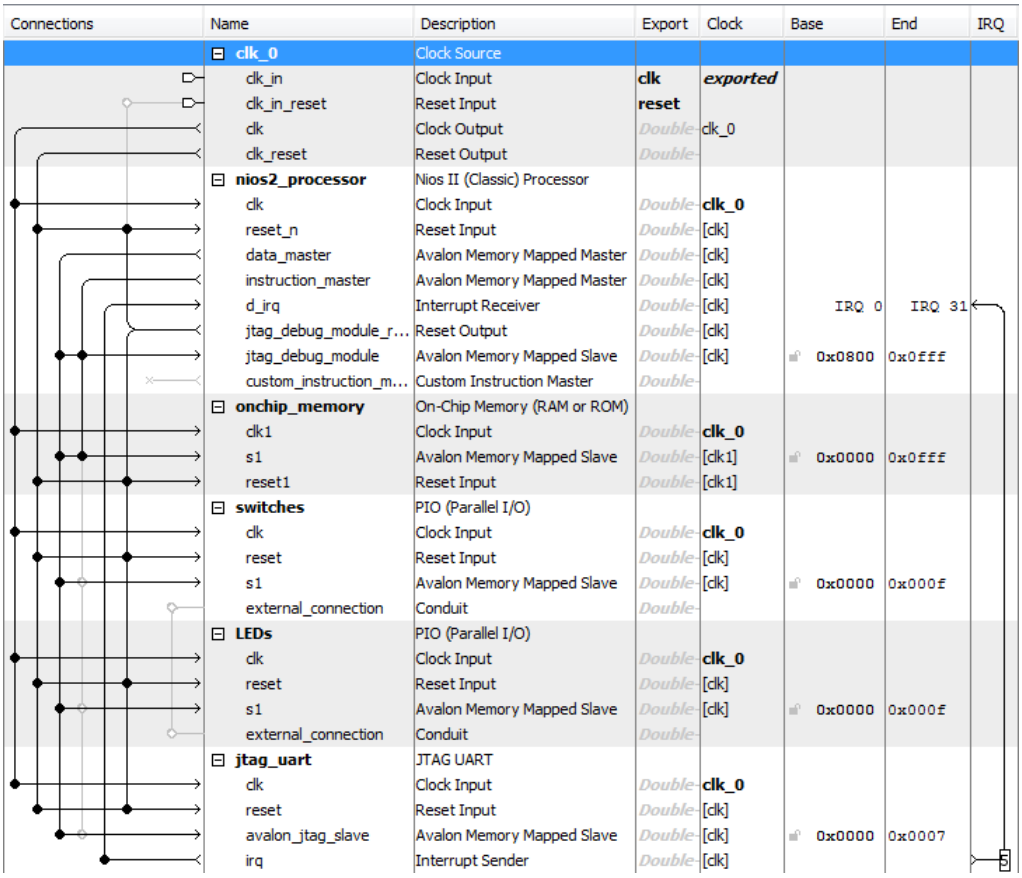

Figure 15. The system with all components appropriately named.

12. Observe that the base and end addresses of the various components in the designed system have not been properly assigned. These addresses can be assigned by the user, but they can also be assigned automatically by the Qsys tool. We will choose the latter possibility. However, we want to make sure that the on-chip memory has the base address of zero. Double-click on the Base address for the on-chip memory in the Qsys window and enter the address 0x00000000. Then, lock this address by clicking on the adjacent lock symbol. Now, let Qsys assign the rest of the addresses by selecting System > Assign Base Addresses (at the top of the window), which produces an assignment similar to that shown in Figure 16.

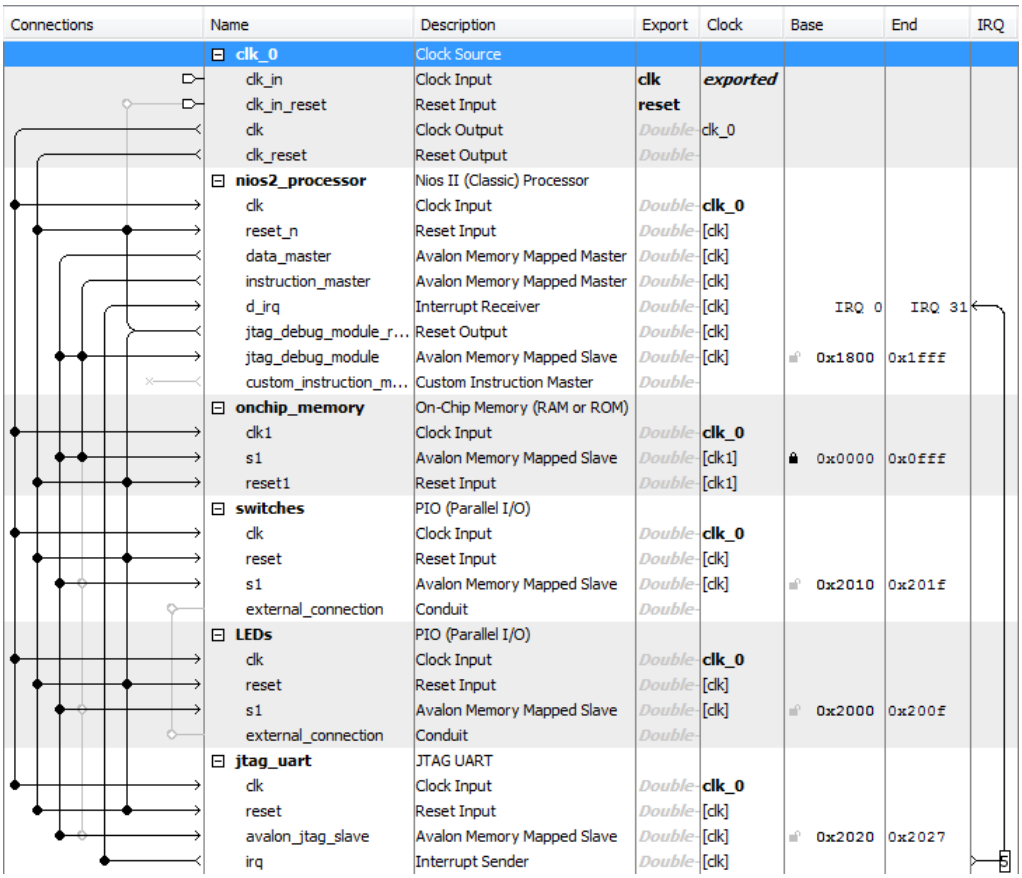

Figure 16. The system with assigned addresses.

13. The behavior of the Nios II processor when it is reset is defined by its reset vector. It is the location in the memory device from which the processor fetches the next instruction when it is reset. Similarly, the exception vector is the memory address of the instruction that the processor executes when an interrupt is raised. To specify these two parameters, perform the following:

    - Right-click on the *nios2_processor* component in the window displayed in Figure 16, and then select Edit to reach the window in Figure 17
    - Select *onchip_memory.s1* to be the memory device for both reset and exception vectors, as shown in Figure 17
    - Do not change the default settings for offsets
    - Observe that the error messages dealing with memory assignments shown in Figure 5 will now disappear
    - Click Finish to return to the System Contents tab

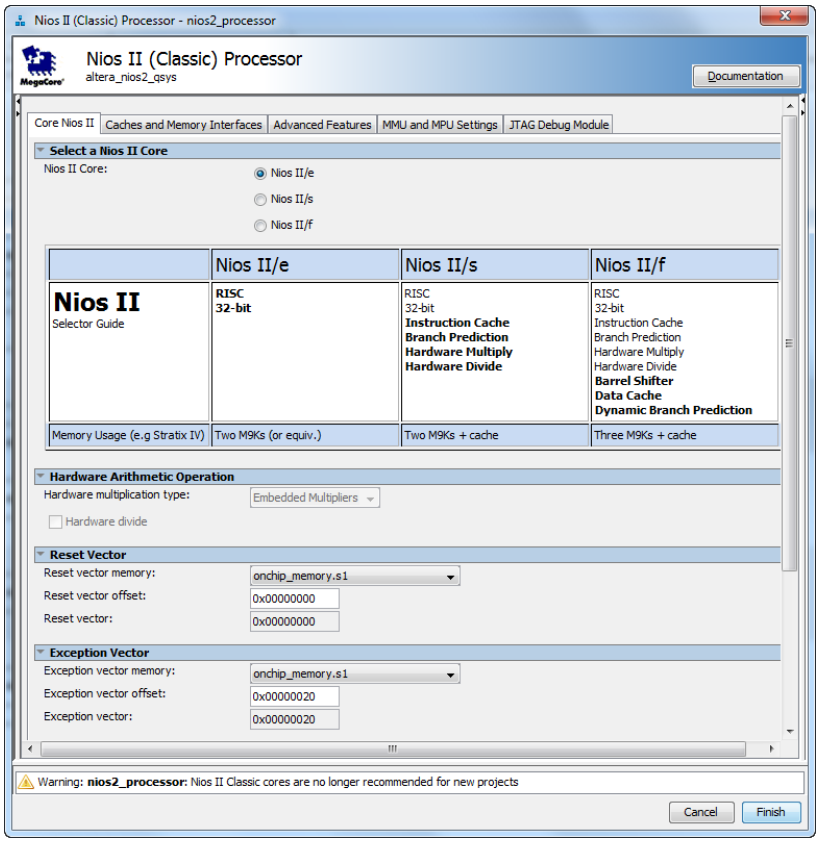

Figure 17. Define the reset and exception vectors.

14. So far, we have specified all connections inside our *nios_system* circuit. It is also necessary to specify connections to external components, which are switches and LEDs in our case. To accomplish this, double click on Double-click to export (in the Export column of the System Contents tab) for external_connection of the switches PIO, and type the name *switches*. Similarly, establish the external connection for the lights, called *leds*. This completes the specification of our *nios_system*, which is depicted in Figure 18.

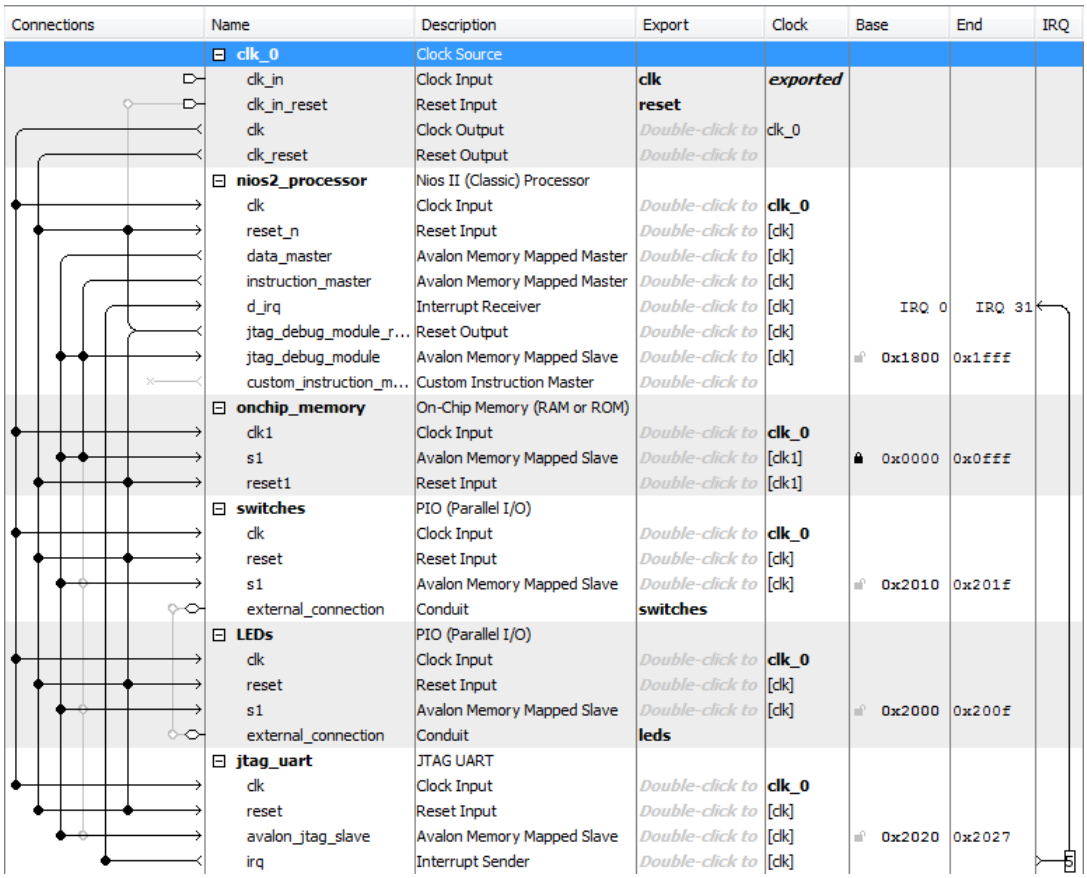

Figure 18. The complete system.

15. Having specified all components needed to implement the desired system, it can now be generated. Save the specified system; we used the name *nios_system*. Then, select Generate > Generate HDL, which leads to the window in Figure 19. Select None for the option Simulation > Create simulation model, because in this tutorial we will not deal with the simulation of hardware. Click Generate on the bottom of the window. When successfully completed, the generation process produces the message “Generate Completed".

    Exit the Qsys tool to return to the main Quartus Prime window.

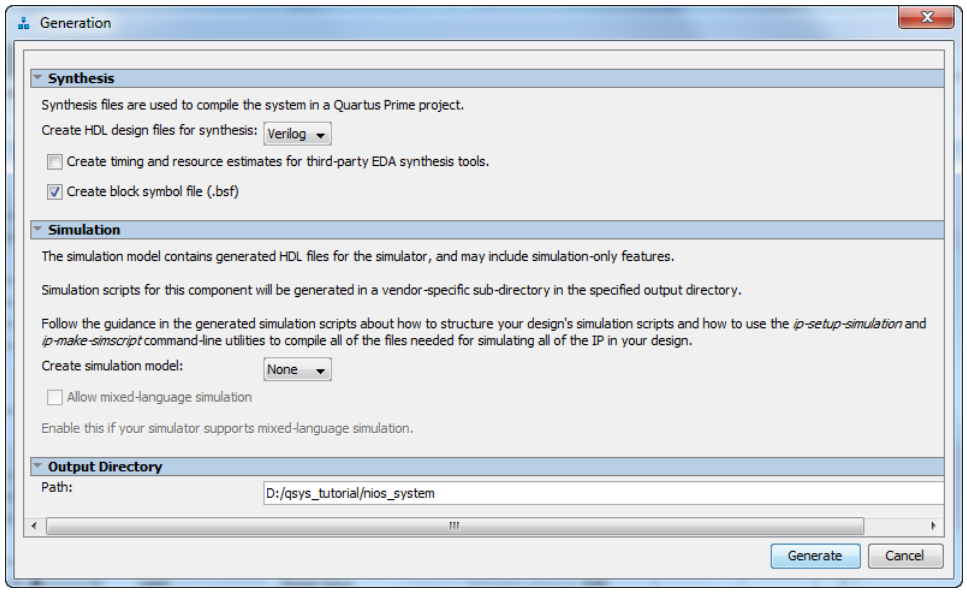

Figure 19. Generation of the system.

Changes to the designed system are easily made at any time by reopening the Qsys tool. Any component in the System Contents tab of the Qsys tool can be selected and edited or deleted, or a new component can be added and the system regenerated.

5 Integration of the Nios II System into a Quartus Prime Project

To complete the hardware design, we have to perform the following:

- Instantiate the module generated by the Qsys tool into the Quartus Prime project
- Assign the FPGA pins
- Compile the designed circuit
- Program and configure the FPGA device on the DE1-SoC board

5.1 Instantiation of the Module Generated by the Qsys Tool

The Qsys tool generates a Verilog module that defines the desired Nios II system. In our design, this module will have been generated in the *nios_system.v* file, which can be found in the directory *qsys_tutorial/nios_system/synthesis* of the project. The Qsys tool generates Verilog modules, which can then be used in designs specified using either Verilog or VHDL languages.

Normally, the Nios II module generated by the Qsys tool is likely to be a part of a larger design. However, in the case of our simple example there is no other circuitry needed. All we need to do is instantiate the Nios II system in

our top-level Verilog or VHDL module, and connect inputs and outputs of the parallel I/O ports, as well as the clock and reset inputs, to the appropriate pins on the FPGA device.

The Verilog code in the *nios_system.v* file is quite large. Figure 20 depicts the portion of the code that defines the input and output ports for the module *nios_system*. The 8-bit vector that is the input to the parallel port *switches* is called *switches_export*. The 8-bit output vector is called *leds_export*. The clock and reset signals are called *clk_clk* and *reset_reset_n*, respectively. Note that the reset signal was added automatically by the Qsys tool; it is called *reset_reset_n* because it is active low.

```
module nios_system (
                input  wire       clk_clk,         //      clk.clk
                output wire [7:0] leds_export,     //     leds.export
                input  wire       reset_reset_n,   //    reset.reset_n
                input  wire [7:0] switches_export  // switches.export
        );
```

Figure 20. A part of the generated Verilog module.

The *nios_system* module has to be instantiated in a top-level module that has to be named *lights*, because this is the name we specified in Figure 3 for the top-level design entity in our Quartus Prime project. For the input and output ports of the *lights* module we have used the pin names that are specified in the DE1-SoC User Manual: *CLOCK_50* for the 50-MHz clock, *KEY* for the pushbutton switches, *SW* for the slider switches, and *LEDR* for the red LEDs. Using these names simplifies the task of creating the needed pin assignments.

5.1.1 Instantiation in a Verilog Module

Figure 21 shows a top-level Verilog module that instantiates the Nios II system. If using Verilog for the tutorial, type this code into a file called *lights.v*, or use the file provided with this tutorial.

```
// Implements a simple Nios II system for the DE-series board.
// Inputs:     SW7−0 are parallel port inputs to the Nios II system
//             CLOCK_50 is the system clock
//             KEY0 is the active-low system reset
// Outputs:    LEDR7−0 are parallel port outputs from the Nios II system
module  lights (CLOCK_50, SW, KEY, LEDR);
    input  CLOCK_50;
    input  [7:0]  SW;
    input  [0:0]  KEY;
    output  [7:0]  LEDR;
// Instantiate the Nios II system module generated by the Qsys tool:
    nios_system  NiosII  (
        .clk_clk(CLOCK_50),
        .reset_reset_n(KEY),
        .switches_export(SW),
        .leds_export(LEDR));
endmodule
```

Figure 21. Instantiating the Nios II system using Verilog code.

5.1.2 Instantiation in a VHDL Module

Figure 22 shows a top-level VHDL module that instantiates the Nios II system. If using VHDL for the tutorial, type this code into a file called *lights.vhd*, or use the file provided with this tutorial.

```
-- Implements a simple Nios II system for the DE-series board.
-- Inputs:     SW7-0 are parallel port inputs to the Nios II system
--             CLOCK_50 is the system clock
--             KEY0 is the active-low system reset
-- Outputs:    LEDR7-0 are parallel port outputs from the Nios II system

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY lights IS

PORT (
    CLOCK_50  : IN STD_LOGIC;
    KEY       : IN STD_LOGIC_VECTOR (0 DOWNTO 0);
    SW        : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    LEDR      : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END lights;

ARCHITECTURE lights_rtl OF lights IS
    COMPONENT nios_system
        PORT (
            SIGNAL clk_clk: IN STD_LOGIC;
            SIGNAL reset_reset_n : IN STD_LOGIC;
            SIGNAL switches_export : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            SIGNAL leds_export : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
            );
    END COMPONENT;
BEGIN
NiosII : nios_system
    PORT MAP(
        clk_clk => CLOCK_50,
        reset_reset_n => KEY(0),
        switches_export => SW(7 DOWNTO 0),
        leds_export => LEDR(7 DOWNTO 0)
    );
END lights_rtl;
```

Figure 22. Instantiating the Nios II system using VHDL code.

6 Compiling the Quartus Prime Project

Add the *lights.v/vhd* file to your Quartus Prime project. Also, add the necessary pin assignments for the DE-series board to your project. The procedure for making pin assignments is described in the tutorial *Quartus Prime Introduction Using Verilog/VHDL Designs*. Note that an easy way of making the pin assignments when we use the same pin names as in the DE1-SoC User Manual is to import the assignments from a Quartus Prime Setting File with Pin Assignments. For example, the pin assignments for the DE1-SoC board are provided in the *DE1_SoC.qsf* file, which can be found on Altera's University Program website.

Since the system we are designing needs to operate at a 50-MHz clock frequency, we can add the needed timing assignment in the Quartus Prime project. The tutorial *Using TimeQuest Timing Analyzer* shows how this is done. However, for our simple design, we can rely on the default timing assignment that the Quartus Prime compiler assumes in the absence of a specific specification. The compiler assumes that the circuit has to be able to operate at a clock frequency of 1 GHz, and will produce an implementation that either meets this requirement or comes as close to it as possible.

Finally, before compiling the project, it is necessary to add the *nios_system.qip* file (IP Variation file) in the directory *qsys_tutorial/nios_system/synthesis* to your Quartus Prime project. Then, compile the project. You may see some warning messages associated with the Nios II system, such as some signals being unused or having wrong bit-lengths of vectors; these warnings can be ignored.

7 Using the Altera Monitor Program to Download the Designed Circuit and Run an Application Program

The designed circuit has to be downloaded into the FPGA device on a DE-series board. This can be done by using the Programmer Tool in the Quartus Prime software. However, we will use a simpler approach by using the Altera Monitor Program, which provides a simple means for downloading the circuit into the FPGA as well as running the application programs.

A parallel I/O interface generated by the Qsys tool is accessible by means of registers in the interface. Depending on how the PIO is configured, there may be as many as four registers. One of these registers is called the Data register. In a PIO configured as an input interface, the data read from the Data register is the data currently present on the PIO input lines. In a PIO configured as an output interface, the data written (by the Nios II processor) into the Data register drives the PIO output lines. If a PIO is configured as a bidirectional interface, then the PIO inputs and outputs use the same physical lines. In this case there is a Data Direction register included, which determines the direction of the input/output transfer. In our unidirectional PIOs, it is only necessary to have the Data register. The addresses assigned by the Qsys tool are 0x00002010 for the Data register in the PIO called *switches* and 0x00002000 for the Data register in the PIO called *LEDs*, as indicated in Figure 16.

Our application task is very simple. A pattern selected by the current setting of slider switches has to be displayed on the LEDs. We will show how this can be done in both Nios II assembly language and C programming language.

7.1 A Nios II Assembly Language Program

Figure 23 gives a Nios II assembly-language program that implements our task. The program loads the addresses of the Data registers in the two PIOs into processor registers $r2$ and $r3$. It then has an infinite loop that merely transfers the data from the input PIO, *switches*, to the output PIO, *leds*.

```
.equ      switches, 0x00002010
.equ      leds, 0x00002000
.global   _start
_start:   movia    r2, switches
          movia    r3, leds
LOOP:     ldbio    r4, 0(r2)
          stbio    r4, 0(r3)
          br       LOOP
.end
```

Figure 23. Assembly-language code to control the lights.

The directive .global _start indicates to the Assembler that the label *_start* is accessible outside the assembled object file. This label is the default label we use to indicate to the Linker program the beginning of the application program.

For a detailed explanation of the Nios II assembly language instructions see the tutorial *Introduction to the Altera Nios II Soft Processor*, which is available on Altera's University Program website.

Enter this code into a file *lights.s*, or use the file provided with this tutorial, and place the file into a working directory. We placed the file into the directory *qsys_tutorial\app_software*.

7.2 A C-Language Program

An application program written in the C language can be handled in the same way as the assembly-language program. A C program that implements our simple task is given in Figure 24. Enter this code into a file called *lights.c*, or use the file provided with this tutorial, and place the file into a working directory.

```
#define switches (volatile char *) 0x0002010
#define leds (char *) 0x0002000
void main()
{         while (1)
          *leds = *switches;
}
```

Figure 24. C-language code to control the lights.

7.3 Using the Altera Monitor Program

The Altera University Program provides the *monitor* software, called *Altera Monitor Program*, for use with the DE-series boards. This software provides a simple means for compiling, assembling and downloading of programs onto a DE-series board. It also makes it possible for the user to perform debugging tasks. A description of this software is available in the *Altera Monitor Program* tutorial. We should also note that other Nios II development systems are provided by Altera, for use in commercial development. Although we will use the Altera Monitor Program in this tutorial, the other Nios II tools available from Altera could alternatively be used with our designed hardware system.

Open the Altera Monitor Program, which leads to the window in Figure 25.

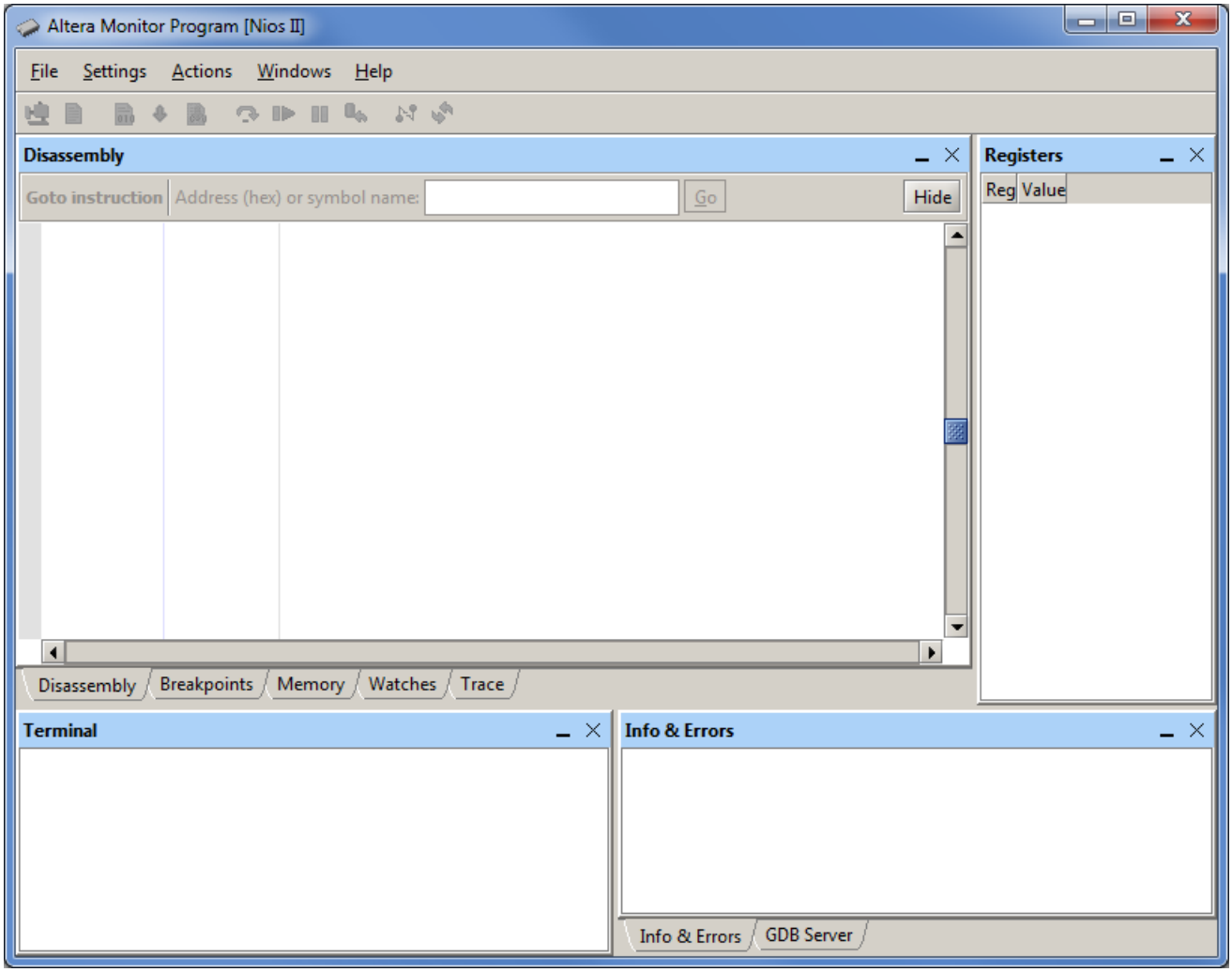

Figure 25. The Altera Monitor Program main window.

The monitor program needs to know the characteristics of the designed Nios II system, which are given in the file *nios_system.qsys*. Click the File > New Project menu item to display the New Project Wizard window, shown in Figure 26, and perform the following steps:

1. Enter the *qsys_tutorial\app_software* directory as the Project directory by typing it directly into the Project directory field, or by browsing to it using the Browse... button.
2. Enter *lights_example* (or some other name) as the Project name
3. Select Nios II as the Architecture and click Next, leading to Figure 27.

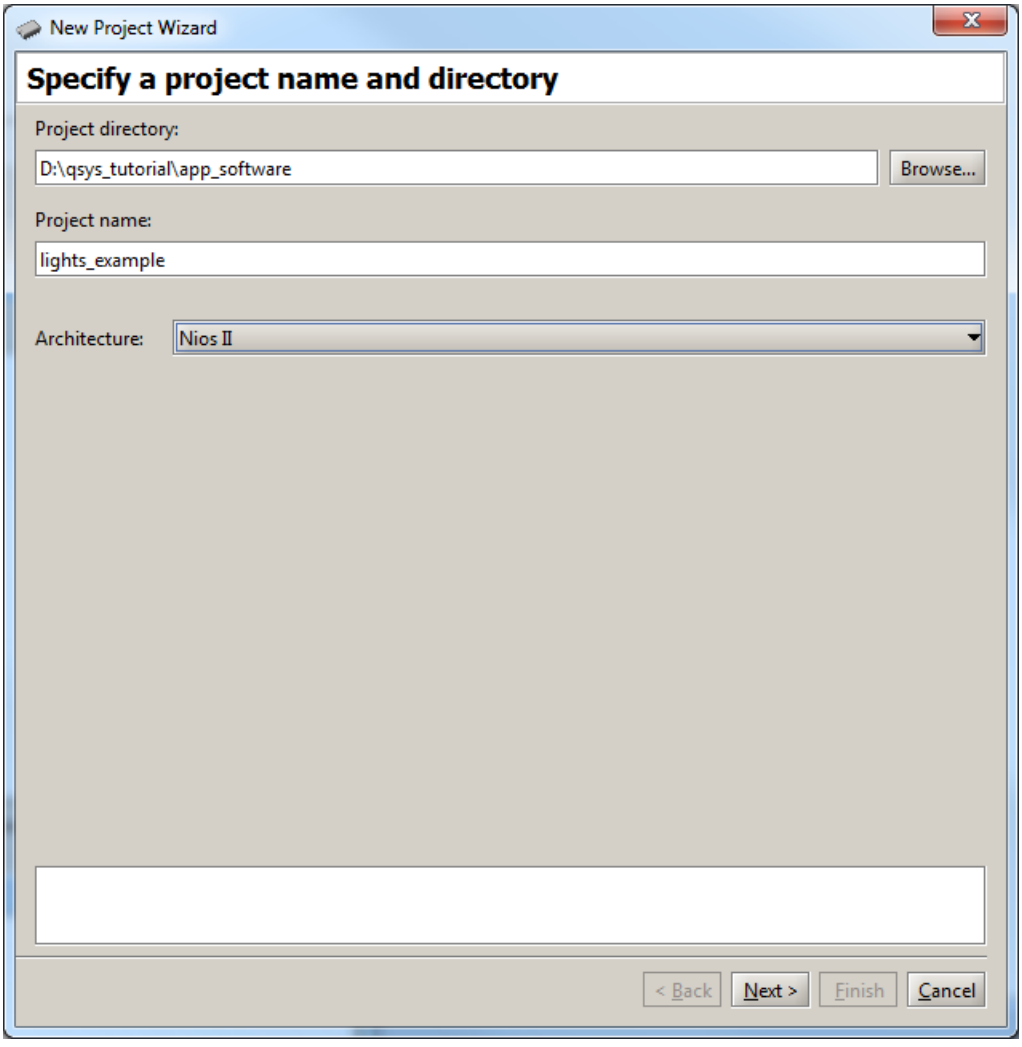

Figure 26. Specify the project directory and name.

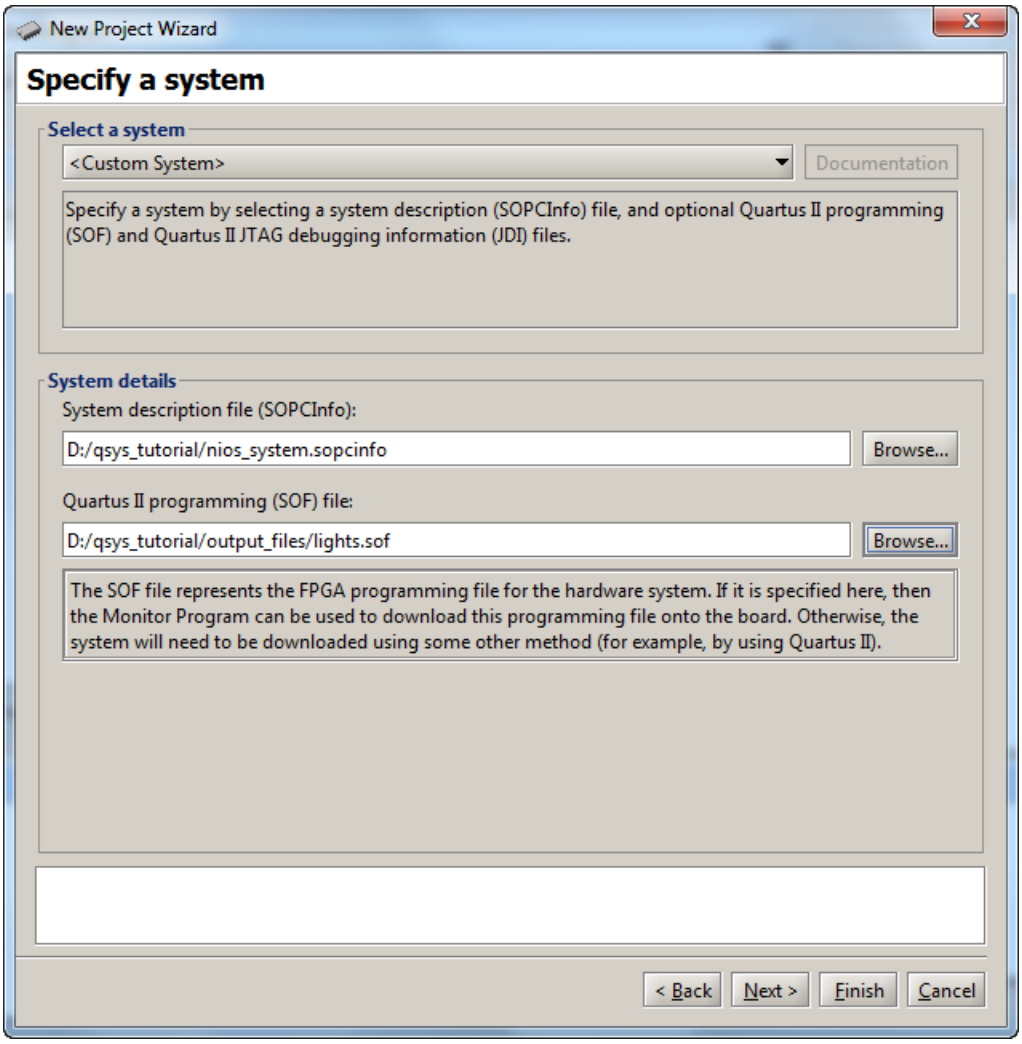

Figure 27. The System Specification window.

4. From the Select a System drop-down box select Custom System, which specifies that you wish to use the hardware that you designed.

   Click Browse... beside the System description field to display a file selection window and choose the *nios_system.sopcinfo* file. Note that this file is in the design directory *qsys_tutorial*.

   Select the *lights.sof* file in the Quartus Prime programming (SOF) file field, which provides the information needed to download the designed system into the FPGA device on the DE-series board. Click Next, which leads to the window in Figure 28.

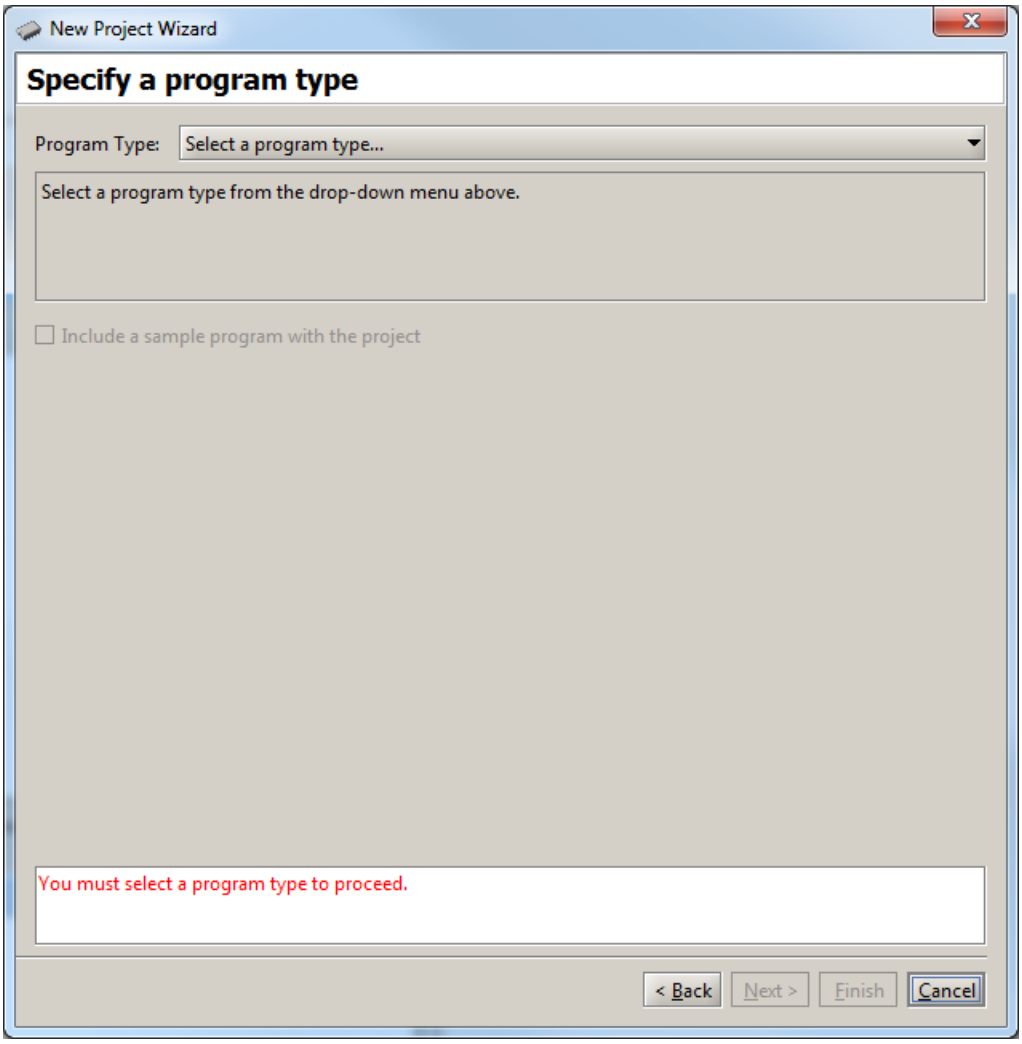

Figure 28. Specification of the program type.

5. If you wish to use a Nios II assembly-language application program, select Assembly Program as the program type from the drop-down menu. If you wish to use a C-language program, select C Program. Click Next, leading to Figure 29.

6. Click Add... to display a file selection window and choose the *lights.s* file, or *lights.c* for a C program, and click Select. We placed the application-software files in the directory *qsys_tutorial\app_software*. Upon returning to the window in Figure 29, click Next.

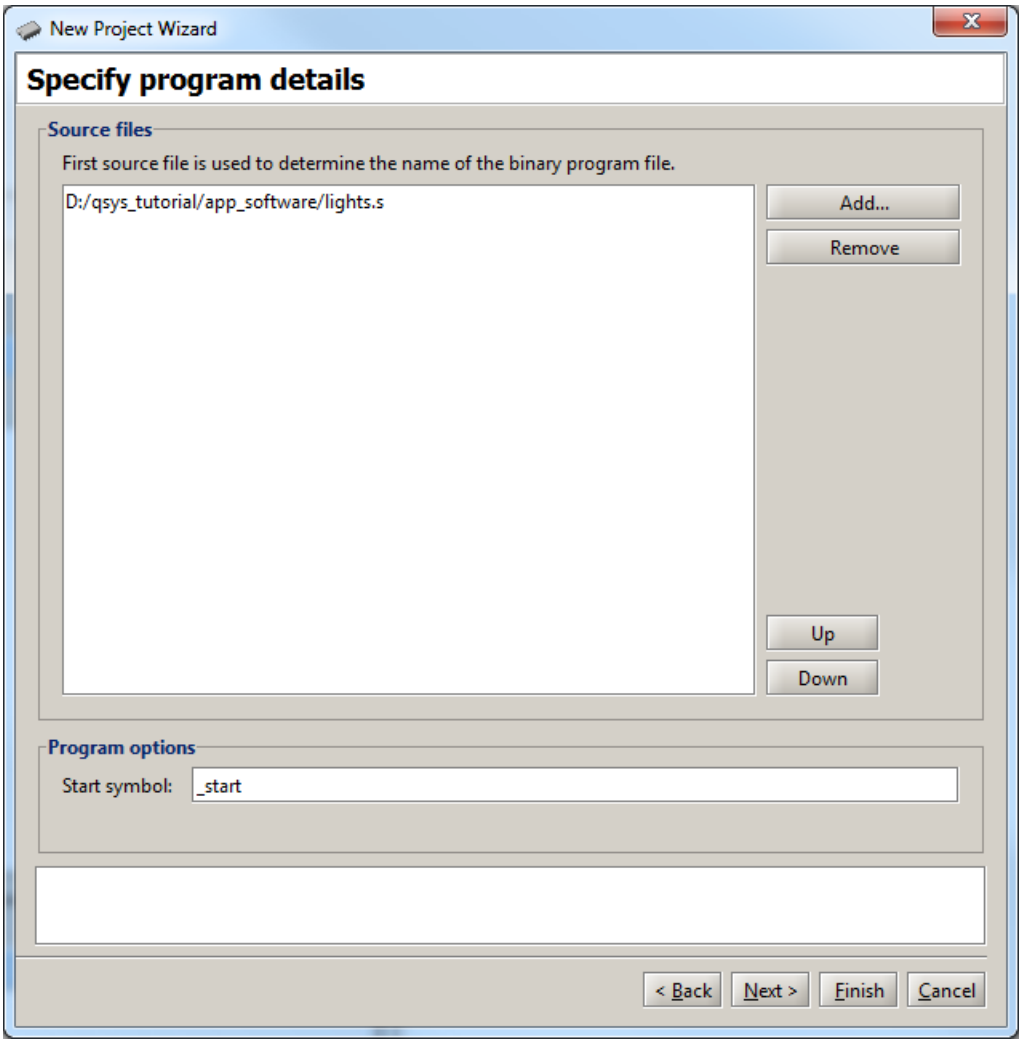

Figure 29. Specify the application program to use.

7. In the window in Figure 30, ensure that the Host Connection is set to *USB-Blaster*, the Processor is set to *nios2_processor* and the Terminal Device is set to *jtag_uart*. Click Next.

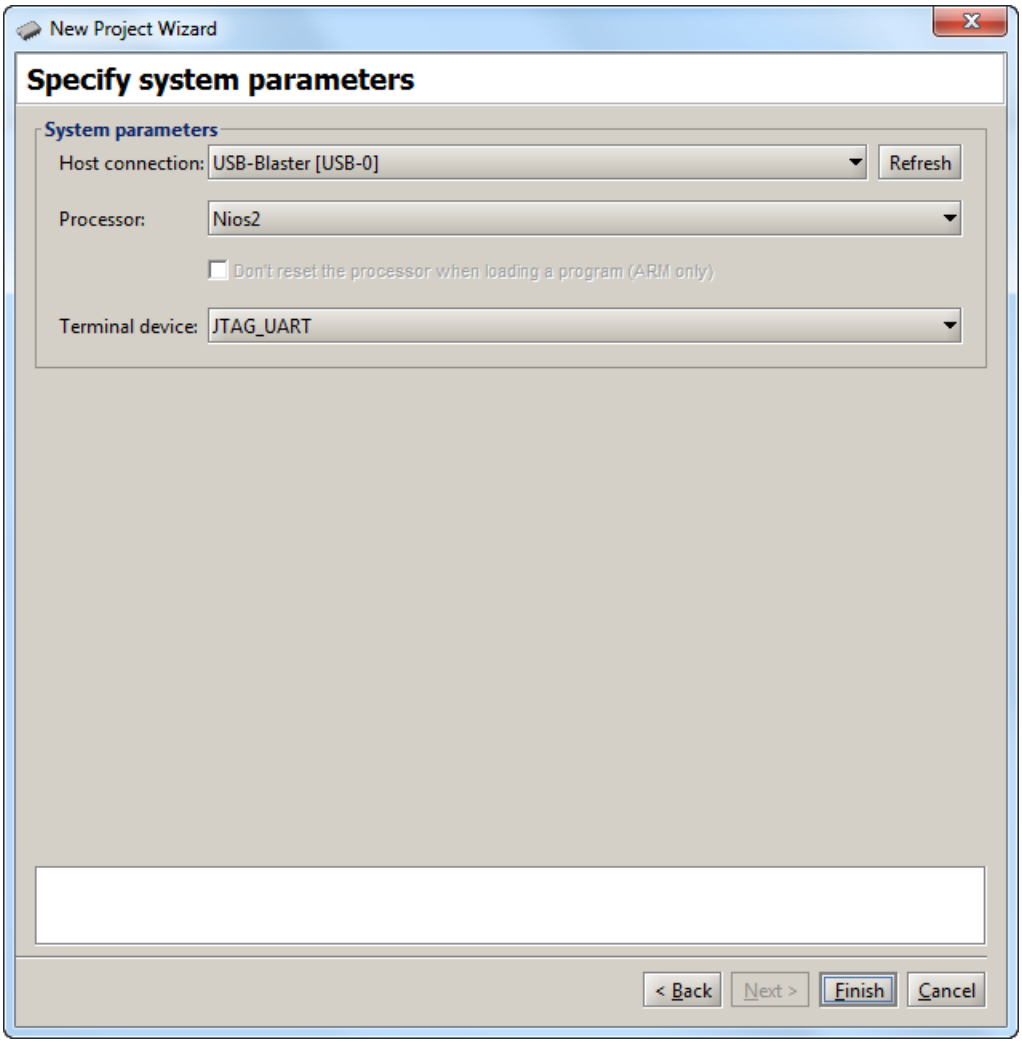

Figure 30. Specify the system parameters.

8. The Monitor Program also needs to know where to load the application program. In our case, this is the memory block in the FPGA device. The name assigned to this memory is *onchip_memory*. Since there is no other memory in our design, the Monitor Program will select this memory by default, as shown in Figure 31.

   Having provided the necessary information, click Finish to confirm the system configuration. When a pop-up box asks you if you want to have your system downloaded onto the DE-series board click Yes.

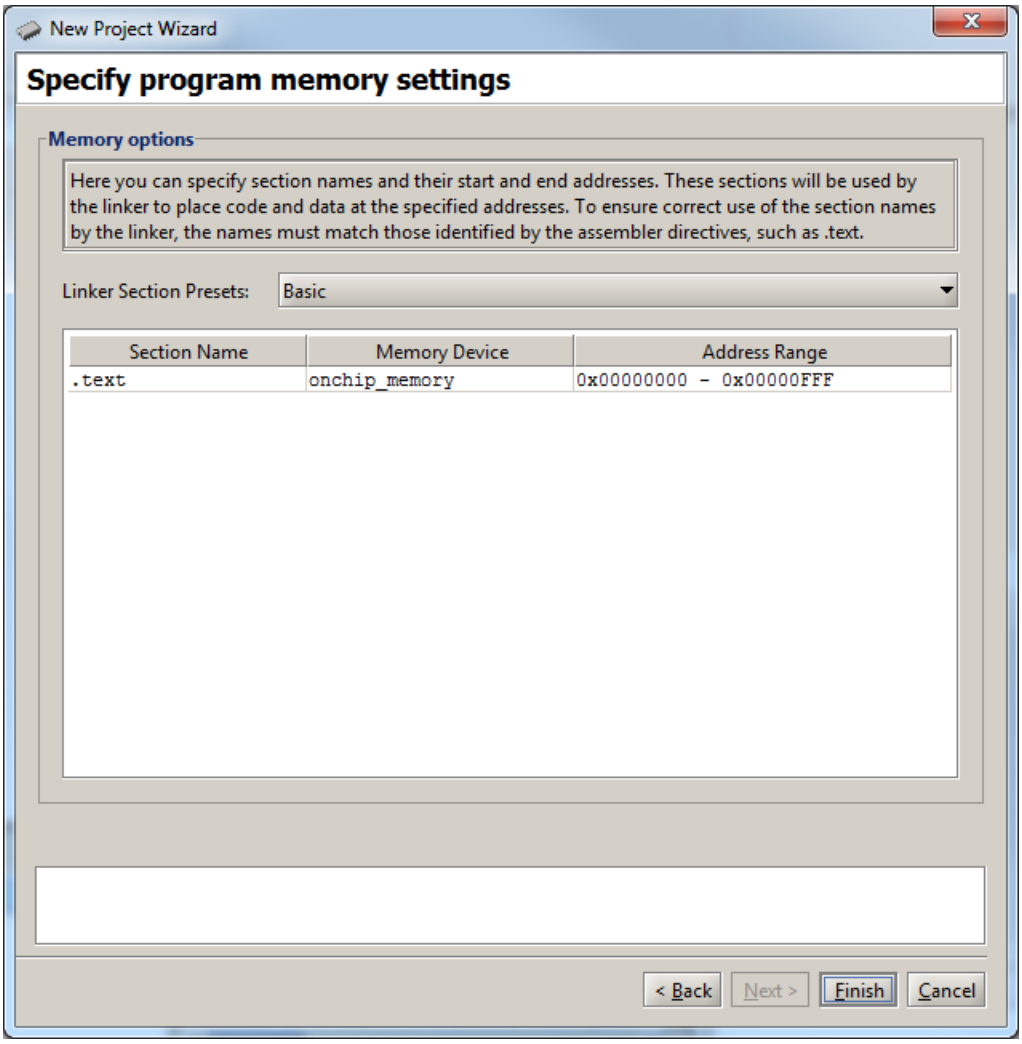

Figure 31. Specify where the program will be loaded in the memory.

9. Now, in the monitor window in Figure 25 select Actions > Compile & Load to assemble and download your program.

10. The downloaded program is shown in Figure 32. Run the program and verify the correctness of the designed system by setting the slider switches to a few different patterns.

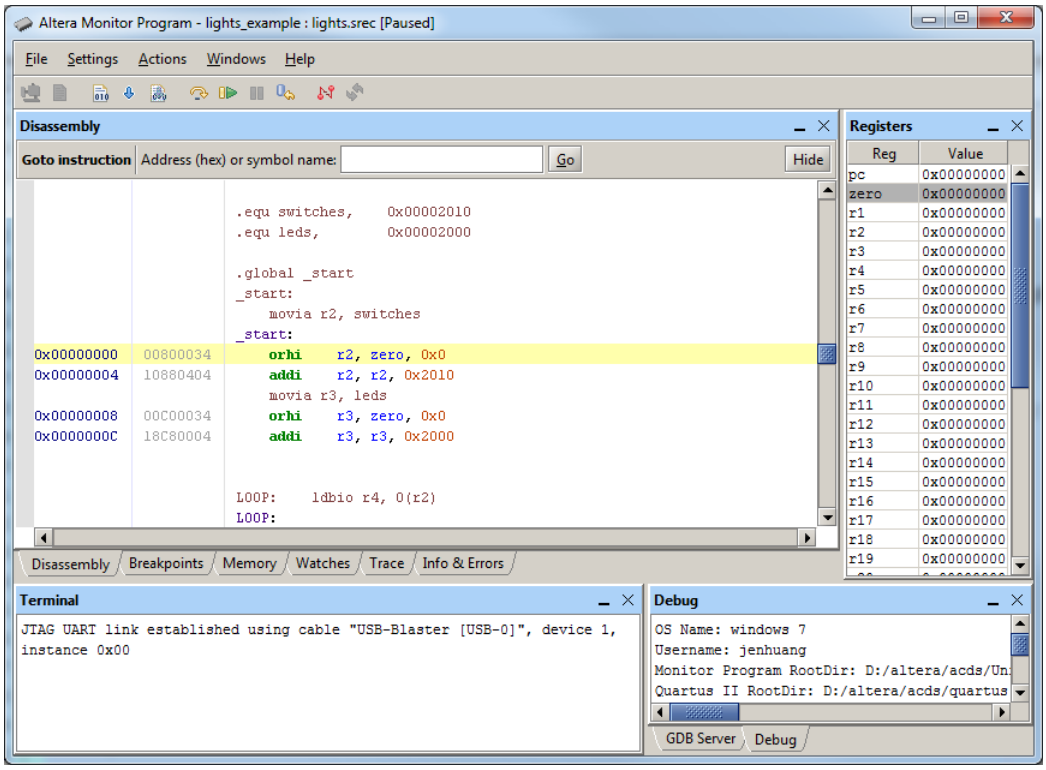

Figure 32. Display of the downloaded program.