

## Lab#4



### – Digital Electronics –

## Building and programming a simple NIOS system

The purpose of this laboratory session is to learn how to build and program a simple NIOS-based system

#### Contents:

1. Building a NIOS system
2. Do it yourself

#### Abbreviations and acronyms:

IC – Integrated Circuit

LED – Light Emitting Diode

MUX – Multiplexer

VHDL – Very high speed integrated circuits Hardware Description Language

[VHDL cookbook: <http://www.onlinefreeebooks.net/engineering-ebooks/electrical-engineering/the-vhdl-cookbook-pdf.html>]

**USE EXACTLY THE SAME I/O PINS SPECIFIED IN THIS DOCUMENT.**

## 1 – Building a NIOS system

Follow the tutorial in the appendix. It explains you how to build a simple NIOS system with QSYS and how to run a simple program.

**PLEASE REFER TO THE LAB#0 FILE FOR DETAILS ON PROGRAMMING THE FPGA**

## 2 – Do it yourself

- a. **Create** a new Quartus project (*lab2\_system*) with the following ports which correspond to names already defined in the pin assignment .csv file.
  - CLOCK\_50: in std\_logic
  - KEY: in std\_logic\_vector(3 downto 0)
  - SW: in std\_logic\_vector(7 downto 0)
  - HEX0: out std\_logic\_vector(6 downto 0)
  - HEX1: out std\_logic\_vector(6 downto 0)
  - LEDR : out std\_logic\_vector(7 downto 0)
- b. Stemming from the tutorial **create** with the QSYS Tool a NIOS processor system (*lab2\_processor*) with
  - one NIOS II/e processor
  - a 4kB one chip memory
  - one 4 bits input parallel port
  - one 8 bits input parallel port
  - one 8 bits output parallel port
  - two 7 bits output parallel port
  - a JTAG UART
- c. **Declare** *lab2\_processor* as a component inside *lab2\_system* and **instantiate** it connecting:
  - the processor clock to CLOCK\_50 and the reset to KEY(0).
  - the four bits input port to KEY and the 8 bits input port to SW
  - the 8 bit output port to LEDR and the two 7 bits output ports to 7 segment displays HEX0 and HEX1 respectively.
- d. **Compile** the hardware system and verify correctness. Then download it on the DE1-SoC board.
- e. **Write, compile and execute** the 3 different programs doing the following tasks:
  - Connect SW to LEDR
  - Encode SW(3) SW(2) SW(1) SW(0) into 7 segment display HEX0. The mapping from the binary input to the output is the one that makes input 0000 corresponding to '0' on the display, 0001 corresponding to '1' and so on till 1111 corresponding to 'F'.
  - Count the rising transitions of the signal coming from pushbutton KEY(1). The maximum count is 256. Then, display the output of the counter in hexadecimal code on 7-segment displays HEX1 and HEX0. Reset the counter if pushbutton KEY(2) is pressed.

### Deliverable report must contain:

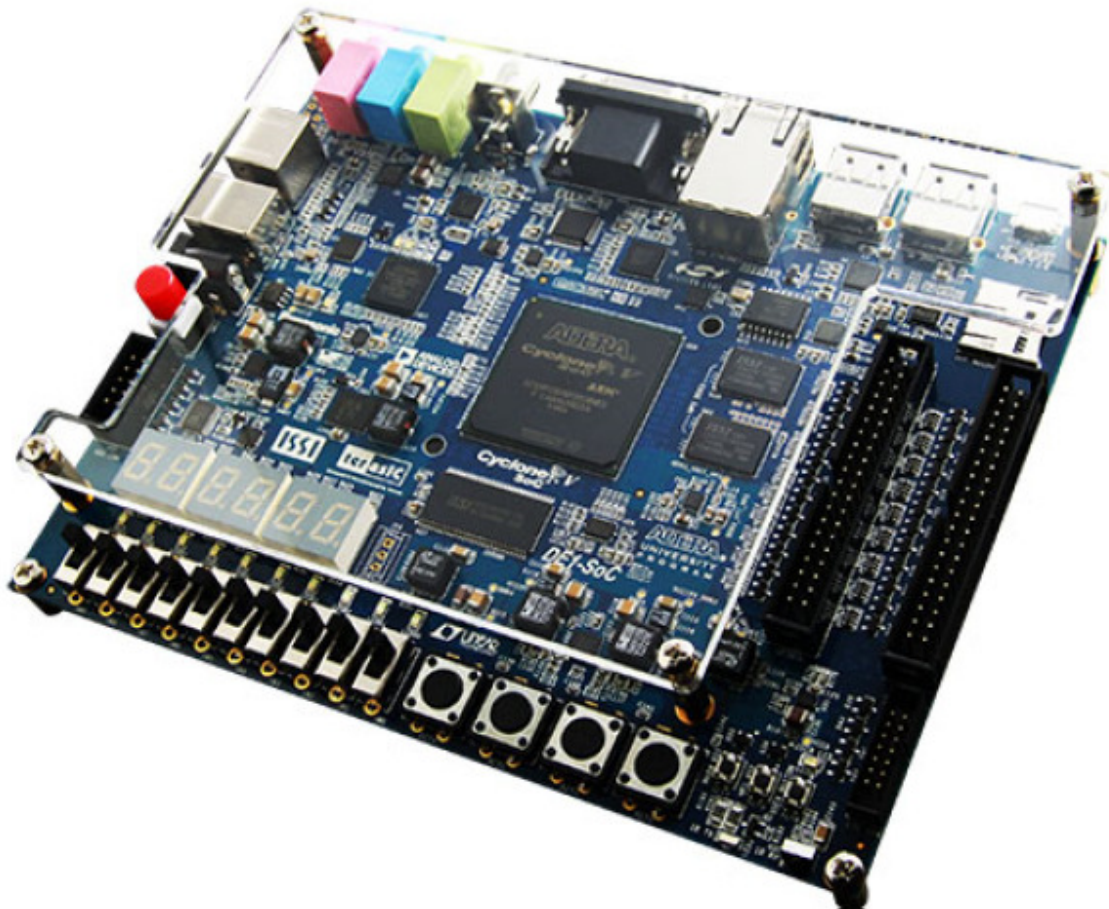
- A snapshot of the final configuration of the SOPC tool.
- The top entity VHDL description.
- The listings of the programs, with comments to explain the code.
- A snapshot of the NIOS EDS tool after one of the programs has been executed.

## APPENDIX – Building a NIOS-based system with QSYS

**READ THIS VERY CAREFULLY**

### 1. Introduction

The tutorial is intended for the DE1-SoC board (see Figure 1)



**Figure 1 DE1-SoC board**

This tutorial focuses on the simple hardware system depicted in Figure 2. 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 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 build tool based on Eclipse to download the designed circuit into the FPGA device, and download and execute a Nios II program that performs the desired task.

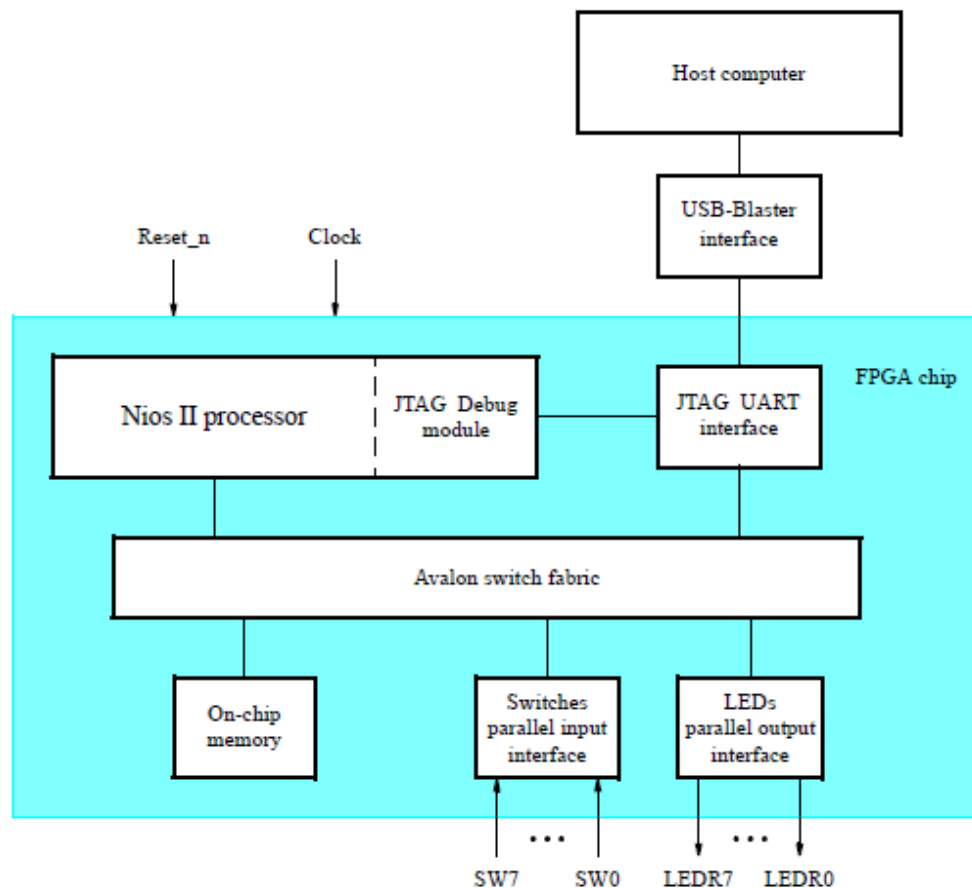


Figure 2 Simple Nios based system

## 2. The 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 the DE1-SoC board, namely **Cyclone V SoC 5CSEMA5F31C6**.

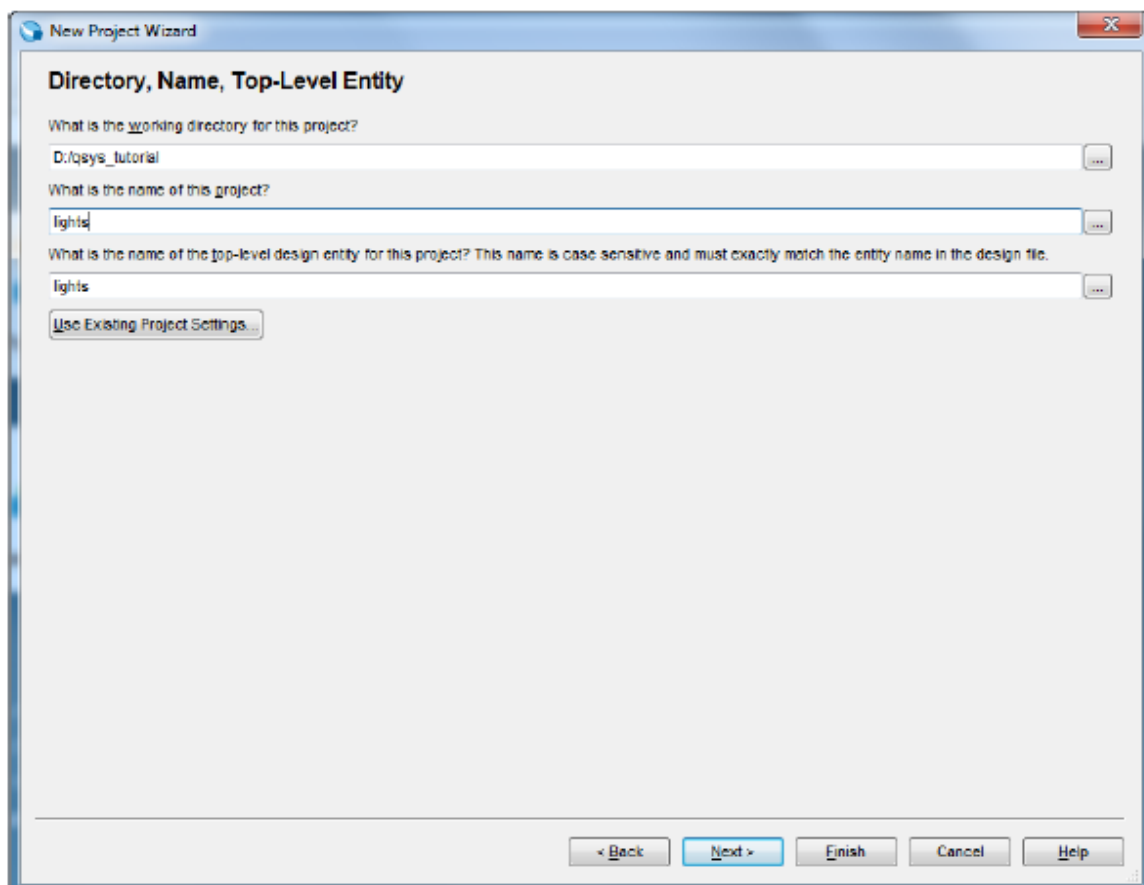


Figure 3 Create a new project

2. After completing the New ProjectWizard 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.

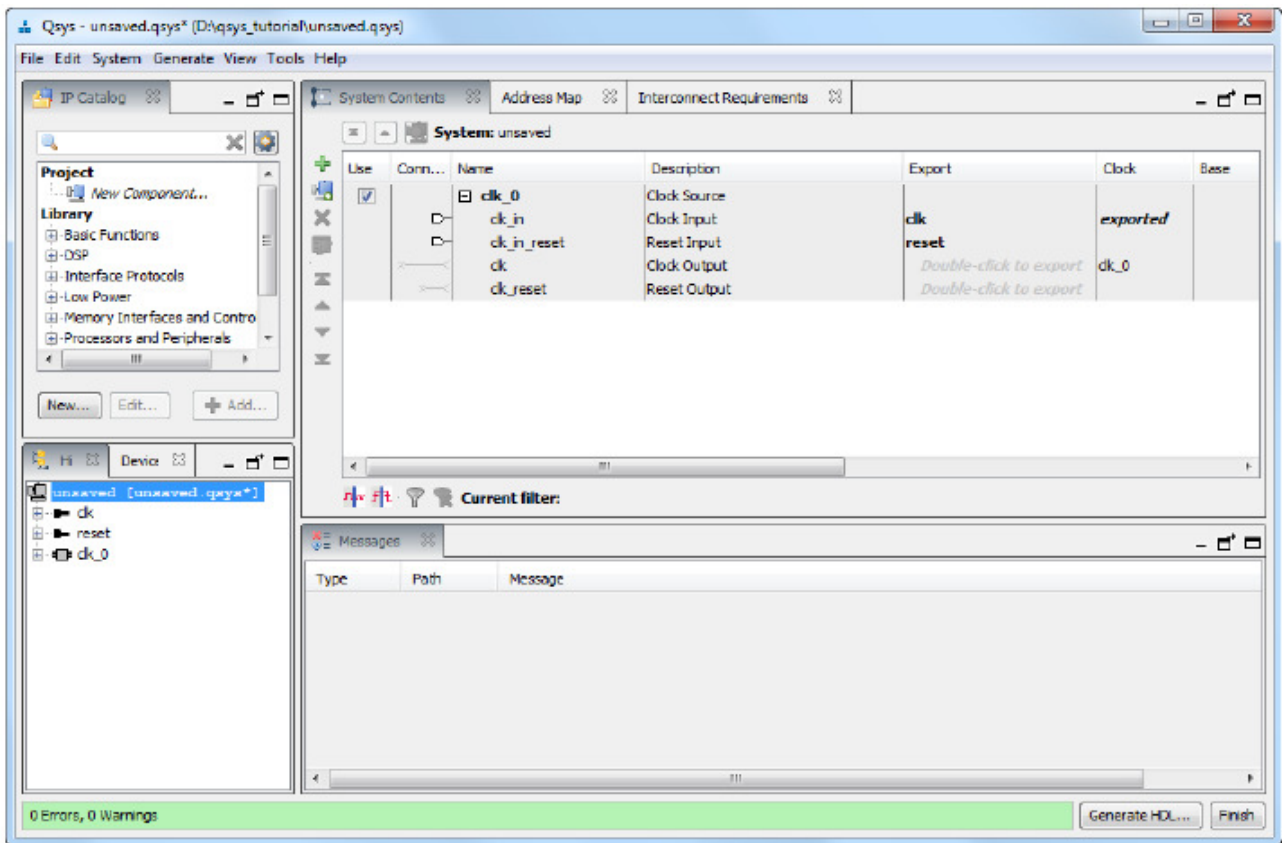


Figure 4 Create a new Nios II system

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.
  - 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. Click **Finish** to return to the main Qsys window, which now shows the Nios II processor specified as indicated in Figure 6.

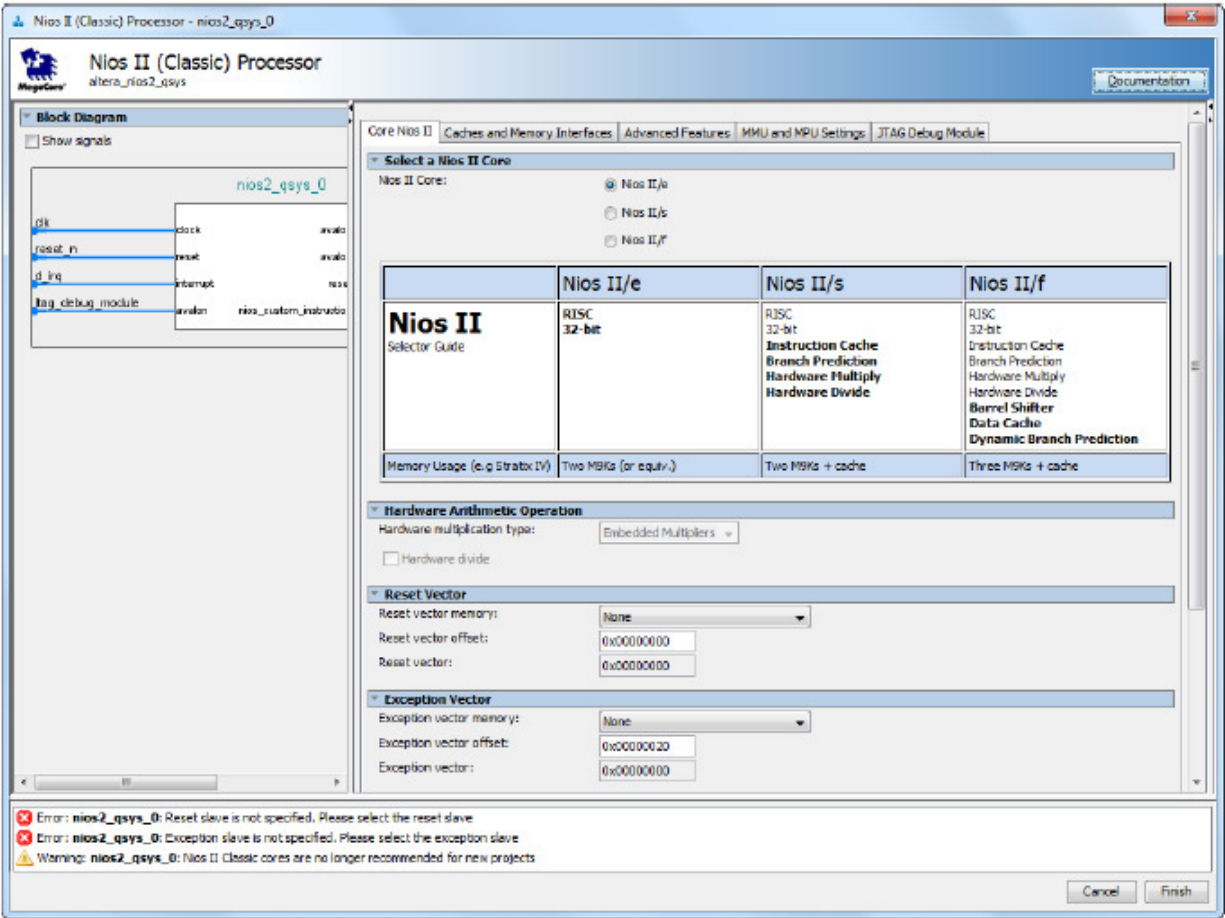


Figure 5 Create a Nios II processor



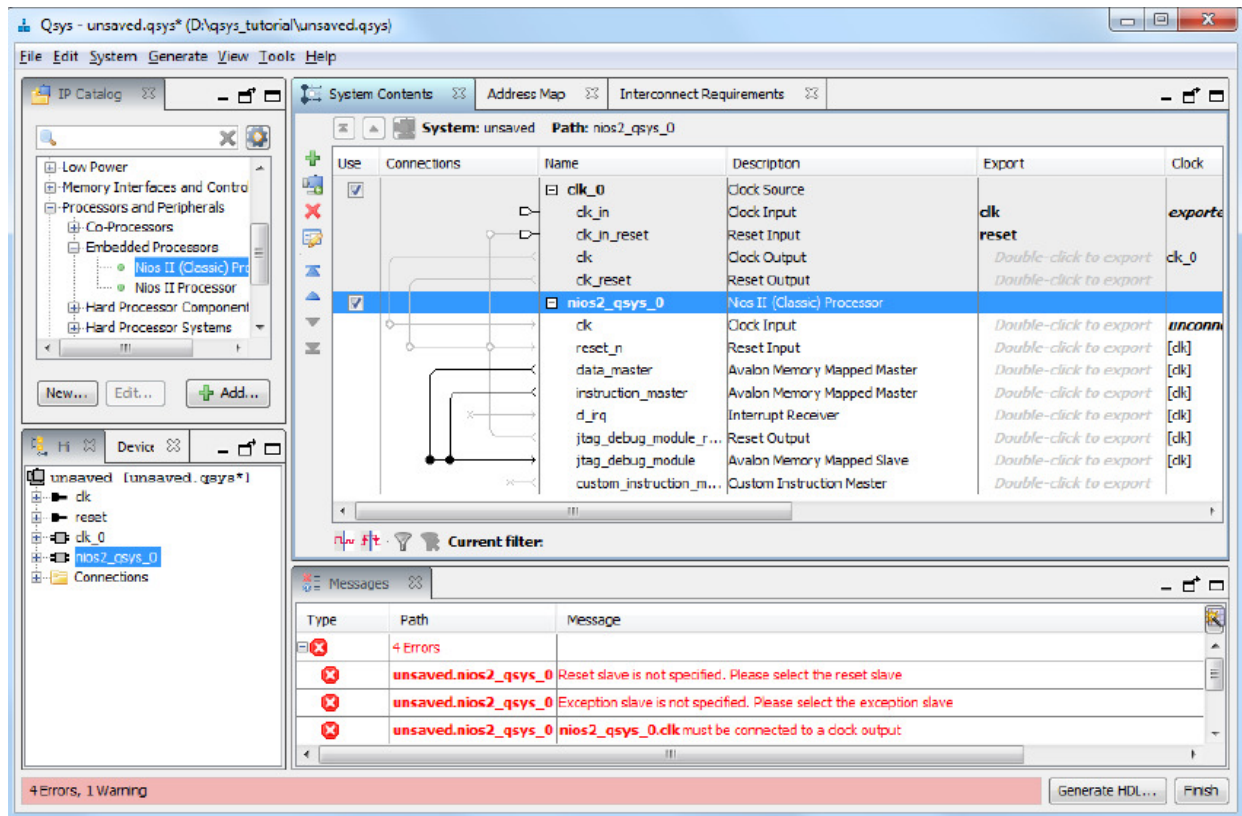


Figure 6 Inclusion of the Nios II processor in the design

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



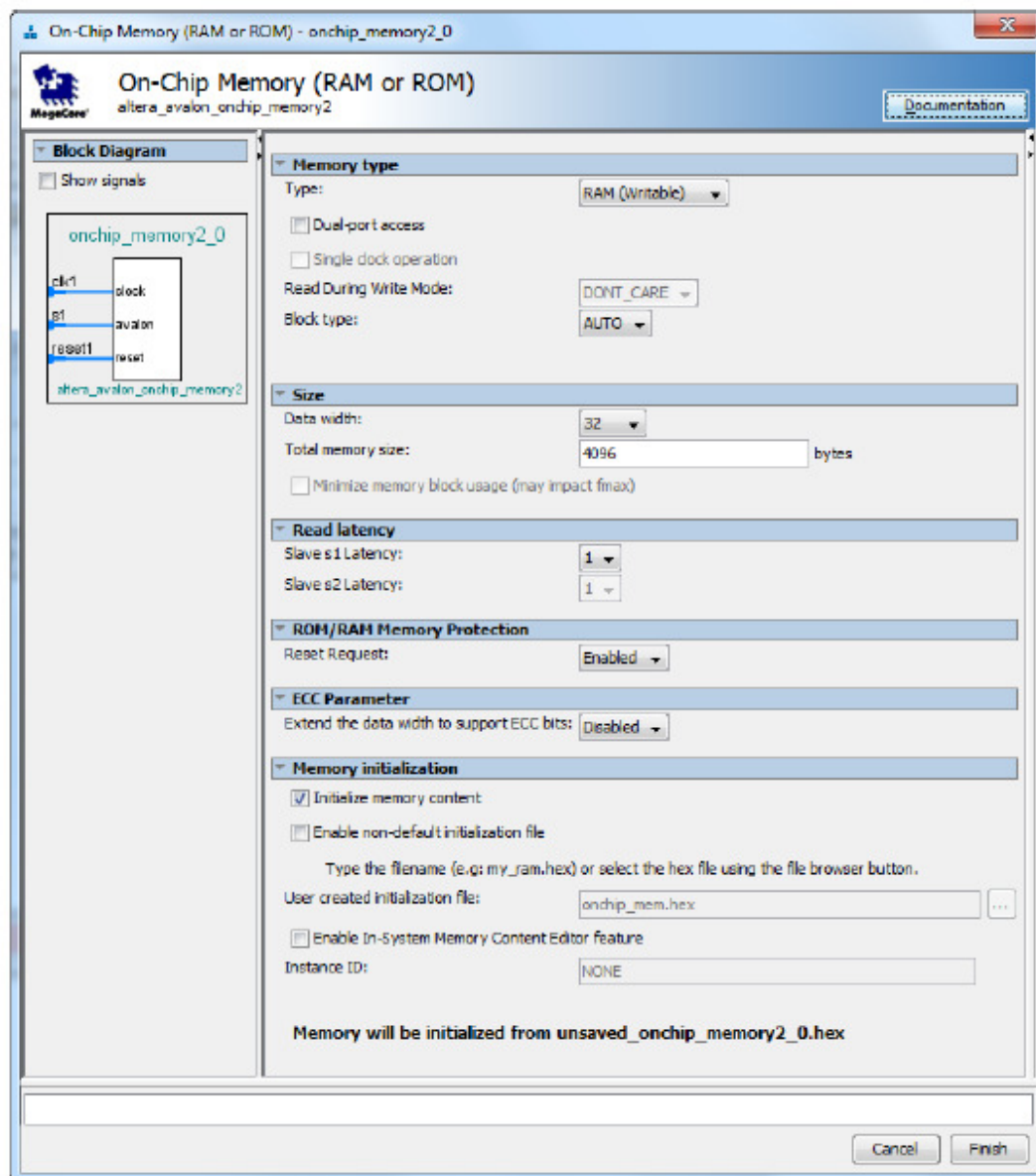


Figure 7 Define the on chip memory

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. We want the clock of the memory be the same one as the processor. Thus, connect clock inputs of the processor and the memory to the clock output of the clock component. Then, the reset to system can come either from the clock component or from the jtag\_debug\_module. So connect reset inputs of the processor and the memory to both the reset output of the clock component and the jtag\_debug\_module reset output. Finally, with a Von Neuman architecture the memory contains both data and instructions: connect 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.

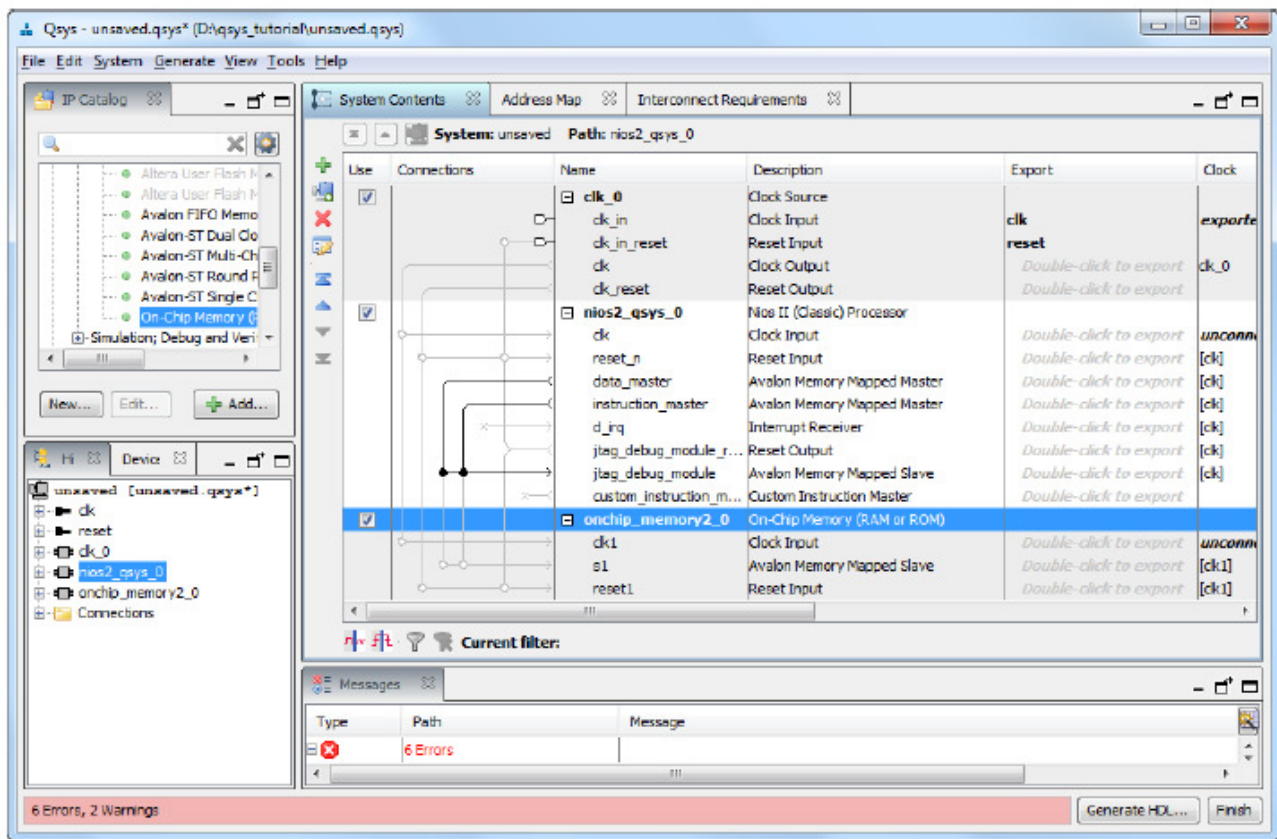


Figure 8 The on chip memory included in the design

Connections	Name	Description	Exp...	Clock
	<b>clk_0</b>	Clock Source		
	clk_in	Clock Input	clk	exported
	clk_in_reset	Reset Input	reset	
	clk	Clock Output	Double-click to export	clk_0
	clk_reset	Reset Output	Double-click to export	
	<b>nios2_qsys_0</b>	Nios II (Classic) Processor		
	clk	Clock Input	Double-click to export	unconnected [clk]
	reset_n	Reset Input	Double-click to export	
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]
	d_irq	Interrupt Receiver	Double-click to export	[clk]
	jtag_debug_module_r...	Reset Output	Double-click to export	[clk]
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]
	custom_instruction_m...	Custom Instruction Master	Double-click to export	
	<b>onchip_memory2_0</b>	On-Chip Memory (RAM or ROM)		
	clk1	Clock Input	Double-click to export	unconnected [clk1]
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]
	reset1	Reset Input	Double-click to export	[clk1]

Figure 9 Connections before

Connections	Name	Description	Exp...
	clk_0	Clock Source	
	clk_in	Clock Input	clk
	clk_in_reset	Reset Input	reset
	clk	Clock Output	Double
	clk_reset	Reset Output	Double
	nios2_qsys_0	Nios II (Classic) Processor	
	clk	Clock Input	Double
	reset_in	Reset Input	Double
	data_master	Avalon Memory Mapped Master	Double
	instruction_master	Avalon Memory Mapped Master	Double
	d_irq	Interrupt Receiver	Double
	jtag_debug_module_r...	Reset Output	Double
	jtag_debug_module	Avalon Memory Mapped Slave	Double
	custom_instruction_m...	Custom Instruction Master	Double
	onchip_memory2_0	On-Chip Memory (RAM or ROM)	
	clk1	Clock Input	Double
	s1	Avalon Memory Mapped Slave	Double
	reset1	Reset Input	Double

Figure 10 Connections after

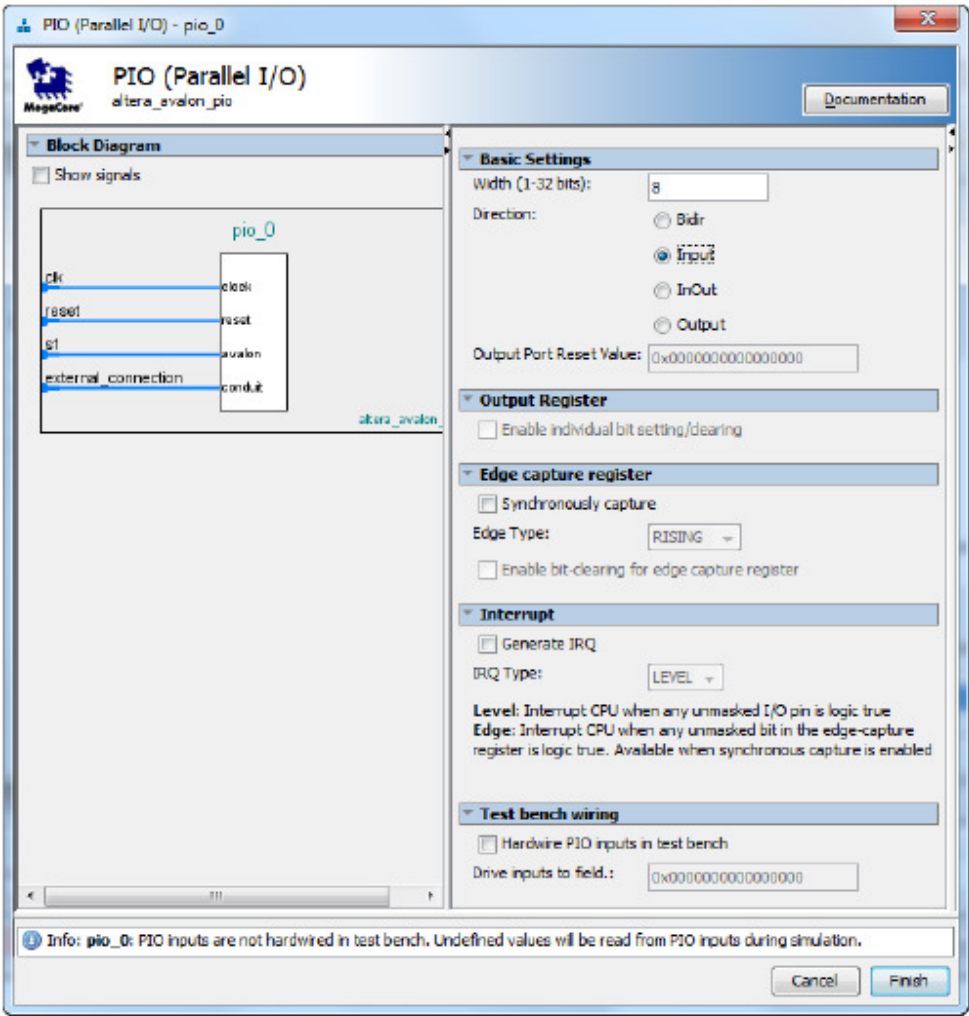


Figure 11 Define a parallel input interface

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**.

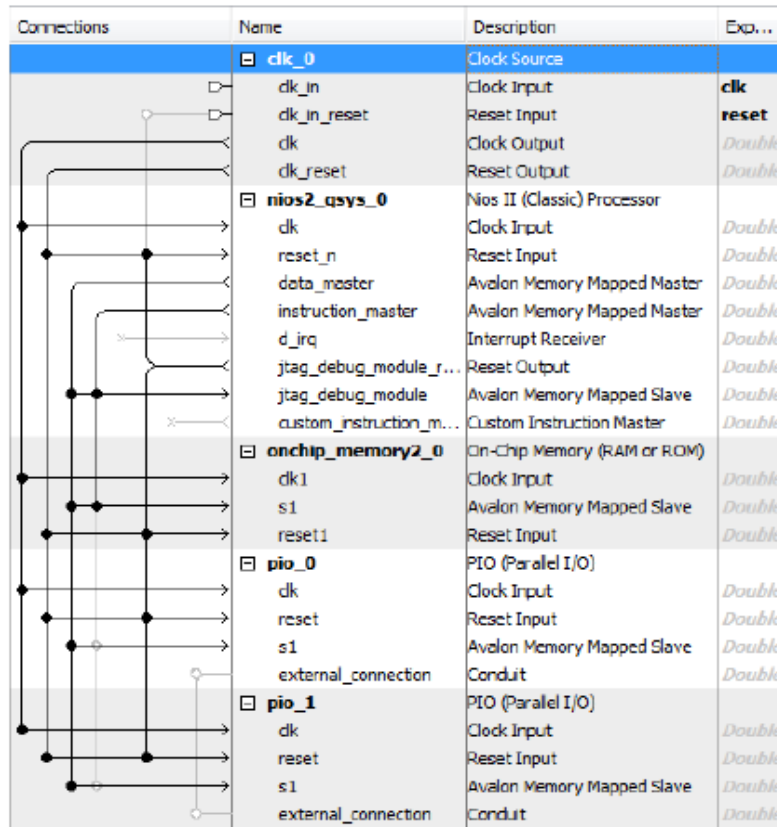


Figure 12 The system with all components and connections

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. We need the clock of the PIOs to be same one as the processor (and the memory). Connect clock input of the PIO to the clock output of the clock component. Similarly, for the reset: connect reset input of the PIO to the reset output of the clock component and the jtag\_debug\_module\_reset output. Since we want to send/receive only data (not instructions) to/from PIOs, connect the s1 input of the PIO only to the data\_master output of the processor.

The resulting design is depicted in Figure 12.

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

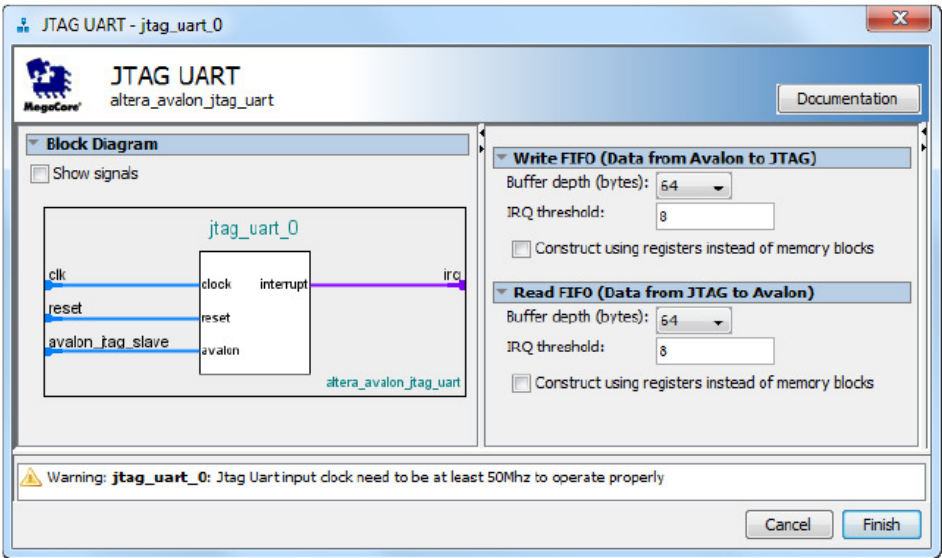


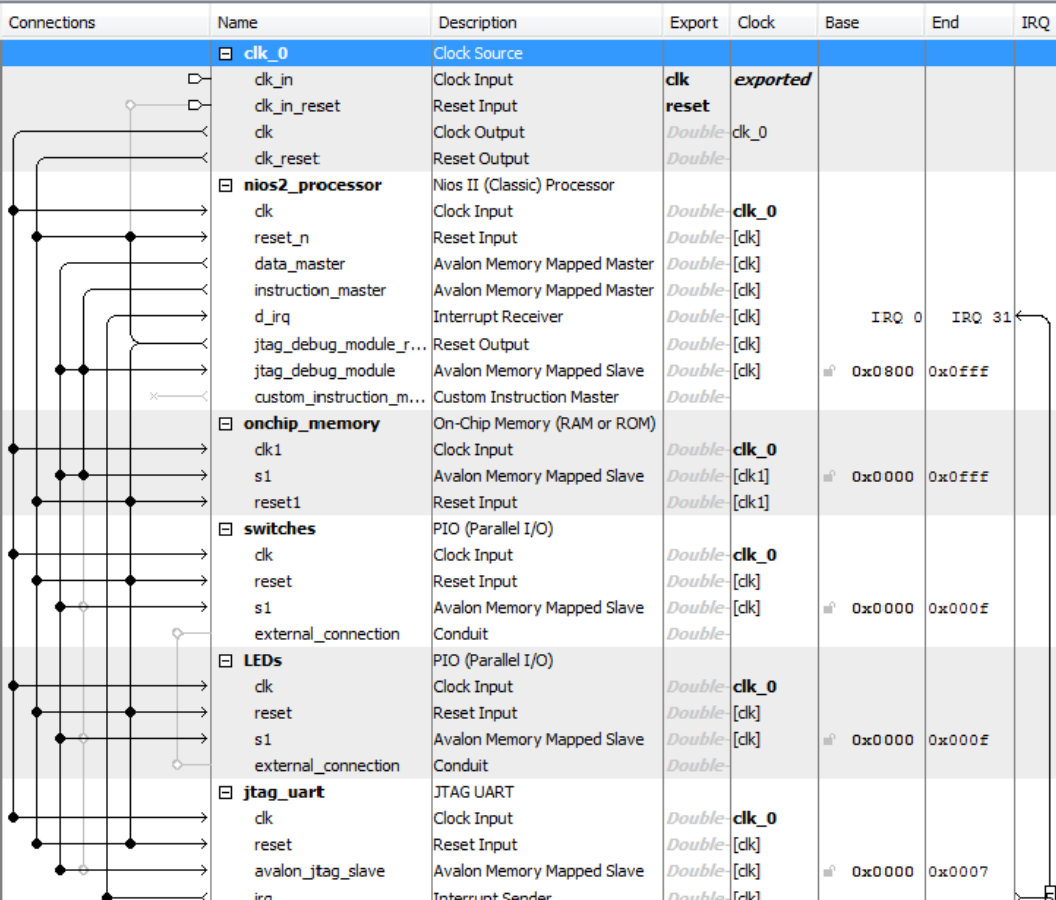
Figure 13 Define the JTAG UART interface

Connections	Name	Description	Export	Clock	Base	End	IRQ
	<b>clk_0</b>	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double clk_0				
	clk_reset	Reset Output	Double				
	<b>nios2_qsys_0</b>	Nios II (Classic) Processor					
	clk	Clock Input	Double clk_0				
	reset_n	Reset Input	Double [clk]				
	data_master	Avalon Memory Mapped Master	Double [clk]				
	instruction_master	Avalon Memory Mapped Master	Double [clk]				
	d_irq	Interrupt Receiver	Double [clk]			IRQ 0	IRQ 31
	jtag_debug_module_r...	Reset Output	Double [clk]				
	jtag_debug_module	Avalon Memory Mapped Slave	Double [clk]		0x0800	0x0FFF	
	custom_instruction_m...	Custom Instruction Master	Double				
	<b>onchip_memory2_0</b>	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double clk_0				
	s1	Avalon Memory Mapped Slave	Double [clk1]		0x0000	0x0FFF	
	reset1	Reset Input	Double [clk1]				
	<b>pio_0</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double clk_0				
	reset	Reset Input	Double [clk]				
	s1	Avalon Memory Mapped Slave	Double [clk]		0x0000	0x000F	
	external_connection	Conduit	Double				
	<b>pio_1</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double clk_0				
	reset	Reset Input	Double [clk]				
	s1	Avalon Memory Mapped Slave	Double [clk]		0x0000	0x000F	
	external_connection	Conduit	Double				
	<b>jtag_uart_0</b>	JTAG UART					
	clk	Clock Input	Double clk_0				
	reset	Reset Input	Double [clk]				
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double [clk]		0x0000	0x0007	
	irq	Interrupt Sender	Double [clk]				

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

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..

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 used 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.

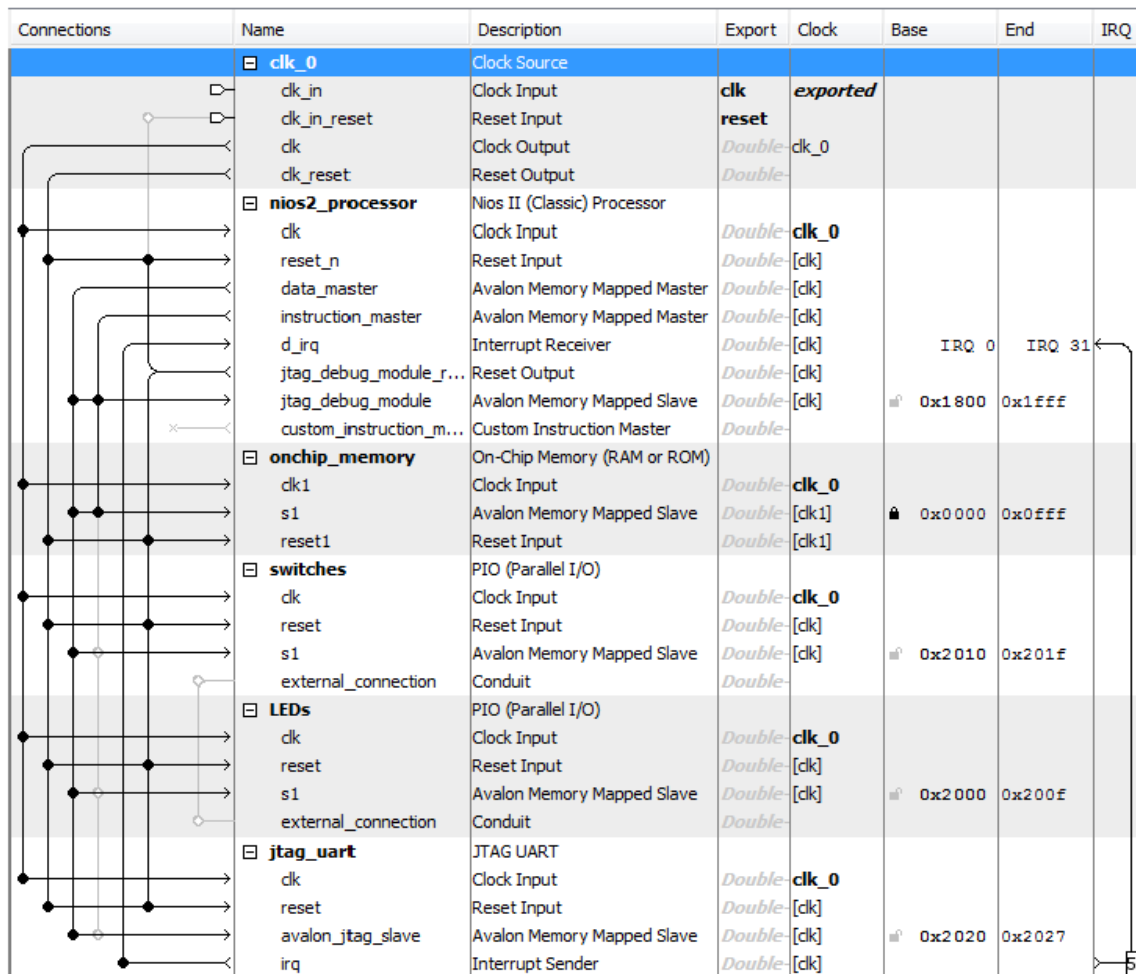


Connections	Name	Description	Export	Clock	Base	End	IRQ
	<b>clk_0</b>	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double	clk_0			
	clk_reset	Reset Output	Double				
	<b>nios2_processor</b>	Nios II (Classic) Processor					
	clk	Clock Input	Double	clk_0			
	reset_n	Reset Input	Double	[clk]			
	data_master	Avalon Memory Mapped Master	Double	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double	[clk]			
	d_irq	Interrupt Receiver	Double	[clk]		IRQ 0	IRQ 31
	jtag_debug_module_r...	Reset Output	Double	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double	[clk]	0x0800	0x0fff	
	custom_instruction_m...	Custom Instruction Master	Double				
	<b>onchip_memory</b>	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double	clk_0			
	s1	Avalon Memory Mapped Slave	Double	[clk1]	0x0000	0x0fff	
	reset1	Reset Input	Double	[clk1]			
	<b>switches</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double	clk_0			
	reset	Reset Input	Double	[clk]			
	s1	Avalon Memory Mapped Slave	Double	[clk]	0x0000	0x000f	
	external_connection	Conduit	Double				
	<b>LEDs</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double	clk_0			
	reset	Reset Input	Double	[clk]			
	s1	Avalon Memory Mapped Slave	Double	[clk]	0x0000	0x000f	
	external_connection	Conduit	Double				
	<b>jtag_uart</b>	JTAG UART					
	clk	Clock Input	Double	clk_0			
	reset	Reset Input	Double	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double	[clk]	0x0000	0x0007	
	irq	Interrupt Sender	Double	[clk]			

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.





Connections	Name	Description	Export	Clock	Base	End	IRQ
	<b>clk_0</b>	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double: clk_0				
	clk_reset	Reset Output	Double:				
	<b>nios2_processor</b>	Nios II (Classic) Processor					
	clk	Clock Input	Double: clk_0				
	reset_n	Reset Input	Double: [clk]				
	data_master	Avalon Memory Mapped Master	Double: [clk]				
	instruction_master	Avalon Memory Mapped Master	Double: [clk]				
	d_irq	Interrupt Receiver	Double: [clk]			IRQ 0	IRQ 31
	jtag_debug_module_r...	Reset Output	Double: [clk]				
	jtag_debug_module	Avalon Memory Mapped Slave	Double: [clk]		0x1800	0x1fff	
	custom_instruction_m...	Custom Instruction Master	Double:				
	<b>onchip_memory</b>	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double: clk_0				
	s1	Avalon Memory Mapped Slave	Double: [clk1]		0x0000	0x0fff	
	reset1	Reset Input	Double: [clk1]				
	<b>switches</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double: clk_0				
	reset	Reset Input	Double: [clk]				
	s1	Avalon Memory Mapped Slave	Double: [clk]		0x2010	0x201f	
	external_connection	Conduit	Double:				
	<b>LEDs</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double: clk_0				
	reset	Reset Input	Double: [clk]				
	s1	Avalon Memory Mapped Slave	Double: [clk]		0x2000	0x200f	
	external_connection	Conduit	Double:				
	<b>jtag_uart</b>	JTAG UART					
	clk	Clock Input	Double: clk_0				
	reset	Reset Input	Double: [clk]				
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double: [clk]		0x2020	0x2027	
	irq	Interrupt Sender	Double: [clk]				

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
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.



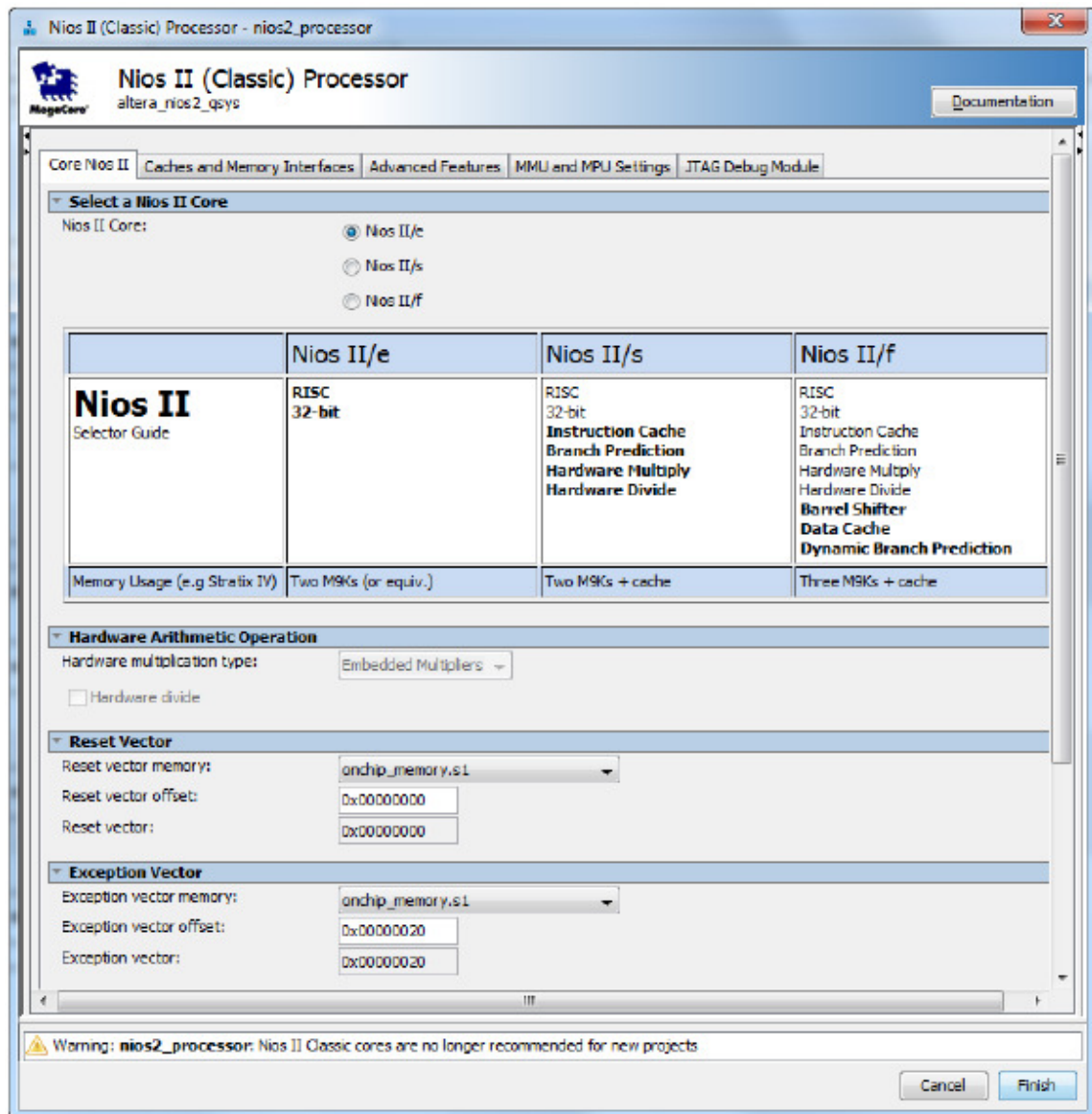


Figure 17 Define the reset and exception vectors

- 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.

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.

Connections	Name	Description	Export	Clock	Base	End	IRQ
	<b>clk_0</b>	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double-click to	clk_0			
	clk_reset	Reset Output	Double-click to				
	<b>nios2_processor</b>	Nios II (Classic) Processor					
	clk	Clock Input	Double-click to	clk_0			
	reset_in	Reset Input	Double-click to	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	d_irq	Interrupt Receiver	Double-click to	[clk]		IRQ 0	IRQ 31
	jtag_debug_module_r...	Reset Output	Double-click to	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to	[clk]	m0	0x1000	0x1fff
	custom_instruction_m...	Custom Instruction Master	Double-click to				
	<b>onchip_memory</b>	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to	clk_0			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk1]	m0	0x0000	0x0fff
	reset1	Reset Input	Double-click to	[clk1]			
	<b>switches</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	m0	0x2010	0x201f
	external_connection	Conduit	switches				
	<b>leds</b>	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	m0	0x2000	0x200f
	external_connection	Conduit	leds				
	<b>jtag_uart</b>	JTAG UART					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	m0	0x2020	0x2027
	irq	Interrupt Sender	Double-click to	[clk]			

Figure 18 The complete system

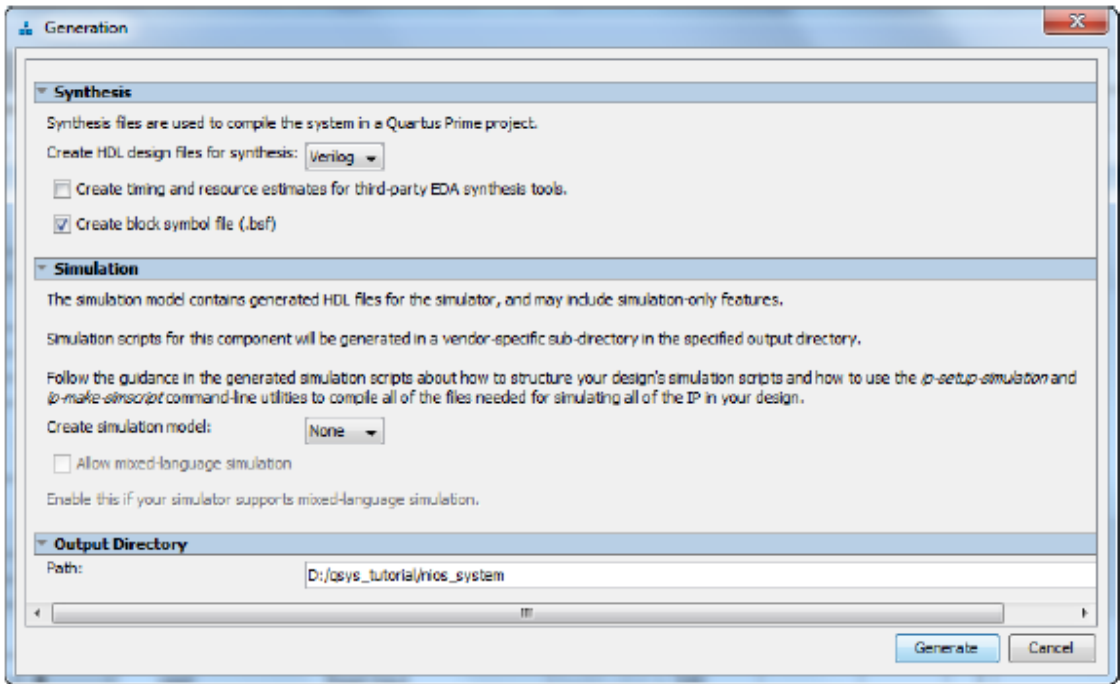


Figure 19 Generation of the system

### 3. 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

#### 3.1 Instantiation of the Module Generated by the Qsys Tool

The Qsys tool generates a Verilog/VHDL module that defines the desired Nios II system. In our design, this module will have been generated in the *nios\_system.v/nios\_system.vhd* file, which can be found in the directory *qsys\_tutorial/nios\_system/synthesis* of the project. The Qsys tool generates Verilog/VHDL 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/VHDL code in the *nios\_system.v/nios\_system.vhd* 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.

```
entity nios_system is
  port (
    clk_clk      : in  std_logic              := '0';           -- clk.clk
    hex0_export  : out std_logic_vector(6 downto 0);           -- hex0.export
    hex1_export  : out std_logic_vector(6 downto 0);           -- hex1.export
    key_export   : in  std_logic_vector(3 downto 0) := (others => '0'); -- key.export
    leds_export  : out std_logic_vector(7 downto 0);           -- leds.export
    reset_reset_n : in  std_logic              := '0';           -- reset.reset_n
    switches_export : in std_logic_vector(7 downto 0) := (others => '0') -- switches.export
  );
end entity nios_system;
```

**Figure 20 A part of the generated VHDL**

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, as they are the default names in the *DE1\_SoC.qsf* file.

Figure 21 shows a top-level VHDL module that instantiates the Nios II system. Type this code into a file called *lights.vhd*.

```
-- 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;

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 (
        clk_clk: IN STD_LOGIC;
        reset_reset_n : IN STD_LOGIC;
        switches_export : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        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 21 Instantiating the Nios II system using VHDL code**

## 4. Compiling the Quartus Prime Project

Add the *lights.vhd* file to your Quartus Prime project. Also, add the necessary pin assignments for the DE1-SoC board (*DE1\_SoC.qsf*). The procedure for making pin assignments is described in the LAB#0 document.

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 (see LAB#2 appendix). 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.

## 5. Nios II Embedded Development System (EDS)

Having configured the required hardware in the FPGA device, it is now necessary to create and execute an application program that performs the desired operation. Such a program is composed of a user defined part and a system defined support library. In order to perform the execution the program requires:

- to be edited
- to be compiled
- to be linked
- to be transferred to target system and launched

These tasks are supported by the Nios II EDS. After program development when the system has been debugged the executable binary code may be stored into a non-volatile memory in the target system.

The Nios II EDS is an easy-to-use GUI that automates build and makefile management, and integrates a text editor, debugger, the Nios II flash programmer, and the Quartus II Programmer. Software application templates included in the GUI make it easy for new software programmers to get started quickly. We will use the Nios II EDS to compile a simple C language example software program to run on the Nios II standard system configured in the FPGA on your development board.

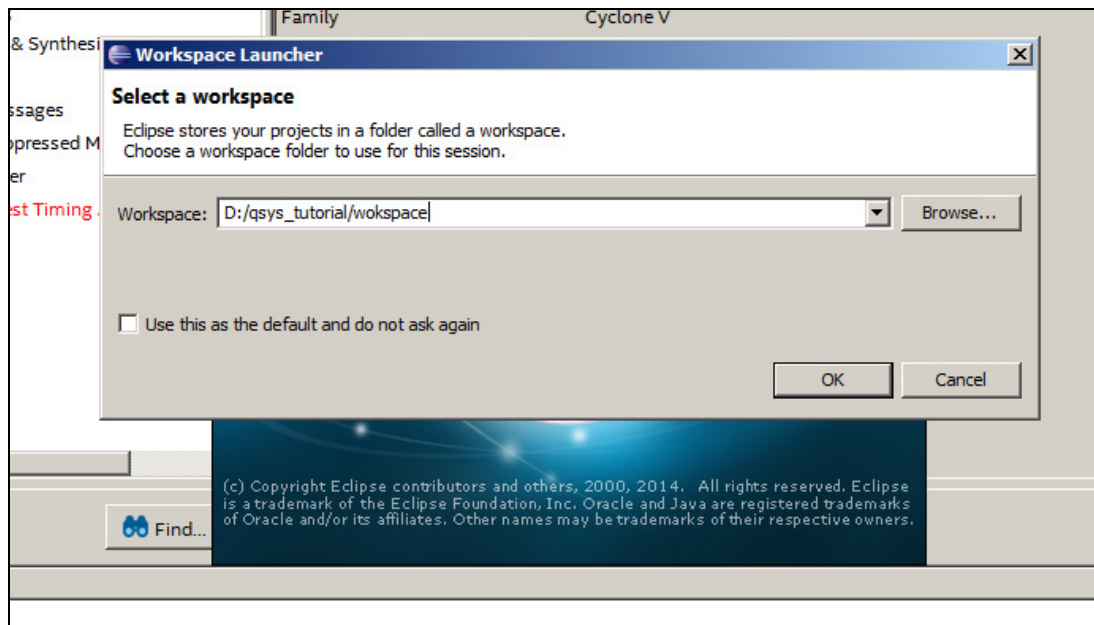


Figure 22 Instantiating the Nios II system using VHDL code

You create a new software project, build it, and run it on the target hardware. To start in the **Tools** menu of Quartus select and click the **Nios II Software Build Tools for Eclipse**. The pop up window shown in Figure 22 will appear. Choose the directory in which Eclipse will store your programs. It is suggested that you create a new one in your *<project directory>* using the **Browse** interface button. Namely, if your project is *qsys\_tutorial*, you can create the workspace inside it. The EDS main window will open as in Figure 23.

### 1. Create a new software project

Select **File > New > Nios II Application and BSP from Template**.

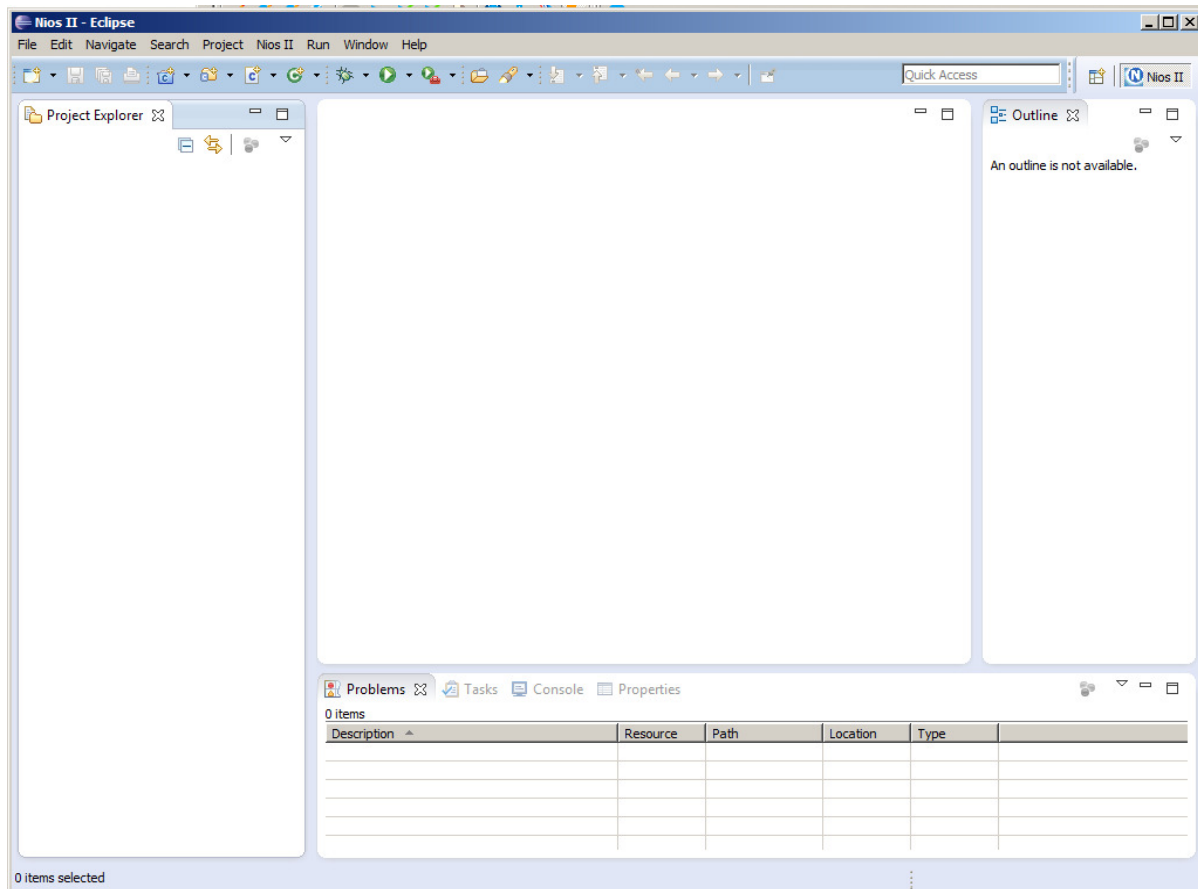


Figure 23 EDS main window

First we introduce the information about the hardware (*Target Hardware Information*). When generating a system, Qsys produces as well a file describing the hardware. For SOPC Information File name, we browse to *qsys\_tutorial* where we have created the hardware project using Qsys and open the SOPC Information File (.sopcinfo) for the design: *nios\_system.sopcinfo*. Choose a name for your software project and in the *Project name* box, we type: *test\_sw\_leds*. In the *Templates* list, select the **Hello World Small** project template (see Figure 24). Then, click **Next** to reach the second window of the wizard in which we click **Finish**.

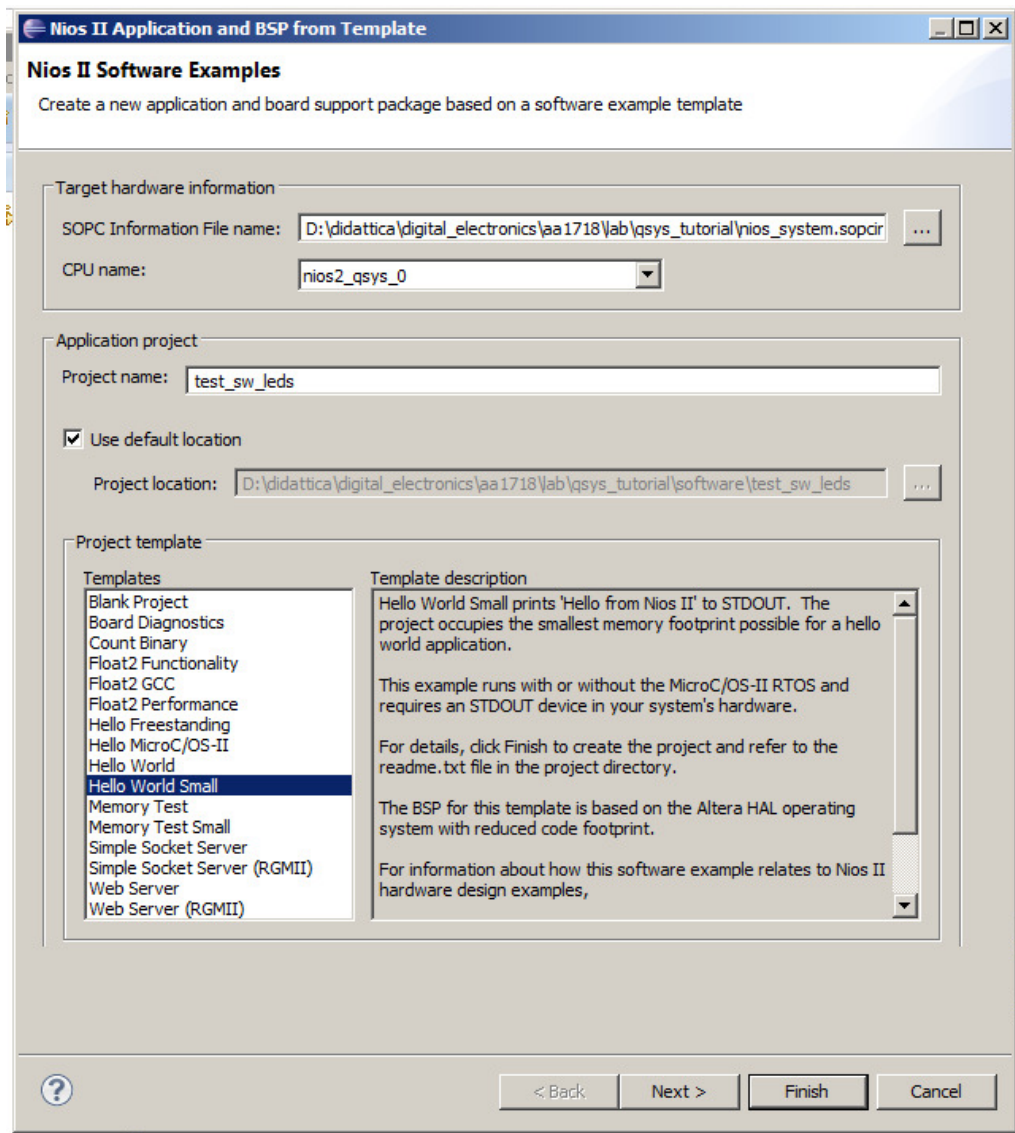


Figure 24 Creating a new application

The Nios II SBT for Eclipse creates the *test\_sw\_leds* project and returns to the Nios II perspective. In the Project Explorer view, expand *test\_sw\_led*. Double-click *hello\_world\_small.c* to view the source code. As you can see the template program includes the reduced stdio library and contains a statement printing the Hello from NIOS string and an infinite waiting loop. If we click on *test\_sw\_leds.bsp* then it is possible to see the contents of the Board Support Package (BSP) associated to the project (see Figure 25). As an example the *system.h* file contains symbols related to the hardware system. Similarly, the *drivers* directory and, inside it, the *inc* directory, contains the definition of the functions for accessing our peripherals in a symbolic way.



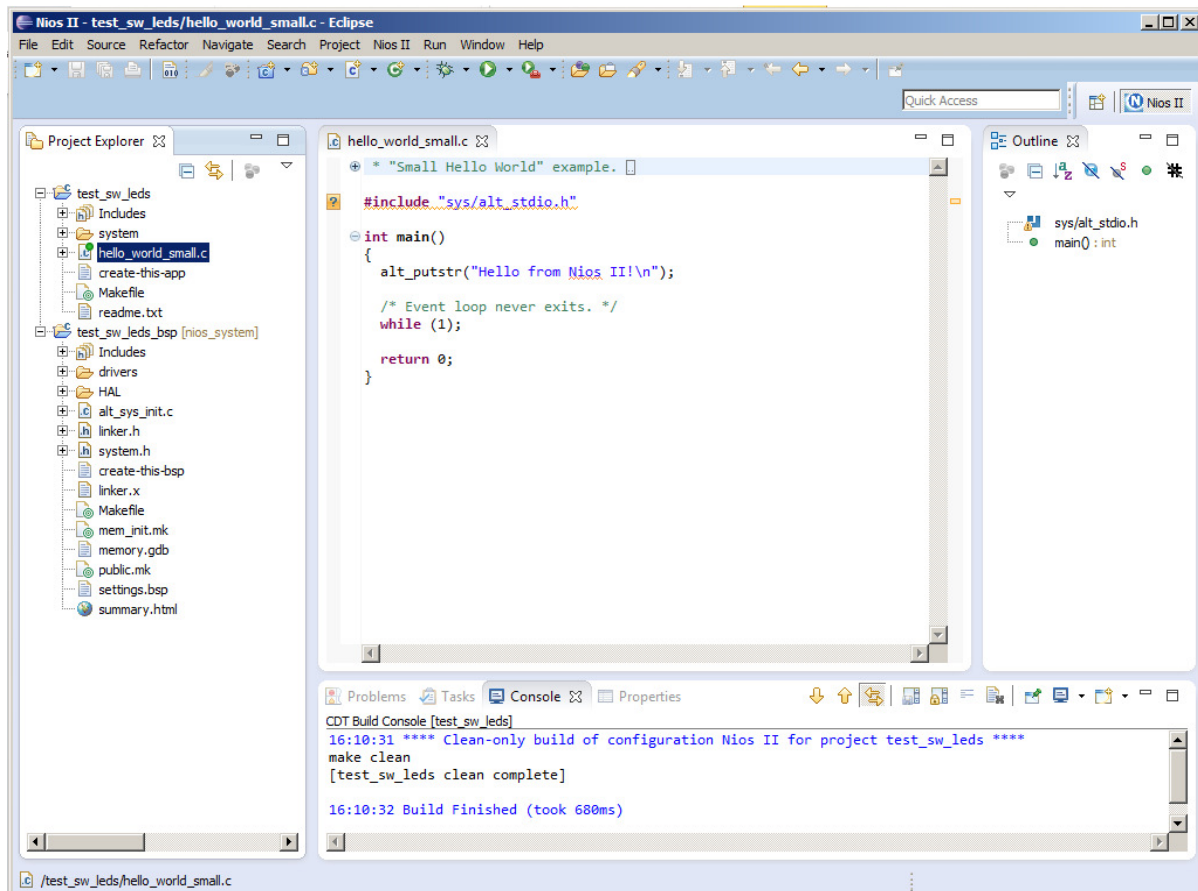


Figure 25 Hello example

## 2. Run the example code

In order to run the example code you have first to compile it: **Project > Build Project**. You can check the compilation correctness switching from the *Console* tab to the *Problems* tab. It should be empty. Now you have run the program on Nios II. Select **Run > Run Configurations** and in the new window choose **Nios II Hardware**.

A new configuration is created and you can now insert the details:

- Give a significant name to the configuration, such as *del\_soc*.
- In the *Project* tab click on the sliding menu next to *Project name* so that you can choose *test\_sw\_leds*. The *Project ELF file name* will be filled automatically to point to the correct ELF file (see Figure 26).
- Switch to the *Target Connection* tab. Into the section *Connections* on the right-most side click **Refresh Connections**. Now the boxes *Processors* and *Byte Stream Devices* are filled with the correct board connection information (see Figure 27).
- Click on **Apply** and then on **Run**.

In the *Console* tab you will see the download progress and if everything is fine the system automatically switches to the *Nios II Console* tab.

If the code is running correctly you see in the *Nios II Console* tab the string "Hello from Nios II!" (See Figure 28).

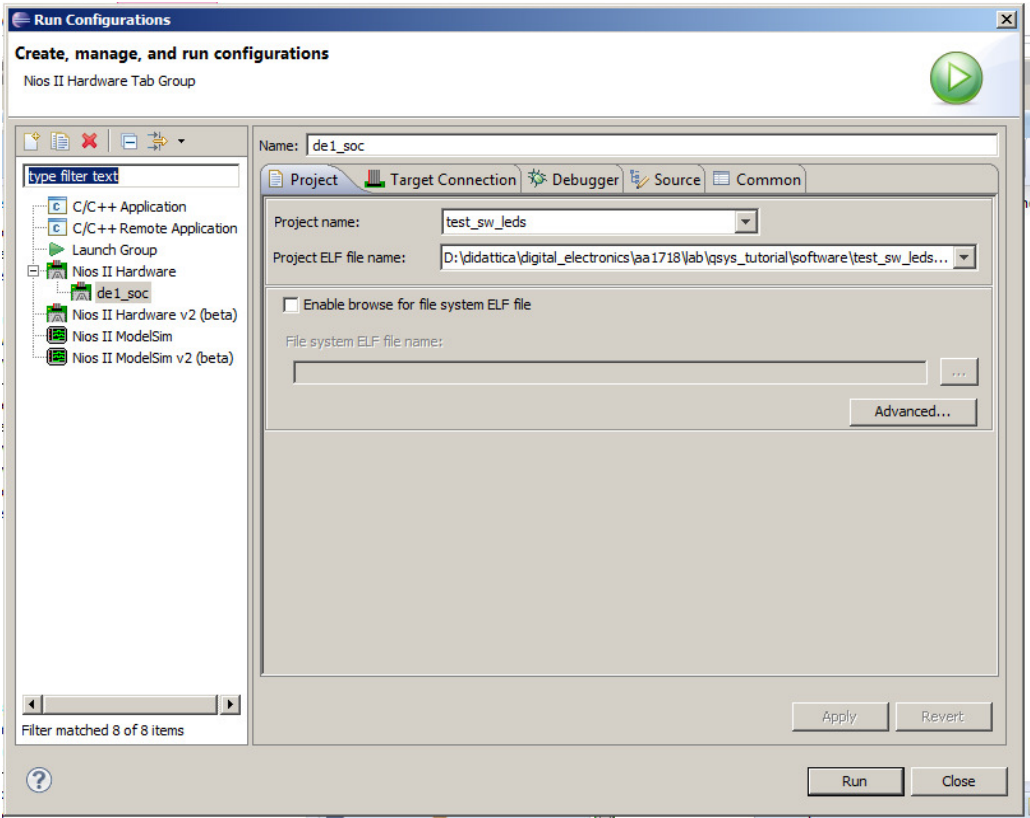


Figure 26 Configuration, Project tab

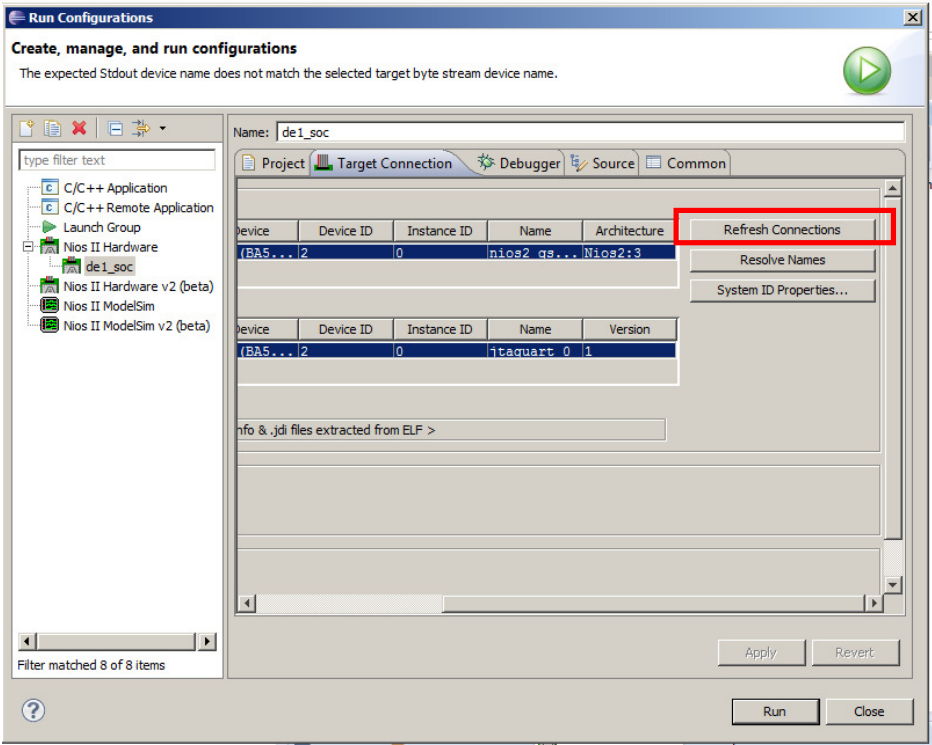


Figure 27 Configuration, Target Connection tab

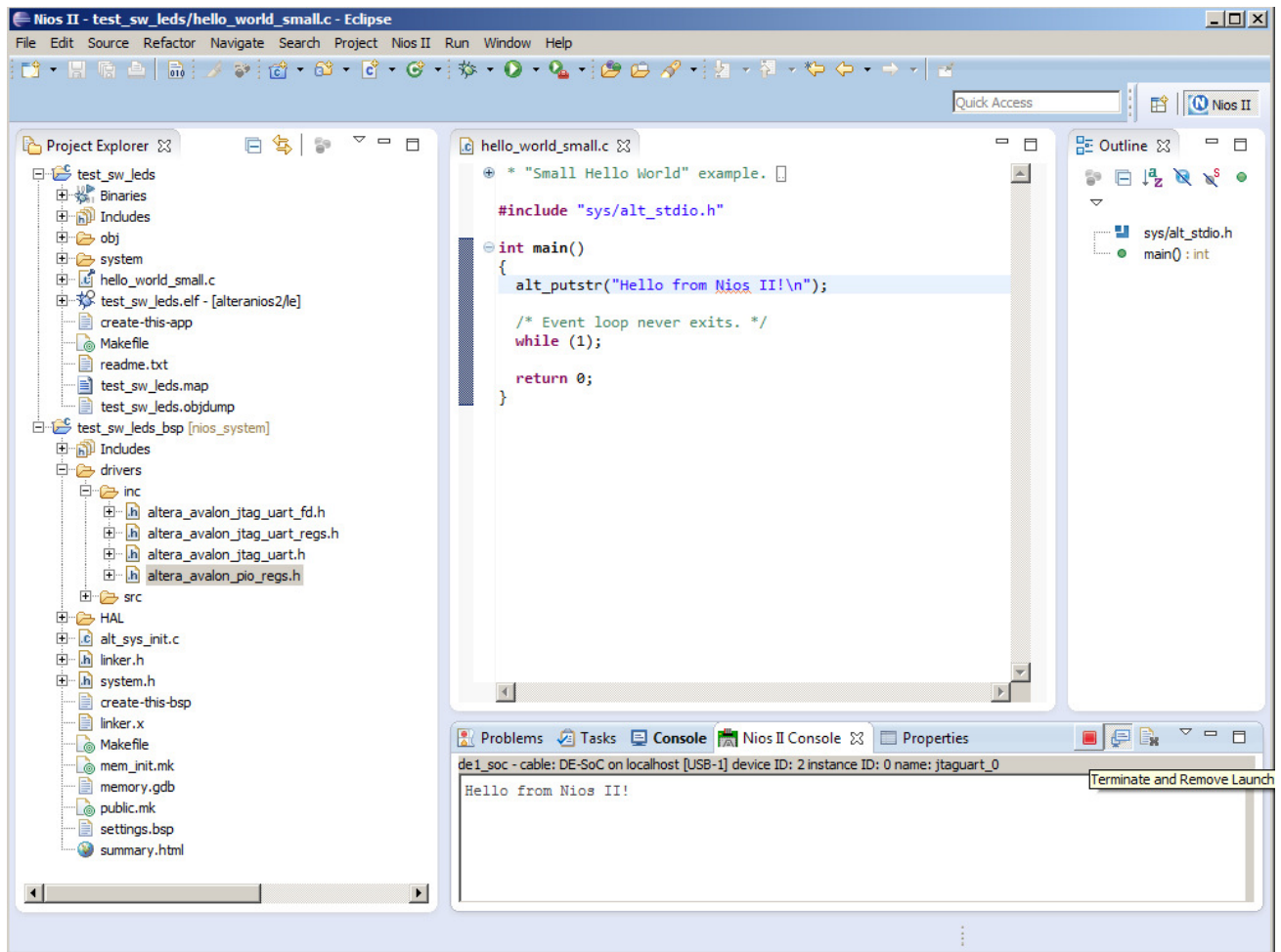


Figure 28 Hello example running

### 3. Customize and run your source code

First stop the execution of the example by clicking on the red square (**Terminate and Remove Launch**) on the right side of the *Nios II Console* tab. Then, we can modify the source code.

What the new code has to do is:

- read the value on the input PIO (*SWITCHES*)
- write the value on the output PIO (*LEDS*)

This can be obtained by using the `IORD` and `IOWR` functions of the PIOs, namely `IORD_ALTERA_AVALON_PIO_DATA` and `IOWR_ALTERA_AVALON_PIO_DATA`, which are defined in the *altera\_avalon\_pio\_regs.h* header file. Thus, the first modification is to include this header file. You can open this file to see the two functions by clicking on **test\_sw\_leds\_bsp** > **drivers** > **inc** > **altera\_avalon\_pio\_regs.h**

The `IORD_ALTERA_AVALON_PIO_DATA(base)` function requires the base address of the PIO. This information has been defined into Qsys when building the system. However, we can exploit the *system.h* header file, which contains mnemonic symbols associated to the base address of each peripheral, instead of writing the base address value. As a consequence, we have first to include the *system.h* header file and then refer to the base address of the *SWITCHES* PIO simply as *SWITCHES\_BASE*.

Similarly, for the `IOWR_ALTERA_AVALON_PIO_DATA(base, data)` we have to specify the base address of the PIO (`LEDS_BASE`) and the value we want to write to the LEDs.

The final code is shown in Figure 29.

```
#include "system.h"
#include "sys/alt_stdio.h"
#include "altera_avalon_pio_regs.h"

int main ()
{
    int flag;

    alt_putstr("Led and switch test\n");

    while (1)
    {
        flag = IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, flag &
0xff);
    }
}
```

**Figure 29 test\_sw\_leds final code**

To compile and execute the new code you have to select **Project > Build Project**. Then, you can run the program by selecting **Run > Run History > 1 del\_soc**.

#### **4. Debugging**

If the behavior of your program is not the expected one you can debug your code. To activate debugging select **Run > Debug History > 1 del\_soc**. When the loading of the program is complete a window pops up asking to switch to the *Nios II debug perspective*, click on **Yes**. This activates the debug interface shown in Figure 30, where you can perform debugging of your code, such as:

- adding breakpoints to your code;
- watching variable content
- perform step-by-step execution

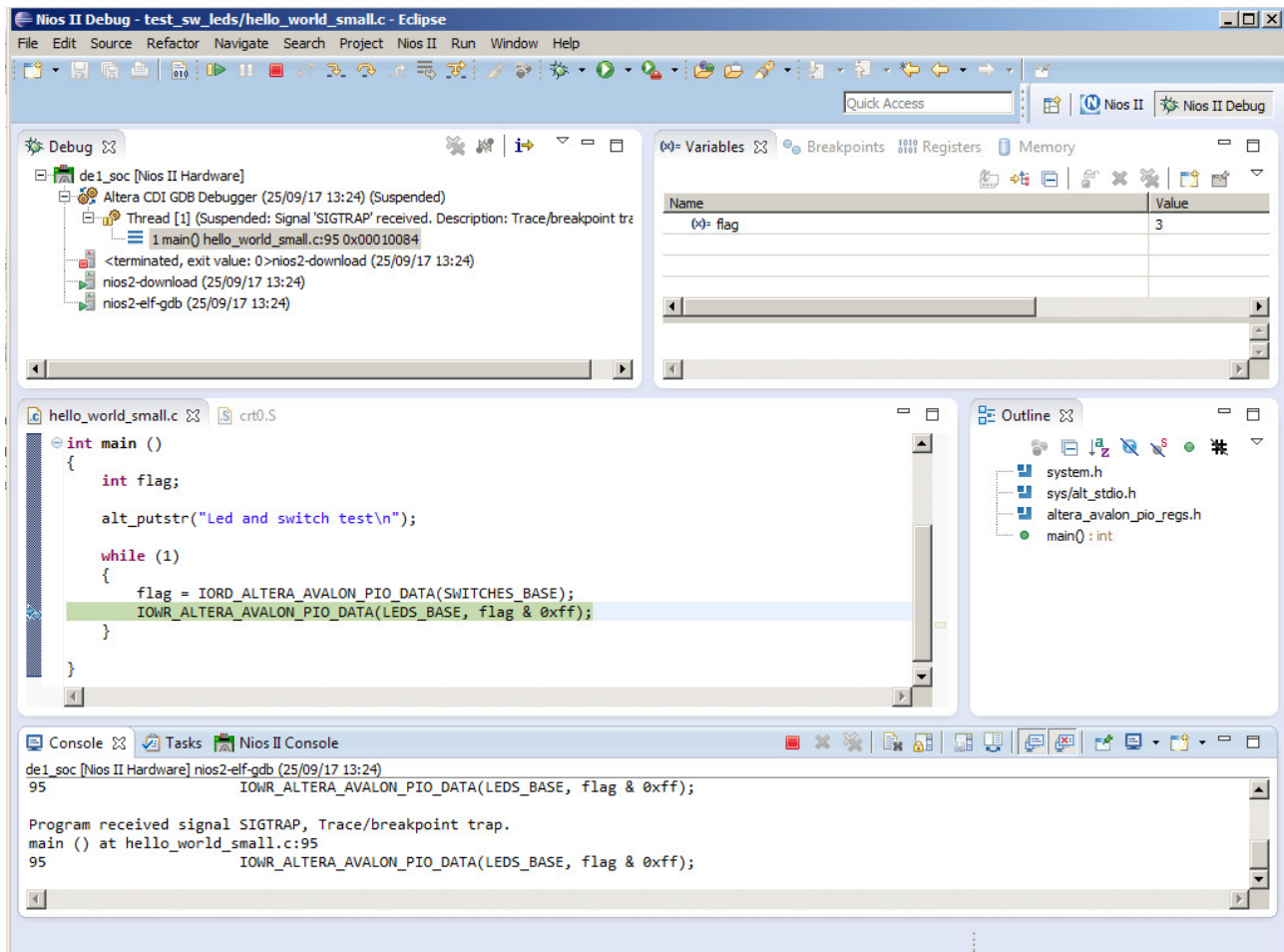


Figure 30 test\_sw\_leds debug

Part of this tutorial is derived from Intel-Altera document “*Introduction to the Qsys System Integration Tool*” which is available on “Portale della didattica”.

Part of this tutorial is derived from the “Nios EDF flow” material prepared by Prof. F. Gregoretti, DET, Politecnico di Torino.