

Laboratory Exercise 5

Implementation of an Embedded System

The purpose of this exercise is to learn how to create an embedded system and implement it in an FPGA device. The system will consist of an Altera Nios II processor and input/output interfaces that connect to switches and displays on an Altera DE-series board (intended for the boards listed in Table 1). We will use the Quartus II and Qsys software to generate the hardware portion of the system. We will use the *Altera Monitor Program* software to compile, load and run application programs. The background knowledge needed to do this exercise can be acquired from the tutorials: *Introduction to the Altera Nios II Soft Processor* and *Introduction to the Altera Qsys Tool*, which can be found in the University Program section of the Altera web site.

In this exercise, we will build a system that has some of the I/O capability of the DE-series Media Computer, sufficient to perform the I/O tasks performed in Laboratory Exercises 3 and 4. The desired system will include the parallel input/output interfaces (PIOs). Recall from Laboratory Exercises 3 and 4 that the PIO interface is a component that can be generated by using the Qsys tool. It provides for data transfer in either input or output (or both) directions. The transfer is done in parallel and it may involve from 1 to 32 bits. The number of bits, n , and the direction of transfer are specified by the user through Altera's Qsys tool (at the time a Nios II based system is being designed). The PIO interface can contain the four registers shown in Figure 1.

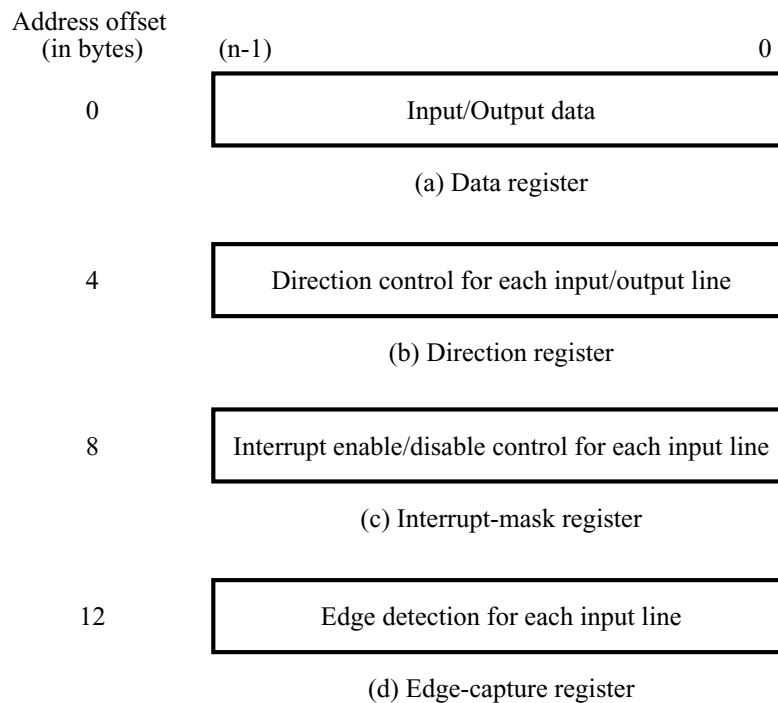


Figure 1. Registers in the PIO interface.

Each register is n bits long. The registers have the following purpose:

- *Data register* holds the n bits of data that are transferred between the PIO interface and the Nios II processor.

It can be implemented as an input, output, or a bidirectional register by the Qsys tool.

- *Direction* register defines the direction of transfer for each of the n data bits when a bidirectional interface is generated.
- *Interrupt-mask* register is used to enable interrupts from the input lines connected to the PIO.
- *Edge-capture* register indicates when a change of logic value is detected in the signals on the input lines connected to the PIO.

Not all of these registers are generated in a given PIO interface. For example, the *Direction* register is included only when a bidirectional interface is specified. The *Interrupt-mask* and *Edge-capture* registers are included if interrupt-driven input/output is used.

The PIO registers are accessible as if they were memory locations. Any base address that has the four least-significant bits equal to 0 can be assigned to a PIO (at the time it is implemented by the Qsys tool). This becomes the address of the *Data* register. The addresses of the other three registers have offsets of 4, 8, or 12 bytes (1, 2, or 3 words) from this base address.

As in Labs 3 and 4, the application task in this exercise consists of adding together a set of signed 8-bit numbers that are entered via the slider switches on the DE-series board. The resulting sum is displayed on the LEDs and 7-segment displays. The exercise makes use of both polling and interrupt I/O schemes.

Part I

In this part we will use the Qsys tool to design a Nios II based system that can be implemented in the FPGA on the DE-series board. We will use switches and LEDs on the board as input and output devices. Use 8 slider switches, SW_{7-0} , as inputs for entering numbers. Use the green lights, *LEDG*, to display the number selected by the switches. Use the red lights, *LEDR*, to display the accumulated sum as a binary number. Use the 7-segment displays, *HEX3-HEX0*, to display the sum as a hexadecimal number. A Nios II system which includes five PIO interfaces is the hardware needed for our task. One PIO circuit, connected to the slider switches, will provide the input data that can be read by the processor. Three PIO circuits, connected to the LEDs and HEX displays, will serve as the output interfaces to display the input number and the accumulated sum.

To provide a control signal for use in both polling and interrupt schemes, we will include a one-bit PIO circuit that will provide the functionality of a status flag and an ability to raise interrupt requests.

Realize the required hardware as follows:

1. Create a Quartus II project. Select the FPGA Device for your DE-series board. Refer to Table 1 for a list of devices on the DE-series boards that can be used in this exercise.

Board	Device Name
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1: DE-series FPGA device names

2. Use the Qsys tool to generate the desired circuit, called *nios_system*, which comprises:
 - On-chip memory - RAM mode and 16 Kbytes in size (leave all other options at their default settings)
 - Nios II/s processor with JTAG Debug Module Level 1
 - Do **not** choose the Hardware Multiply and Hardware Divide options

- Choose on-chip memory as the location for Reset and Exception vectors, as indicated in Figure 2
- Leave all other options for the processor at their default settings

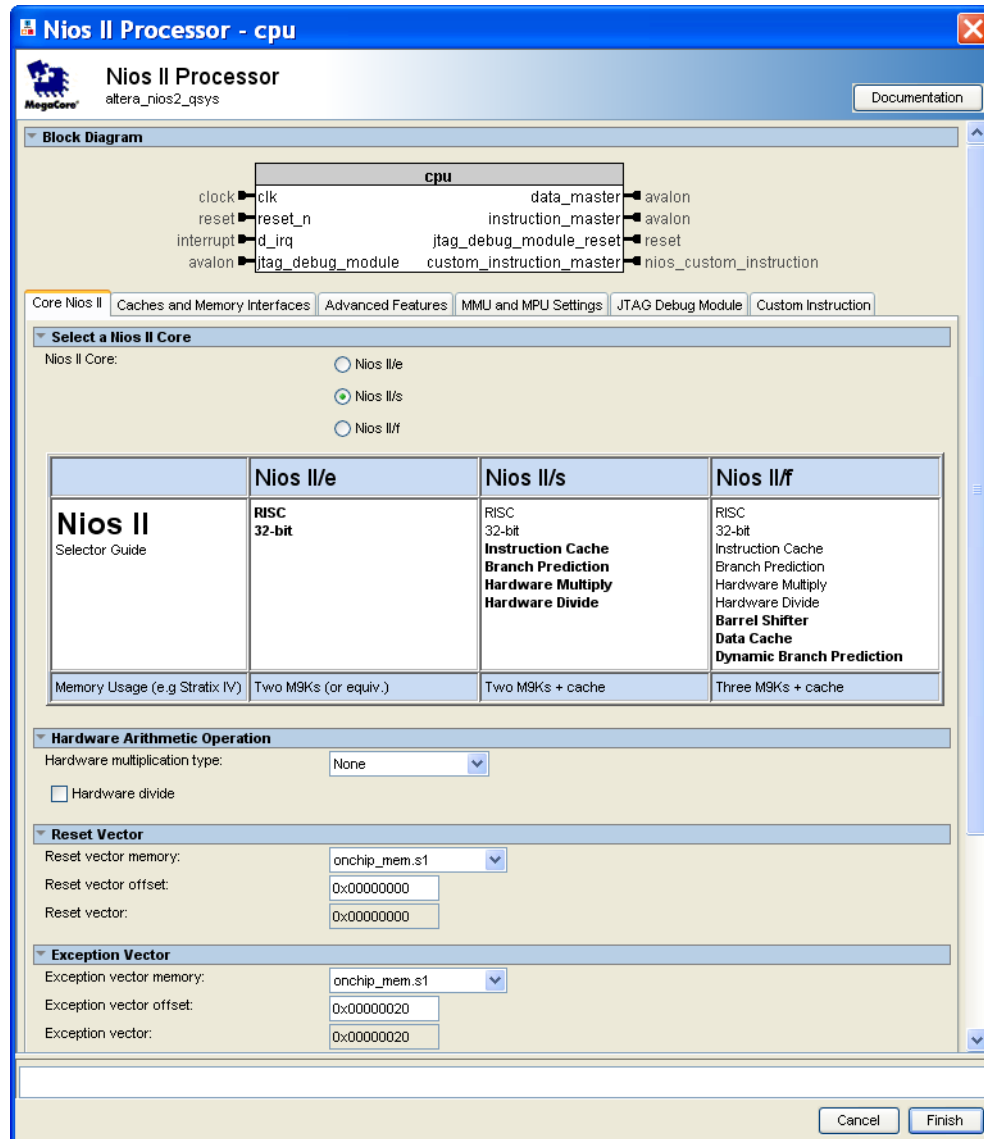


Figure 2. The Qsys specification of the Nios II processor.

- An 8-bit PIO input circuit, which will be connected to slider switches (The PIO components are found by selecting **Peripherals > Microcontroller Peripherals > PIO**.)
- An 8-bit PIO output circuit, which will be connected to green LEDs
- A 16-bit PIO output circuit, which will be connected to red LEDs
- A 32-bit PIO output circuit, which will be connected to HEX displays
- A one-bit PIO circuit that will serve as a status flag, which will be connected to the pushbutton key *KEY₁*. Configure it to be an input port that is one-bit wide. Also, select the following:
 - Synchronously capture feature activated by the Falling edge for the *Edge capture* register.

- Generate IRQ interrupt on Edge
- as indicated in Figure 3.

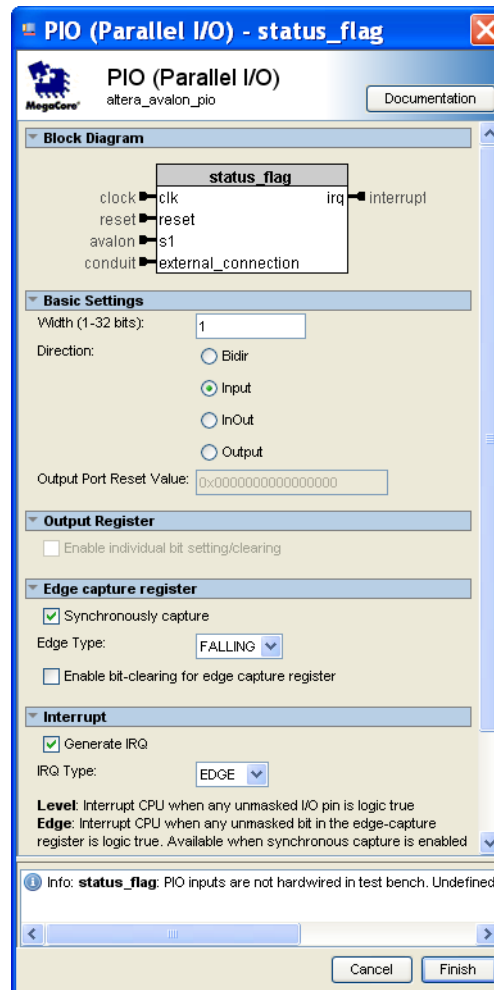


Figure 3. Specification for the status-flag PIO.

- The Qsys tool will automatically assign the names such as *pio*, *pio_1*, *pio_2*, ... to these PIO components. Change these names to something that is more meaningful in the context of a specific design. For example, we can choose the names *new_number*, *green_LEDs*, *red_LEDs*, *hex_displays* and *status_flag*.
 - Make the necessary connections among the components of the specified system, as indicated in Figure 4.
 - Having specified that the *status_flag* PIO can raise an interrupt request, it is necessary to specify the level (IRQ #) for this interrupt. This is done in the main Qsys window as illustrated in Figure 4. In the rightmost column, which is labeled IRQ, specify 1 as the desired level. Of course, the choice of level 1 is arbitrary. This choice will cause the bit position b_1 in control registers *ctl3* (*ienable*) and *ctl4* (*ipending*) to be associated with the *status_flag* PIO.
 - We wish the on-chip memory to occupy addresses that start with the address 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. Let Qsys assign the rest of the addresses by selecting System < Assign Base Addresses.
- Figure 4 shows the resulting system specification.

7. Observe (and record for future reference) the assigned addresses. Select the **Generation** tab and click on the **Generate** button to generate the specified system, which will be produced in a Verilog file *nios_system.v*.
8. Write, in Verilog or VHDL, a top-level module that instantiates the generated *nios_system* circuit and also defines the required connections to the switches and LEDs on the DE-series board. We will place this module in the file *simple_computer.v/vhd*. Connect the *reset* input in the generated *nios_system* module to the pushbutton switch *KEY₀*. Use the pushbutton *KEY₁* as the input to the status-flag PIO. Keep in mind that the pushbutton switches are active low.
9. Assign the pins needed to make the necessary connections, by importing the *qsf* pin-assignment file for your board.
10. Add the *nios_system.qip* file (IP Variation file) to your Quartus II project and then compile the project.

System Contents									
Address Map									
Clock Settings									
Project Settings									
System Inspector									
HDL Example									
Generation									
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	clk_0				
		clk_in	Clock Input	reset					
		clk_in_reset	Reset Input	reset					
		clk	Clock Output	reset					
		clk_reset	Reset Output	reset					
<input checked="" type="checkbox"/>		onchip_memory	On-Chip Memory (RAM or ROM)	clk	clk_0				
		clk1	Clock Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		reset1	Reset Input	clk					
<input checked="" type="checkbox"/>		nios2_processor	Nios II Processor	clk	clk_0				
		clk	Clock Input	clk					
		reset_n	Reset Input	clk					
		data_master	Avalon Memory Mapped Master	clk					
		instruction_master	Avalon Memory Mapped Master	clk					
		jtag_debug_module_re...	Reset Output	clk					
		jtag_debug_module	Avalon Memory Mapped Slave	clk					
		custom_instruction_m...	Custom Instruction Master	clk					
<input checked="" type="checkbox"/>		new_number	PIO (Parallel I/O)	clk	clk_0				
		clk	Clock Input	clk					
		reset	Reset Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		external_connection	Conduit Endpoint	new_number		0x00005000	0x0000500f		
<input checked="" type="checkbox"/>		green_LEDs	PIO (Parallel I/O)	clk	clk_0				
		clk	Clock Input	clk					
		reset	Reset Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		external_connection	Conduit Endpoint	green_leds		0x00005010	0x0000501f		
<input checked="" type="checkbox"/>		red_LEDs	PIO (Parallel I/O)	clk	clk_0				
		clk	Clock Input	clk					
		reset	Reset Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		external_connection	Conduit Endpoint	red_leds		0x00005020	0x0000502f		
<input checked="" type="checkbox"/>		hex_displays	PIO (Parallel I/O)	clk	clk_0				
		clk	Clock Input	clk					
		reset	Reset Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		external_connection	Conduit Endpoint	hex_displays		0x00005030	0x0000503f		
<input checked="" type="checkbox"/>		status_flag	PIO (Parallel I/O)	clk	clk_0				
		clk	Clock Input	clk					
		reset	Reset Input	clk					
		s1	Avalon Memory Mapped Slave	clk					
		external_connection	Conduit Endpoint	status_flag		0x00005040	0x0000504f		

Figure 4. The Nios II system implemented by the Qsys tool.

In the next four parts you will use the designed computer to investigate different aspects of performing I/O tasks, using both polling and interrupt approaches. In Parts II and III, the necessary programs are to be written in the Nios II assembly language. In Parts IV and V, the programs are to be written in the C language. You should use the Altera Monitor Program to handle your programs. For each part you should create a new project in the monitor program.

Part II

In this part, we will use the polling approach to read the numbers entered via the slider switches. The desired operation is that the user provides the next number by setting the slider switches accordingly and then pressing the pushbutton KEY_1 to indicate that the number is ready for reading.

To accomplish this task it is necessary to implement a mechanism that monitors the status of the circuit used to input the numbers. A commonly-used I/O scheme, known as *polling*, is to use a *status flag* which is originally cleared to 0. This flag is then set to 1 as soon as the I/O device interface is ready for the next data transfer. Upon transferring the data, the flag is again cleared to 0. Thus, the processor can *poll* the status flag to determine when an I/O data transfer should be made.

In our case, the I/O device is the user who manually sets the slider switches and presses the pushbutton key. The I/O interface that provides the desired control is the one-bit status-flag PIO circuit generated in Part I, which includes the edge-capture capability and conforms to the register map in Figure 1.

Perform the following:

1. Write an assembly-language program that reads the contents of the switches when the pushbutton KEY_1 is pressed, and displays this number on the green LEDs. Then, it adds the number to a sum that is being accumulated, and displays the sum on the red LEDs and HEX displays. Save the program in a file *lab5_part2.s*.
When KEY_1 is pressed, the circuit designed in Part I will set the status-flag bit in the *edge-capture* register to 1. After reading the new number, your program has to clear the flag by writing a 0 into the *edge-capture* register.
2. Open the Altera Monitor Program and create a new project, as illustrated in Figure 5.
3. Specify that you wish to use the hardware that you designed, by choosing **Custom System** as shown in Figure 6. Find the file *nios_system.qsys*, which represents the designed Nios II system. Also, select the file *simple_computer.sof* which provides the information needed to download the designed system into the FPGA chip on the DE-series board.
4. Specify that an assembly-language program is to be used and that the program is given in the file *lab5_part2.s*, as shown in Figures 7 and 8, respectively.
5. Make sure that the USB-Blaster is used to provide the connection between the DE-series board and the host computer, as indicated in Figure 9.
6. Specify that your program has to be loaded in the on-chip memory, as illustrated in Figure 10. Since your system does not include any other memory, this choice will be made by default.
7. Click **Finish** in the window in Figure 10 and when a pop-up box asks you if you want to have your system downloaded onto the DE-series board click **Yes**.

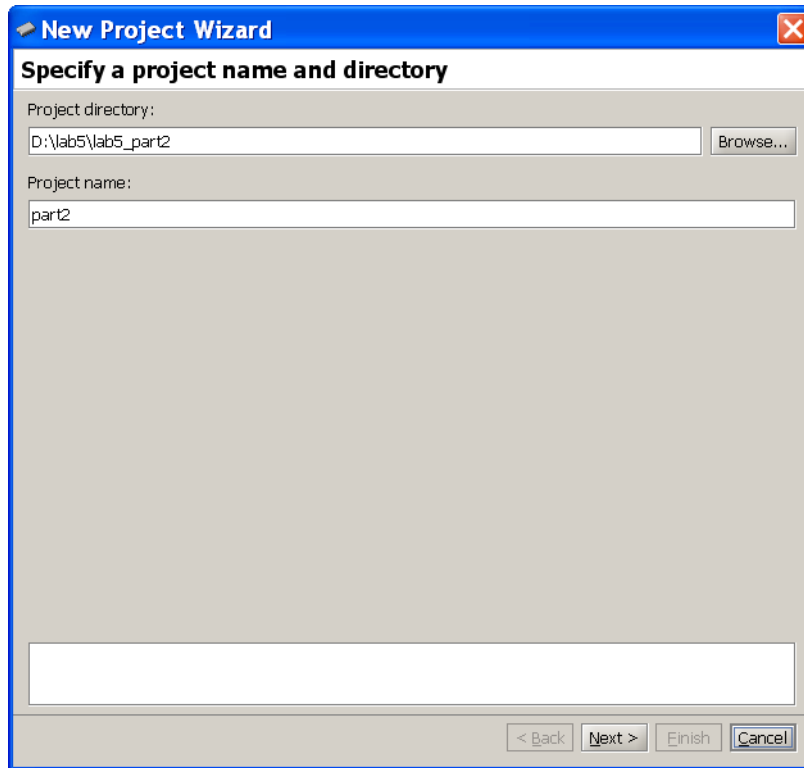


Figure 5. Create a new project in the monitor program.

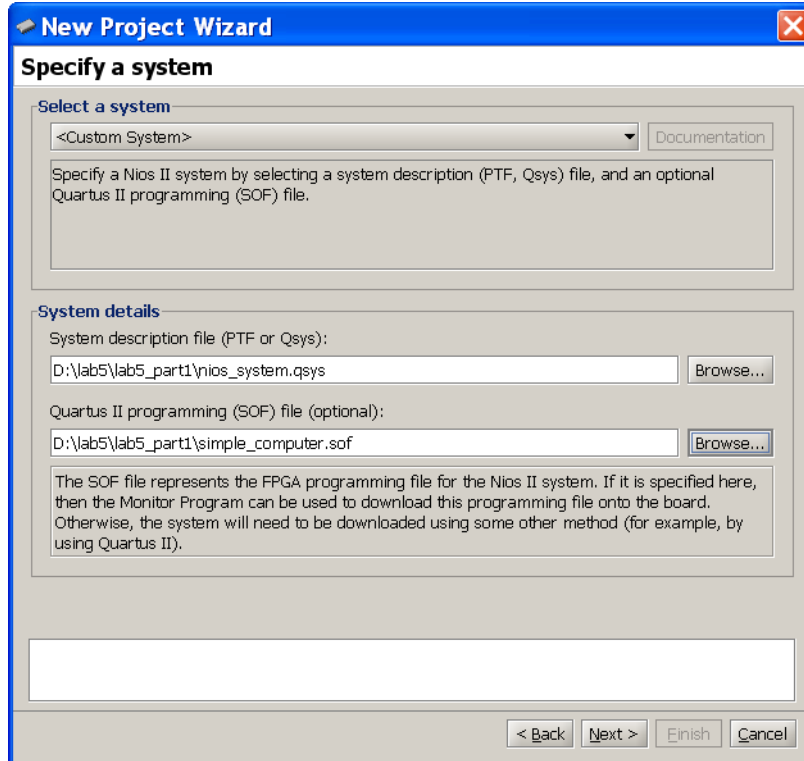


Figure 6. Select the custom Nios II system that you designed.

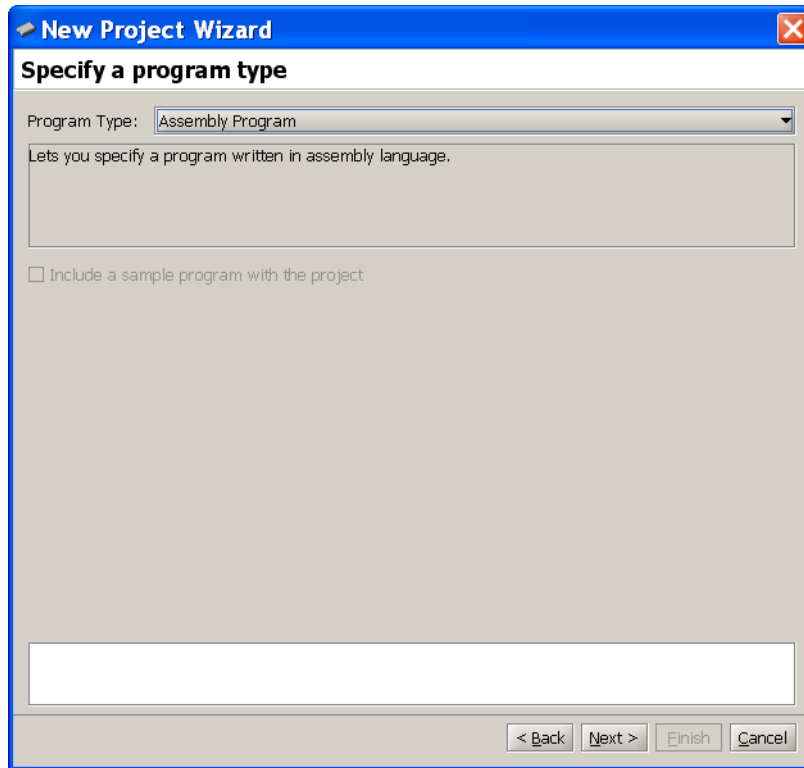


Figure 7. Specify that an assembly-language program is used.

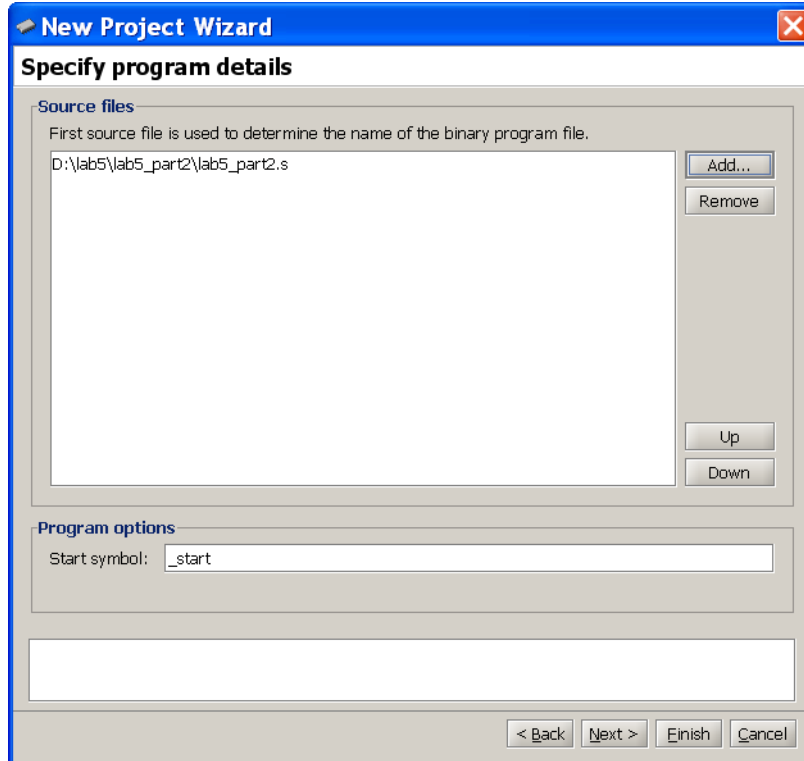


Figure 8. Specify the file that contains the application program.

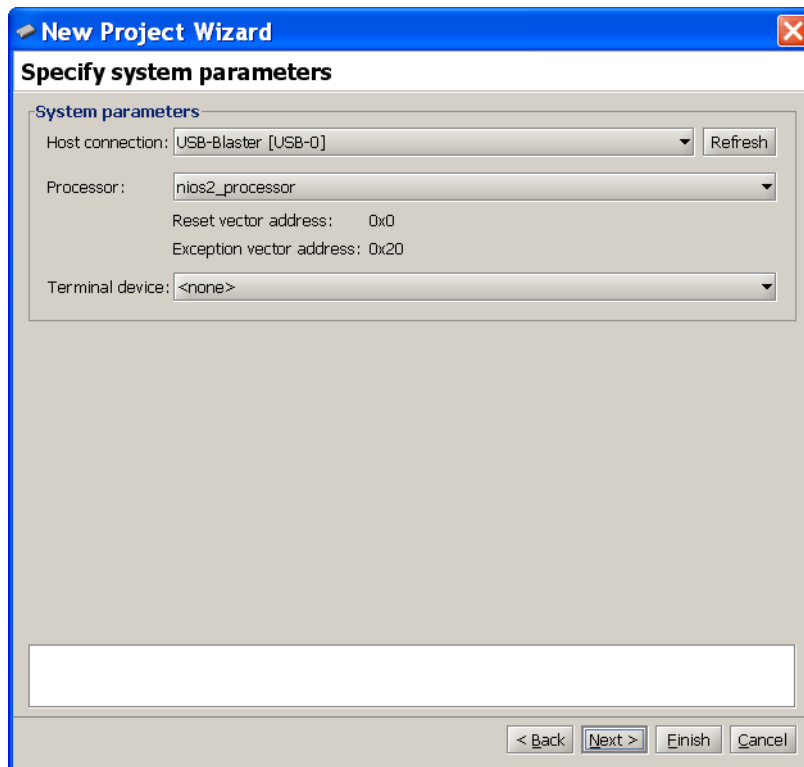


Figure 9. Specify the system parameters.

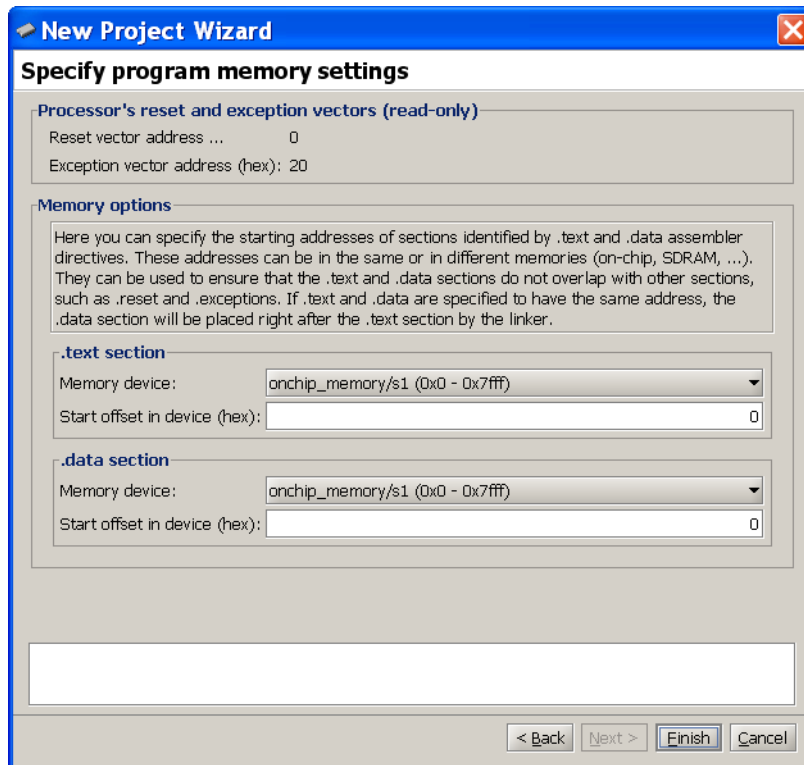


Figure 10. Specify where the program will be loaded in the memory.

8. Now, in the monitor window select **Actions > Compile & Load** to assemble and download your program.
9. Run the program and verify its correctness by inputting several numbers. The program should run continuously and a new number should be added each time the pushbutton KEY_1 is pressed.

Part III

Instead of using the polling approach to read new numbers from the slider switches, we now want to use interrupts for the same purpose. To accomplish this, we will use the ability of the status-flag PIO to raise an interrupt request when the pushbutton KEY_1 is pressed.

Modify your assembly-language program to realize the desired task by using the interrupt approach.

Part IV

Write a C-language program to perform the task in Part II.

Part V

Write a C-language program to perform the task in Part III.

Final Notes: Consult the description of Laboratory Exercise 4 for details on how interrupts should be handled in both the assembly-language and the C-language programs.

The system that you designed in Part I has many similarities with the DE-series Media Computer, but it is not identical. This means that the programs that you wrote for Labs 3 and 4 may need some modifications to run successfully on your system.

Preparation

Your preparation should include the following:

1. System design for Part I
2. Programs for Parts II to V

Copyright ©2011 Altera Corporation.