Laboratory Exercise 5

Implementation of a Computer System

The purpose of this exercise is to learn how to create a computer 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. We will use the Quartus II and Qsys Tool 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 System Integration Tool*, which can be found in the University Program section of the Altera website.

In this exercise, we will build a system that is simpler than the DE-series Basic Computer, but which has similar parallel I/O capabilities. The desired system will have to support the application tasks encountered in Lab 4, which requires the use of parallel input/output interfaces (PIOs). Recall from Lab 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 the 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.

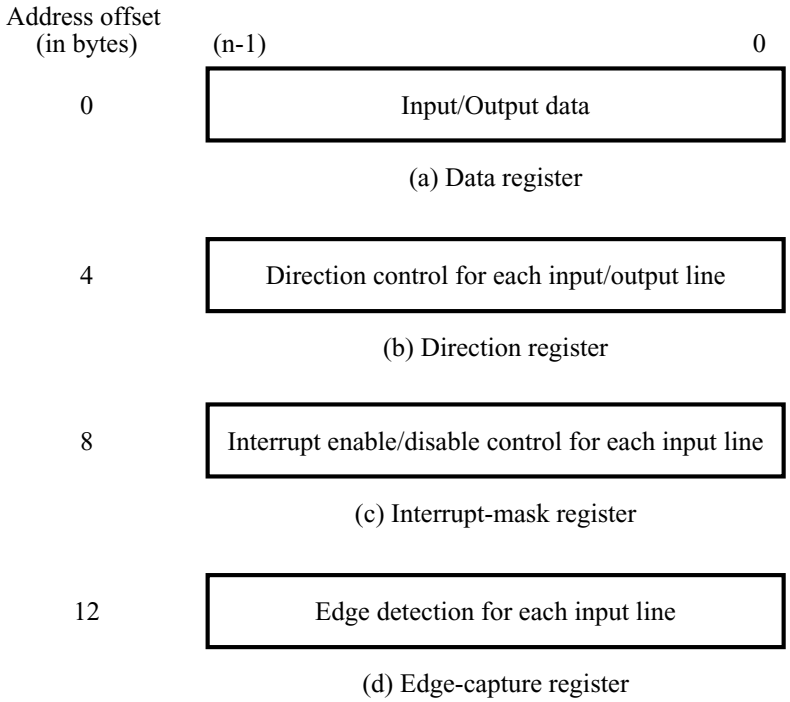

Figure 1. Registers in the PIO interface.

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 Lab 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 a 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, $LEDG_{7-0}$, to display the number defined by the slider switches. Use the 16 red lights, $LEDR_{15-0}$, 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. Table 1 gives a list of devices on the DE-series boards.

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

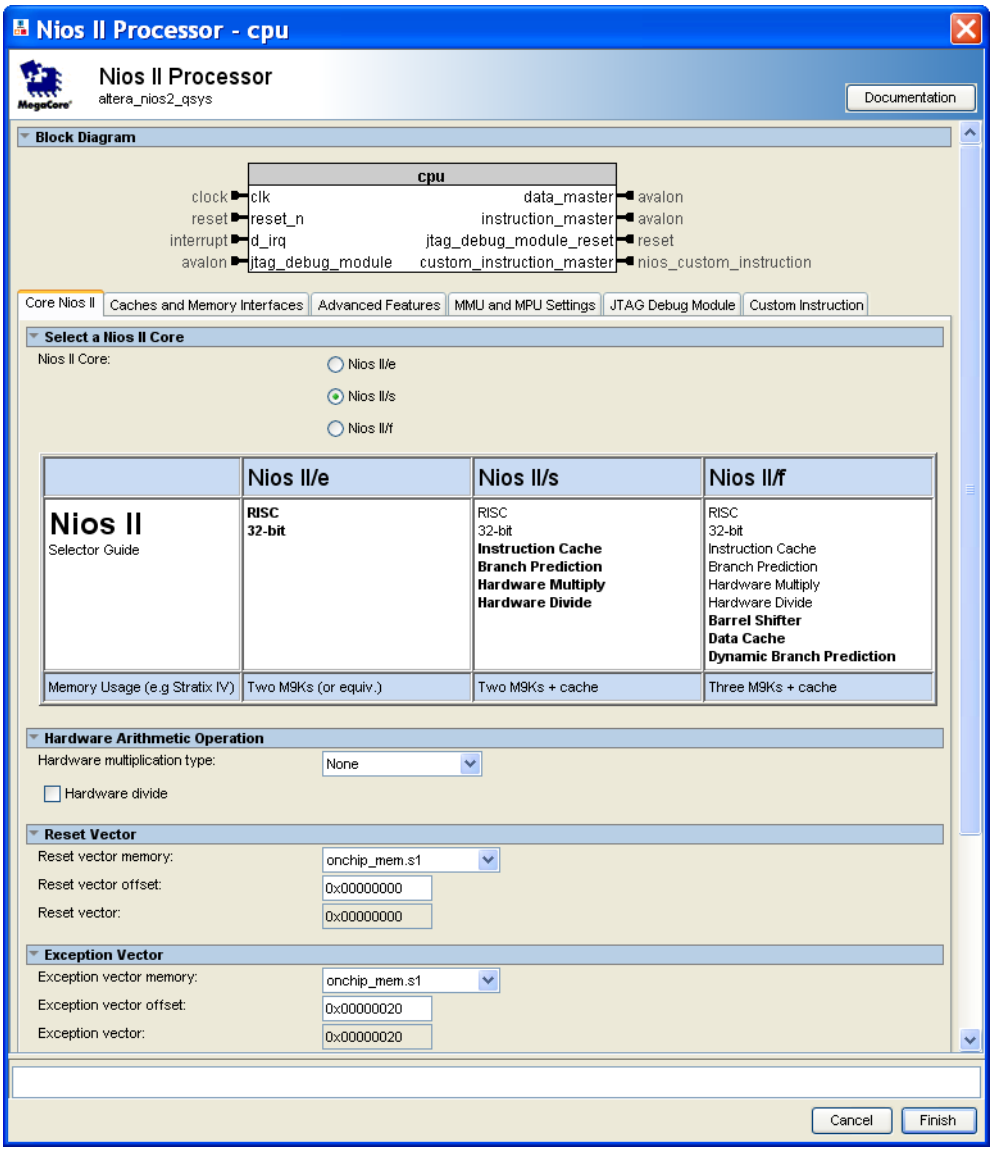

Figure 2. The Qsys specification of the Nios II processor.

- 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 KEY_1. 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.

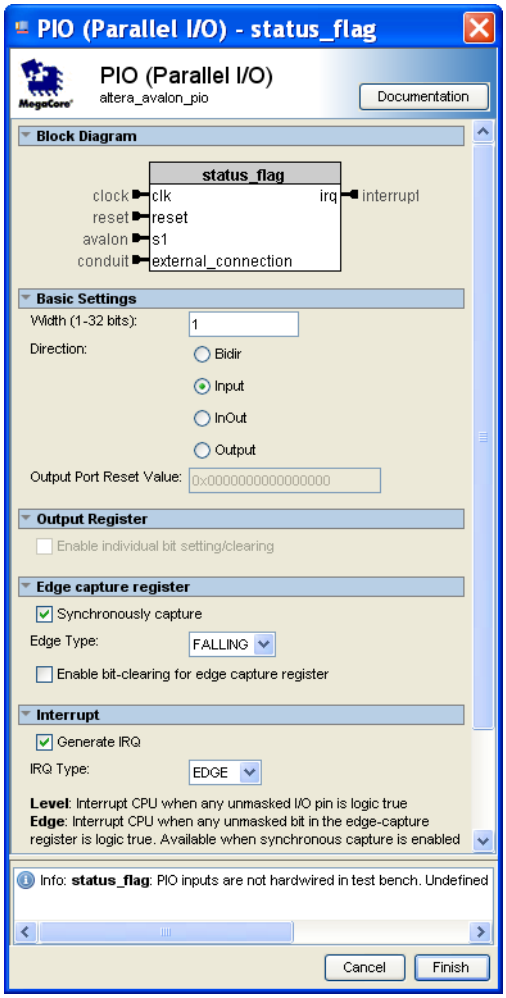

Figure 3. Specification for the status-flag PIO.

3. 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*.

4. For now, ignore any errors that might say that addresses overlap.

5. Make the necessary connections among the components of the specified system, as indicated in Figure 4.

6. 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.

7. 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.

8. 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*.

9. 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 KEY_0. Use the pushbutton KEY_1 as the input to the status-flag PIO. Keep in mind that the pushbutton switches are active low.

10. Assign the pins needed to make the necessary connections, by importing the *qsf* pin-assignment file for your board.

11. Add the *nios_system.qip* file (IP Variation file) to your Quartus II project and then compile the project.

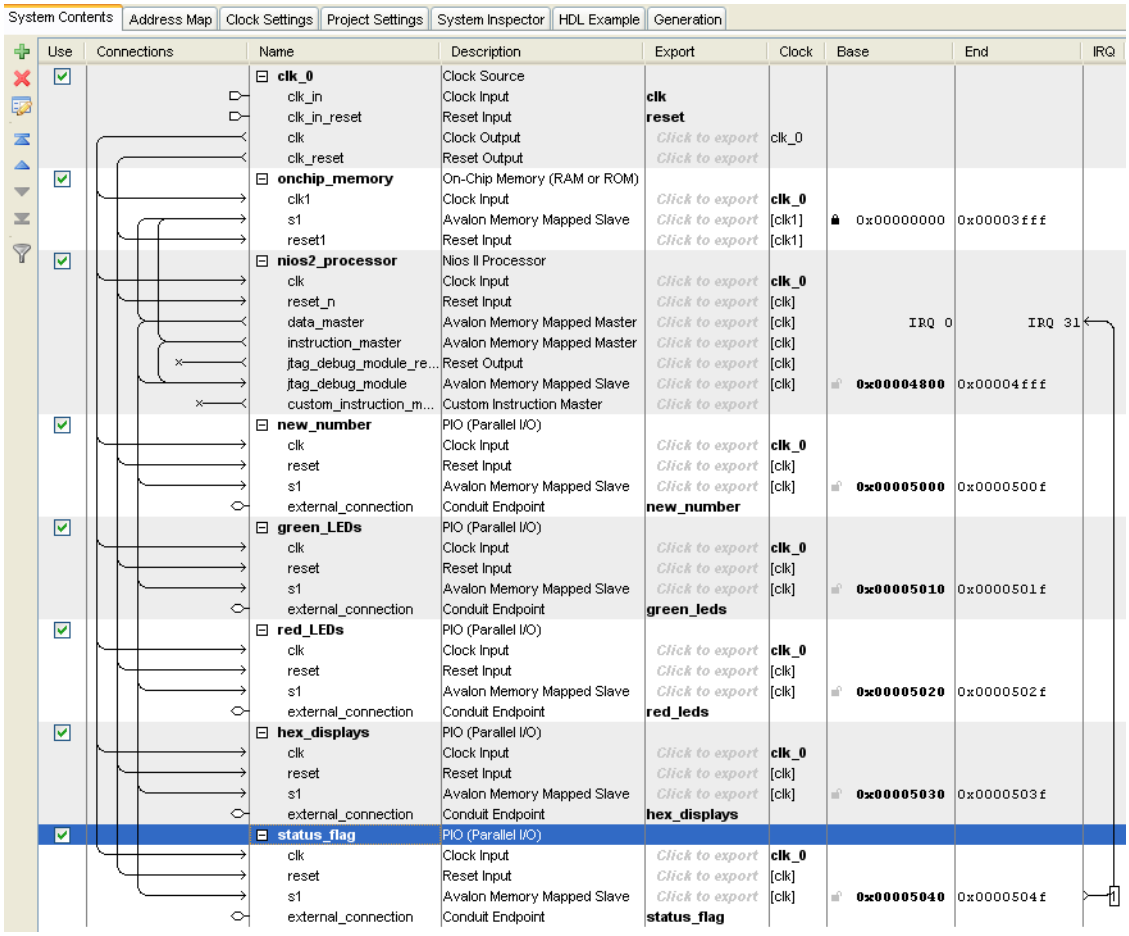

Figure 4. The Nios II system implemented by the Qsys Tool.

In the next three parts we will use the designed computer to investigate different aspects of performing I/O tasks. We will use the Altera Monitor Program to handle the assembly language code written by you to perform each task. For each part you should create a new project in the monitor program.

Part II

In this part we will not use the status flag. Perform the following:

1. Write a program that reads the contents of the switches, displays the corresponding value on the green LEDs, adds this 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*.
2. Open the *Altera Monitor Program* software and create a new project, as illustrated in Figure 5.

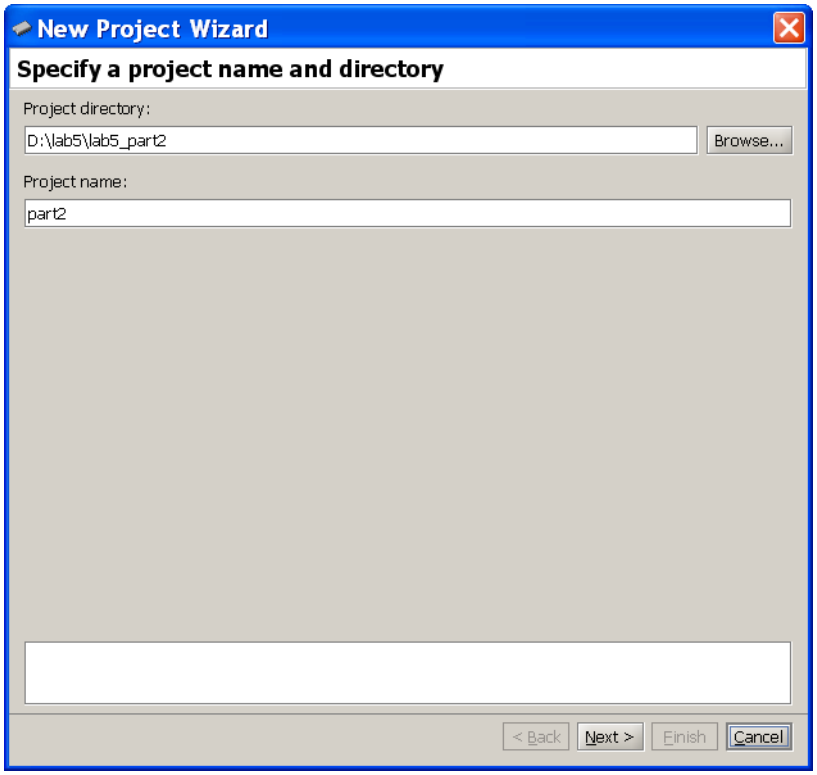

Figure 5. Create a new project in the monitor program.

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.

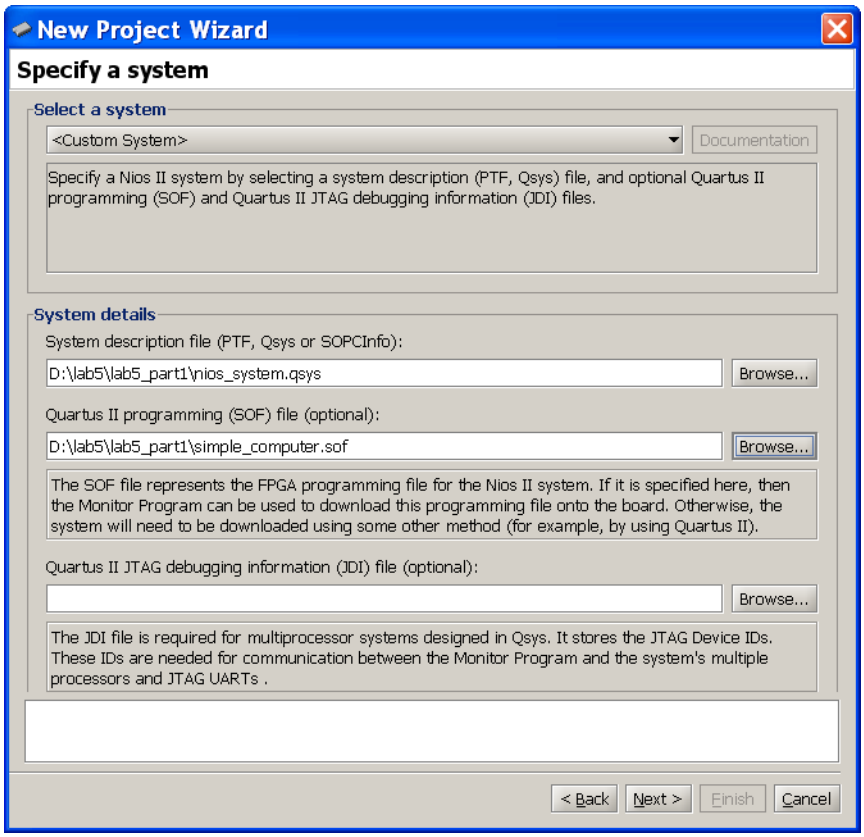

Figure 6. Select the custom Nios II system that you designed.

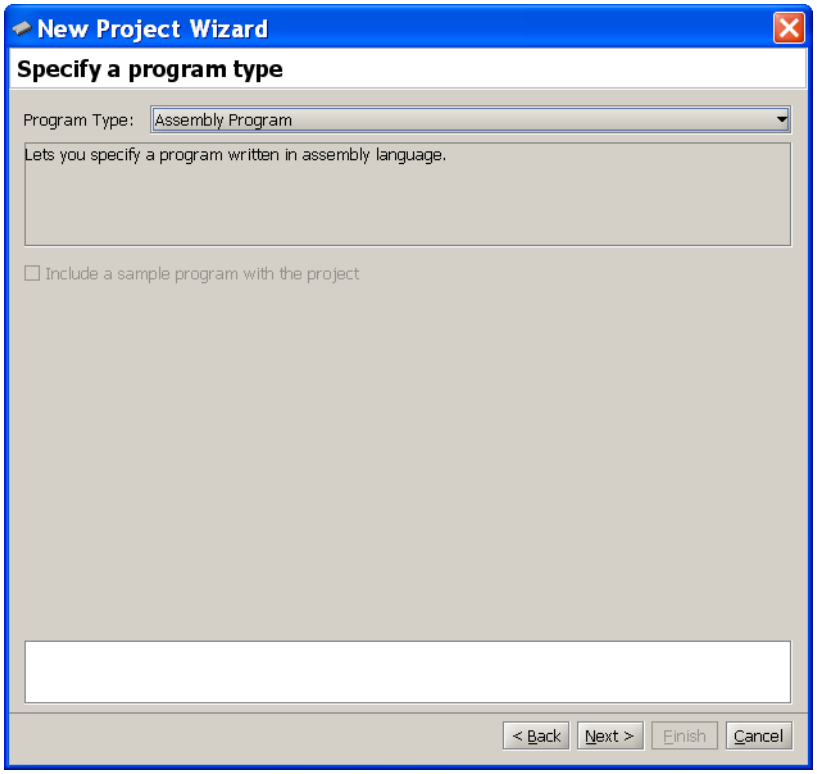

Figure 7. Specify that an assembly-language program is used.

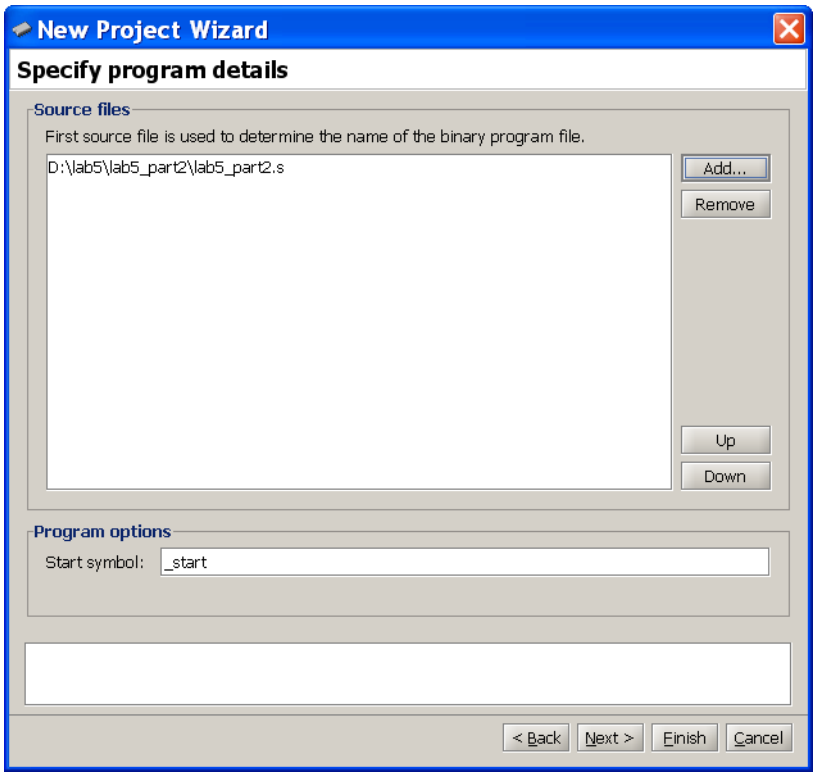

Figure 8. Specify the file that contains the application program.

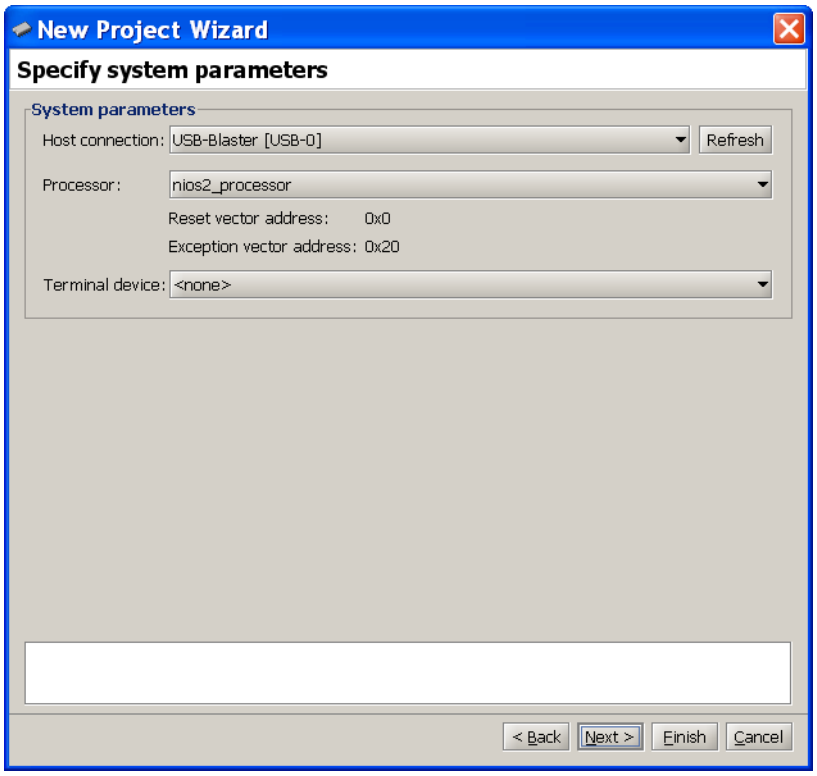

Figure 9. Specify the system parameters.

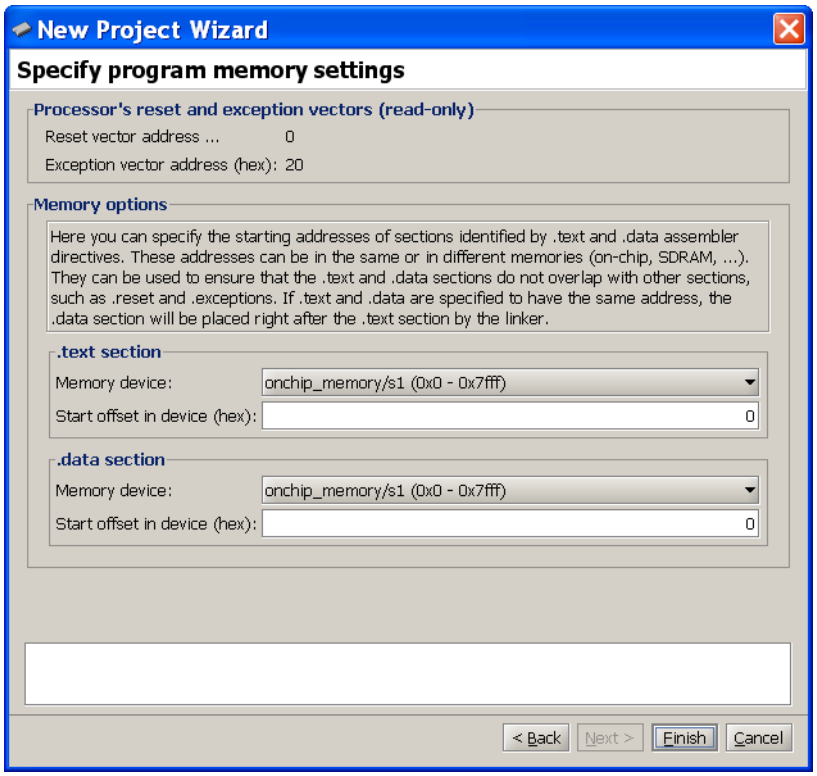

Figure 10. Specify where the program will be loaded in the memory.

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

9. Single-step through the program and verify its correctness by inputting several numbers. Note that single-stepping through the program will allow you to change the input numbers without reading the same number multiple times.

10. Now, run the program and then stop it. Observe what happens and explain the observed behavior.

11. Run the program again and then press the *Reset* pushbutton KEY_0. What happens in this case?

Part III

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. Modify your application program from Part II to accept a new number when the pushbutton KEY_1 is pressed. This action 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. Download and run your program to demonstrate that it works properly. The program should run continuously and a new number should be added each time the pushbutton KEY_1 is pressed.

Part IV

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 when the pushbutton KEY_1 is pressed.

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

Final Note: The system that you designed in all parts of this lab exercise has many similarities with the DE-series Basic Computer, but it is not identical. This means that the assembly language programs that you wrote for Lab 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. Assembly language programs for Parts II to IV