Laboratory Exercise 2

Program-Controlled Input/Output

The purpose of this exercise is to investigate the use of devices that provide input and output capabilities for a processor, and are controlled by software. We will examine these program-controlled I/O operations from both the hardware and software points of view. We will make use of parallel interfaces, *PIOs*, in a Nios II system implemented on an Altera DE2 board. 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 SOPC Builder*, which can be found in the University Program section of the Altera web site.

The PIO interface used in this exercise, which is a component that can be generated by using the SOPC Builder, 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 SOPC Builder. The PIO interface can contain the four registers shown in Figure 1.

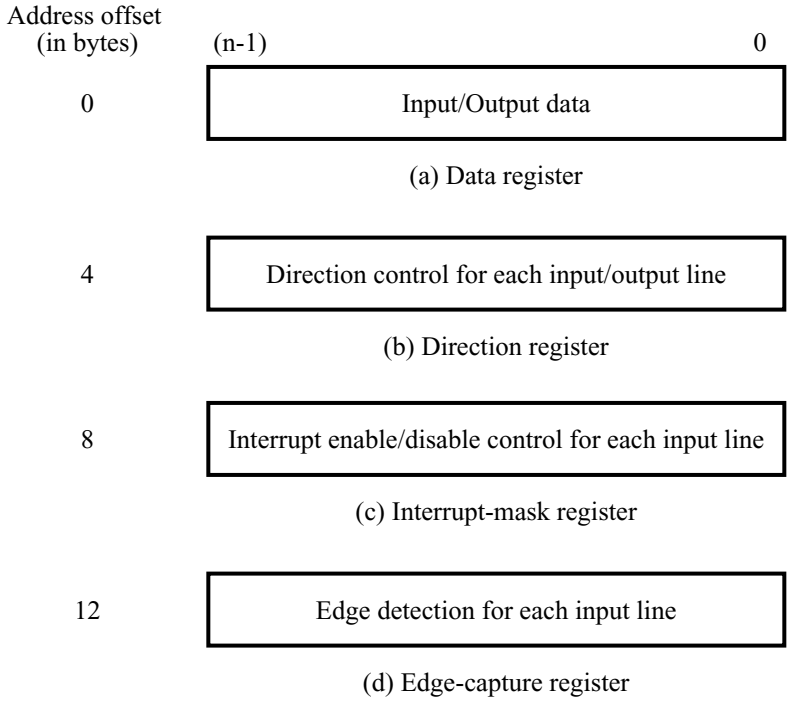

Figure 1. The registers in the PIO interface.

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 SOPC Builder.
- *Direction* register defines the direction of the 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 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 (this may be done automatically by the SOPC Builder). 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). A full description of the PIO module can be found in the document *PIO Core with Avalon Interface*, which is available in the literature section of Altera's web site.

The application task in this exercise consists of adding together a set of signed 8-bit numbers that are entered via the toggle switches on Altera's DE2 board. The resulting sum is displayed on the LEDs and 7-segment displays.

Part I

Use 8 toggle switches, SW_{7-0}, as inputs for entering numbers. Use the green lights, $LEDG_{7-0}$, to display the number defined by the toggle switches. Use the 16 red lights, $LEDR_{15-0}$, to display the accumulated sum. A Nios II system, which includes three PIO interfaces, is the hardware needed for our task. One PIO circuit, connected to the toggle switches, will provide the input data that can be read by the processor. Two other PIO circuits, connected to the green and red lights, will serve as the output interfaces to allow displaying the number selected by the switches and the accumulated sum, respectively.

Realize the required hardware by implementing a Nios II system on the DE2 board, as follows:

1. Create a new Quartus II project. Select Cyclone II EP2C35F672C6 as the target chip, which is the FPGA chip on the Altera DE2 board.
2. Use the SOPC Builder to generate the desired circuit, called *nios_system*, which comprises:
   - Nios II/s processor with JTAG Debug Module Level 1; select the following options:
     - Embedded Multipliers for Hardware Multiply
     - Hardware Divide
   - On-chip memory - RAM mode and 32 Kbytes in size
   - An 8-bit PIO input circuit
   - An 8-bit PIO output circuit
   - A 16-bit PIO output circuit

   The SOPC Builder will automatically assign the names such as *pio_0*, *pio_1* and *pio_2* to the three PIO components. You can 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* and *red_LEDs*.
3. From the System menu, select Auto-Assign Base Addresses. This will assign addresses to all components in the designed system. The result will be a system such as the one shown in Figure 2. Observe the assigned addresses.
4. Instantiate the generated *nios_system* within a Verilog/VHDL file, which also defines the required connections to the switches and LEDs on the DE2 board.
5. Assign the pins needed to make the necessary connections, by importing the pin-assignment file *DE2_pin_assignments_csv*.

6. Compile the Quartus II project.
7. Program and configure the Cyclone II FPGA on the DE2 Board to implement the generated system.

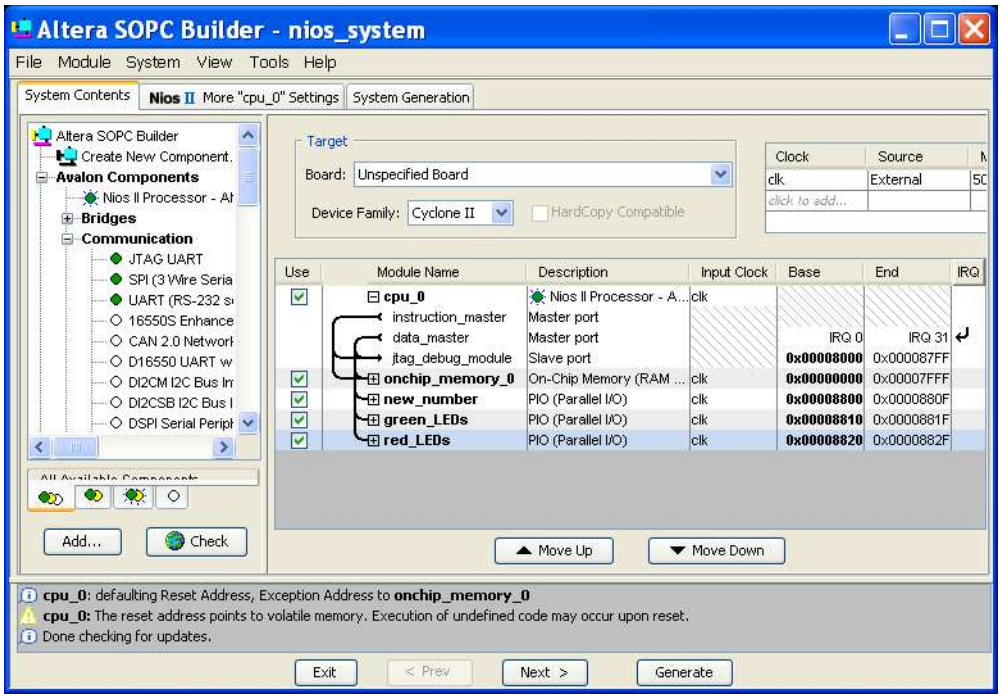

Figure . The Nios II system implemented by the SOPC Builder.

Part II

Implement the desired task using the Nios II assembly language, as follows:

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.
2. Use the *Altera Debug Client* software to assemble and download your program.
3. Single-step through the program and verify its correctness by inputting several numbers. Note that single-stepping through the program will allow you change the input numbers without reading the same number multiple times.

Part III

In this part, we want to add the ability to run the application program continuously and control the reading of new numbers by including a pushbutton switch which is activated by the user when a new number is ready to be read. The desired operation is that the user provides the next number by setting the toggle switches accordingly and then pressing a pushbutton switch 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 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 can be made.

In our case, the I/O device is the user who manually sets the toggle switches. The I/O interface is a PIO circuit generated by the SOPC Builder. To provide a status flag, we will generate a special one-bit PIO circuit and use its edge-capture capability. This PIO is very similar to the regular PIO, and it conforms to the register map in Figure 1. It is defined in the directory *altera_up_avalon_DE2_pio*, which has to be included in your project. The directory can be obtained from the Altera University Program site

ftp://ftp.altera.com/up/pub/University_Program_IP_Cores/DE2_pio.zip

Perform the following steps:

1. Create a new Quartus II project and implement the same system as done in Part I.
2. Copy the *altera_up_avalon_DE2_pio* directory into your project directory. Open the SOPC Builder, which shows the existing Nios II subsystem. To make the *altera_up_avalon_DE2_pio* directory visible to the SOPC Builder, click in the File menu of the SOPC Builder on the item Refresh Component List. The DE2-PIO component will be listed under Avalon Components > University Program DE2 Board.
3. Generate a status-flag PIO using the *DE2-PIO* component. Configure it to be an input port that is one bit wide. Also, in the Input Options tab select the Synchronously capture feature activated by the Falling edge.
4. Generate the new Nios II subsystem.
5. Modify your Verilog/VHDL file that specifies the complete system. Use the pushbutton switch KEY_0 as the input to the status-flag PIO (the pushbutton switches are active low).
6. Do the pin assignment and compile the project.
7. Modify your application program to accept a new number when the pushbutton switch is pressed. This action will set the "status flag" bit in the *edge-capture* register to 1. After adding the number to the accumulated sum, your program has to clear the flag by writing a 0 into the *edge-capture* register.
8. 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 switch is pressed.

Part IV

In the previous parts the accumulated sum was displayed on the red LEDs. Now, augment your design to display this sum as a hexadecimal number on the 7-segment displays HEX3-HEX0, in addition to the red LEDs.

Part V

Augment your design and the application program to display the accumulated sum on the 7-segment displays as a decimal (rather than hexadecimal) number. The application program should do the necessary number conversion.

Note: You can use the div instruction only if you specified the Hardware Divide option in Part I.