Laboratory Exercise 4

Input/Output in an Embedded System

The purpose of this exercise is to investigate the use of devices that provide input and output capabilities for a processor. There are two basic techniques for dealing with I/O devices: program-controlled polling and interrupt-driven approaches. You will use the polling approach in this exercise, writing programs in the ARM assembly language. Your programs will be executed on an ARM Cortex A9 processor in the DE0-Nano-SoC Computer, implemented on an Altera DE0-Nano-SoC board. Parallel port interfaces, as well as a timer module, will be used as examples of I/O hardware.

A parallel port provides for data transfer in either the input or output direction. The transfer of data is done in parallel and it may involve from 1 to 32 bits. The number of bits, n, and the type of transfer depend on the specifications of the specific parallel port being used. The parallel port interface can contain the four registers shown in Figure 1.

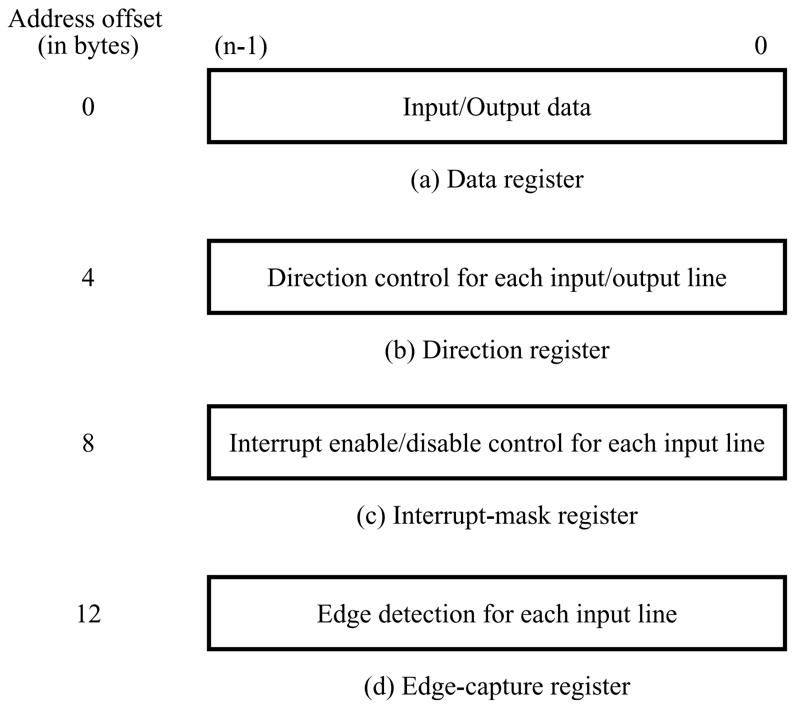

Address offset
(in bytes)
(n-1)
0
0
Input/Output data
(a) Data register
4
Direction control for each input/output line
(b) Direction register
8
Interrupt enable/disable control for each input line
(c) Interrupt-mask register
12
Edge detection for each input line
(d) Edge-capture register

Figure 1: Registers in the parallel port 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 parallel port and the ARM processor. It can be implemented as an input, output, or a bidirectional register.

- *Direction* register: defines the direction of transfer for each of the n data bits when a bidirectional interface is generated.
- *Interrupt-mask* register: used to enable interrupts from the input lines connected to the parallel port.
- *Edge-capture* register: indicates when a change of logic value is detected in the signals on the input lines connected to the parallel port. Once a bit in the edge capture register becomes asserted, it will remain asserted. An edge-capture bit can be de-asserted by writing to it using the ARM processor.

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

The parallel port registers are memory mapped, starting at a specific *base* address. The base address has to be a multiple of four if the parallel port is to be accessed using word accesses from the ARM processor. The base address becomes the address of the *Data* register in the parallel port. The addresses of the other three registers have offsets of 4, 8, or 12 bytes (1, 2, or 3 words) from this base address. The DE0-Nano-SoC Computer has parallel ports connected to slide switches, pushbutton KEYs, and LEDs.

Part I

Write an ARM assembly language program that displays a decimal digit on the green lights LED_{3-0} on the DE0-Nano-SoC board. The other lights LED_{7-4} should be off.

The parallel port in the DE0-Nano-SoC Computer connected to the green lights LED_{7-0} is memory mapped at the address `0xFF200000`. Figure 2 shows how the LEDs are connected to the parallel ports.

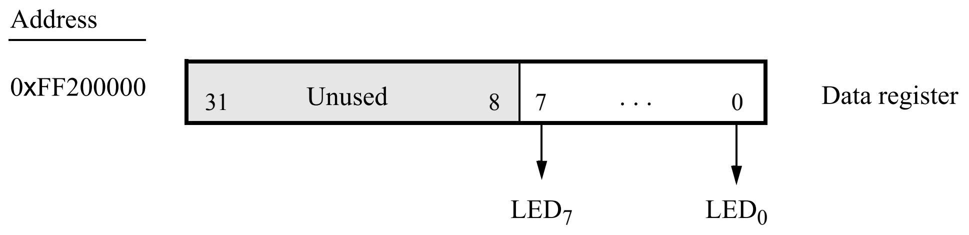


Figure 2: The parallel port connected to the green lights LED_{7-0}.

If KEY_0 is pressed on the DE0-Nano-SoC board, you should set the number displayed on the LEDs to 0. If KEY_1 is pressed and SW_0 is high, then increment the displayed number to a maximum of 9. If KEY_1 is pressed and SW_0 is low, then decrement the number to a minimum of 0. The parallel port connected to the pushbutton *KEYs* has the base address `0xFF200050`, as illustrated in Figure 3. The parallel port connected to the slider switches *SW* has the base address `0xFF200040`, as illustrated in Figure 4. In your program, use polled I/O to read the *Data* registers of the *KEY* and *SW* ports to check the status of the buttons and switches. When you are not pressing any *KEY* the *Data* register provides 0, and when you press KEY_i the *Data* register provides the value 1 in bit position i. Once a button-press is detected, be sure that your program waits until the button is released. You should not use the *Interruptmask* or *Edgecapture* registers for this part of the exercise.

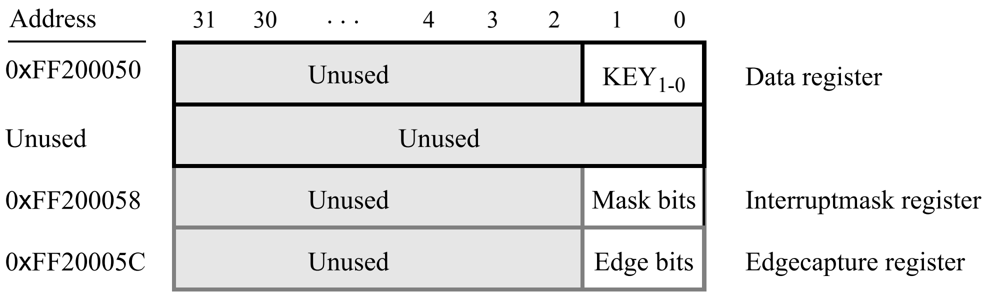


Figure 3: The parallel port connected to the pushbutton *KEYs*.

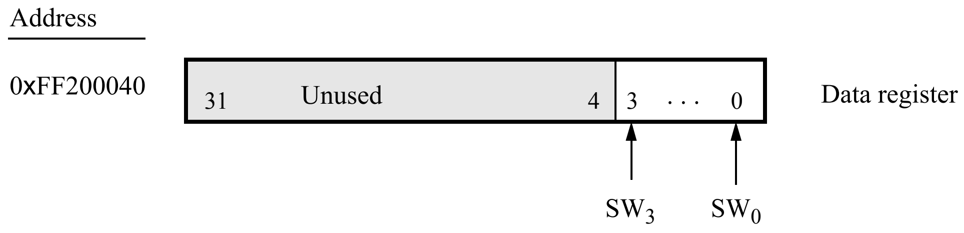


Figure 4: The parallel port connected to the slider switches *SW*.

Perform the following:

1. Create a new folder to hold your Monitor Program project for this part. Create a file called *part1.s* and type your assembly language code into this file.
2. Make a new Monitor Program project in the folder where you stored the *part1.s* file. Use the DE0-Nano-SoC Computer for this project, and select the ARM A9 as the target processor architecture.
3. Compile, download, and test your program.

Part II

Write an ARM assembly language program that displays a two-digital decimal counter on the green LEDs. Show the most-significant decimal digit on LED_{7-4}, and the least-significant digit on LED_{3-0}. The counter should be incremented approximately every 0.25 seconds. When the counter reaches the value 99, it should start again at 0. The counter should stop/start when any pushbutton *KEY* is pressed.

To achieve a delay of approximately 0.25 seconds, use a delay-loop in your assembly language code. A suitable example of such a loop is shown below.

```
DO_DELAY:    LDR     R7, =200000000       // delay counter
SUB_LOOP:    SUBS    R7, R7, #1
             BNE     SUB_LOOP
```

To avoid "missing" any button presses while the processor is executing the delay loop, you should use the *Edgecapture* register in the *KEY* port, shown in Figure 3. When a pushbutton is pressed, the corresponding bit in the *Edgecapture* register is set to 1, and it remains set until reset to 0 by writing into the register.

Perform the following:

1. Create a new folder to hold your Monitor Program project for this part. Create a file called *part2.s* and type your assembly language code into this file.
2. Make a new Monitor Program project in the folder where you stored the *part2.s* file. Use the DE0-Nano-SoC Computer for this project, and select the ARM A9 as the target processor architecture.
3. Compile, download, and test your program.

Part III

In Part II you used a delay loop to cause the ARM processor to wait for approximately 0.25 seconds. The processor loaded a large value into a register before the loop, and then decremented that value until it reached 0. In this part you are to modify your code so that a hardware timer is used to measure an exact delay of 0.25 seconds. You should use polled I/O to cause the ARM processor to wait for the timer.

The DE0-Nano-SoC Computer includes a number of hardware timers. For this exercise use the timer called the A9 *Private Timer*. As shown in Figure 5 this timer has four registers, starting at the base address 0xFFFEC600. To use the timer you need to write a suitable value into the *Load* register. Then, you need to set the enable bit E in the *Control* register to 1, to start the timer. The timer starts counting from the initial value in the *Load* register and counts down to 0 at a frequency of 200 MHz. The counter will automatically reload the value in the *Load* register and continue counting if the A bit in the *Control* register is set to 1. When it reaches 0, the timer sets the F bit in the *Interrupt status* register to 1. You should poll this bit in your program to cause the A9 processor to wait for the timer. To reset the F bit to 0 you have to write a 1 into this bit-position.

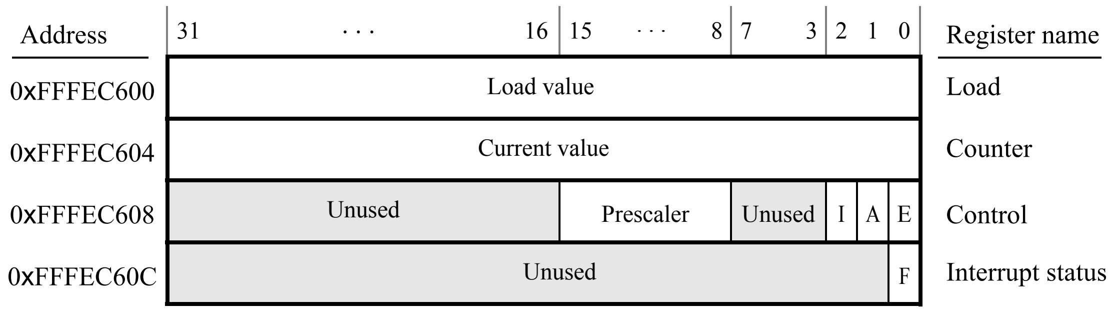


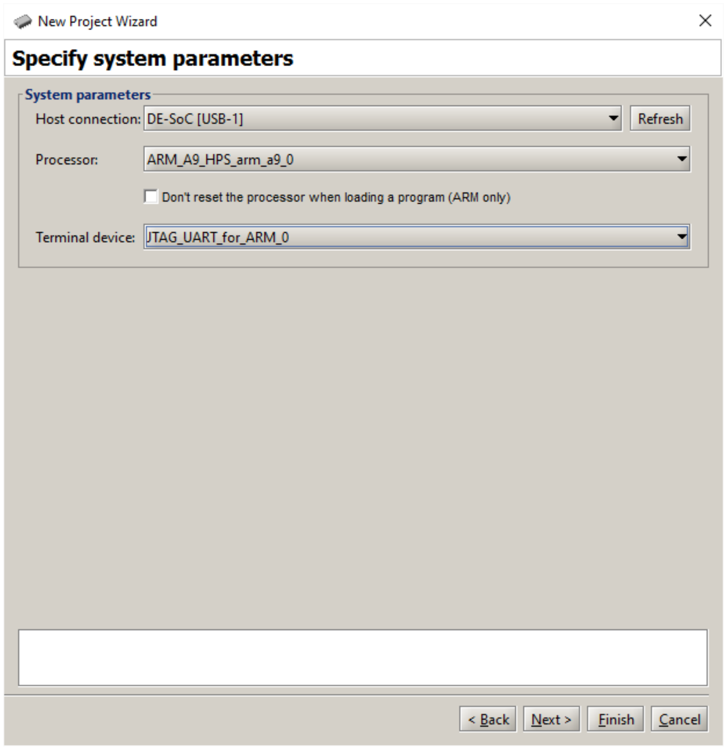
Figure 5: The A9 Private Timer registers.

Make a new folder to hold your Monitor Program project for this part. Create a file called *part3.s* and type your assembly language code into this file. Make a new Monitor Program project for this part of the exercise, and then compile, download, and test your program.

Part IV

In this part you are to write an assembly language program that implements a real-time clock. Display the time on the Monitor Program's Terminal window in the format MM:SS, where *MM* are minutes and *SS* are seconds. Measure time intervals of 1 second in your program by using polled I/O with the A9 Private Timer. You should be able to stop/run the clock by pressing any pushbutton *KEY*. When the clock reaches 59:99, it should wrap around to 00:00.

Make a new Monitor Program project for this part of the exercise. In the screen shown in Figure 6, make sure to select JTAG_UART_for_ARM_0 as the *Terminal device*. Otherwise, no character output will appear on the Terminal window. Refer to Exercise 2, Part IV, for information on using the JTAG UART to communicate with the Monitor Program's Terminal window.


New Project Wizard

Specify system parameters

System parameters

Host connection: DE-SoC [USB-1]

Processor: ARM_A9_HPS_arm_a9_0

☐ Don't reset the processor when loading a program (ARM only)

Terminal device: JTAG_UART_for_ARM_0

Figure 6: Specifying the *Terminal device*.

You may wish to make use of the following text strings. The first one clears the Terminal window, and the second one returns the "cursor" to the upper-left corner of the window:

```
CLR_SCRN:    .asciz  "\033[2J"
HOME:        .asciz  "\033[H"
```