Laboratory Exercise 9

Interrupt-Driven Input/Output

Revision of November 18, 2025

In this lab, you will learn to communicate with I/O devices by using interrupts.

Note: This lab will be marked entirely **in-person during your scheduled practical session via a demo to your TA by the end of your assigned practical session. It is STRONGLY recommended that you make good progress on the lab prior to coming to your in-person PRA session. You are required to submit your lab files but there is no tester or automarker for this lab.**

1 Part I

In Lab 8, you created a counter that uses a hardware timer to increment approximately every 0.25 seconds. You used polled I/O to synchronize with the hardware timer and also used polled I/O to check for key presses to halt or continue the counter as needed. In this lab, you are to create the same application but relying entirely on interrupts instead of polled I/O. Interrupts have to be generated by both the timer and KEY port.

You are provided with a template code on Quercus named `lab9_template.s`. The skeleton of this code is also shown below. At the beginning of the main program, the stack pointer is initialized. Then, the main program must enable two types of interrupts: the Timer and the Key pushbuttons. To achieve this, the main program calls the subroutines *CONFIG_TIMER* and *CONFIG_KEYS*, which you need to write. In *CONFIG_TIMER*, set up the Timer to generate an interrupt every 0.25 seconds. Additionally, ensure that the edge bits are cleared when configuring *CONFIG_KEYS*. Review your lecture notes to understand how to enable interrupts for these two devices.

After configuring the hardware, ensure that interrupts are enabled in the NIOS V processor and set up the address handling location to the *interrupt_handler* subroutine, as covered in the lecture.

The main program executes an endless loop, writing values of the global variable COUNT to the red lights LEDR.

In the addition to the aforementioned changes, you are to modify the code given in `lab9_template.s` to make it work as follows:

- You must write 2 separate interrupt service routines: one for the timer (`Timer_ISR`)

and one for the KEYs (`KEYs_ISR`). In `KEYs_ISR`, you are to switch the value of the RUN global variable every time `KEYs_ISR` is called (RUN is switched between 0 and 1). In `Timer_ISR`, you are to increment the variable COUNT by the value of the RUN global variable, which should either be 1 or 0. Your counter should reset to 0 after the count reaches 255 that is, we display 255, we wait for 0.25 seconds and then we reset. Make sure to clear the interrupt and edge-capture bits accordingly in either interrupt service routines.

- You will also need to modify the *interrupt_handler* subroutine so that it calls the appropriate interrupt service routine, depending on whether an interrupt is caused by the timer or the KEYs.

2 Part II

Modify your program from Part I (in a file called *part2.s*) so that you can vary the speed at which the counter displayed on the red lights is incremented. All of your changes for this part should be made in the interrupt service routine for the KEYs. The main program and the rest of your code should not be changed.

Implement the following behavior: When KEY_0 is pressed, the value of the *RUN* variable should be toggled, as in Part I. Hence, pressing KEY_0 stops/runs the incrementing of the *COUNT* variable. When KEY_1 is pressed, the rate at which *COUNT* is incremented should be doubled, and when KEY_2 is pressed the rate should be halved. You should implement this feature by stopping the timer within the KEYs interrupt service routine, modifying the load value used in the timer, and then restarting the timer. Additionally, pressing the button to halve the speed might reduce it to zero. If you then try to double the speed to increase the timer, it will not work. Therefore, set both a minimum and maximum speed limit.

3 Submission and grading

You must submit both parts of lab using the submit command. You will be graded based on a demo of your working code to your TA **during your practical session.** TA will test the functionality and ask you questions about your code. It will take the TAs time to mark everyone's demos to please come to lab with much of the lab work completed already. **Do not leave the lab before having your code marked by the TA.**

```
.global _start
_start:
    .equ LEDs, 0xFF200000
    .equ TIMER, 0xFF202000
    .equ PUSH_BUTTON, 0xFF200050

    #Set up the stack pointer
    li sp, 0x20000

    jal CONFIG_TIMER #configure the Timer
    jal CONFIG_KEYS #configure the KEYs port

    /* Enable Interrupts in NIOS V processor, and set up the address
     handling location to be the interrupt_handler subroutine */
    # Your code goes below here:
      # Your code should:
      # Turn off interrupts in case an interrupt is called before correct set up
      # Activate interrupts from IRQ18 (Pushbuttons) and IRQ16 (Timer)
      # set IRQ on
      # set the mtvec register to be the interrupt_handler location
      # Turn  the interrupts back on

    la s0, LEDs
    la s1, COUNT
    LOOP:
        lw s2, 0(s1)  # get current count
        sw s2, 0(s0)  # store count in LEDs
    j LOOP

interrupt_handler:
    # code not shown
mret

CONFIG_TIMER:
    # code not shown
jr ra

CONFIG_KEYS:
    # code not shown
jr ra

.data

.global COUNT
  COUNT: .word 0x0     # used by timer
.global RUN           # used by pushbutton KEYs
  RUN:    .word 0x1    # initial value to increment COUNT
.end
```