University of Puerto Rico at Mayagüez
Department of Electrical and Computer Engineering
Microprocessor Interfacing Laboratory

Experiment 2: Interrupts & Switch Debouncing

Objectives

- Learn how to read a key via interrupts
- Become familiar with the process of debouncing a switch

Duration

- 2 Hours in laboratory and extra time for homework.

Materials

- Your Microcontroller IDE application
- Your Microcontroller development board
- List of parts attached to this Laboratory.

Introduction

The purpose of this session is to guide you through the basics in interrupts and contact bounce. We continue to explore the different tools that you might find in any particular MCU. You will learn how to handle and setup interrupts, also how to control the bounce problem either with hardware or software solutions.

About Interrupts

In computing, an interrupt is an asynchronous signal indicating the need for attention or a synchronous event in software indicating the need for a change in execution. In your case you will be working with hardware interrupts that cause the processor to save its state of execution and begin execution of an interrupt handler.

To configure interrupts in a typical MCU it is important to follow these steps:

- **Stack Setup:** If you are using assembly language, you'll need to allocate stack space in memory and initialize the stack pointer (this is also necessary when you use call instructions).You do not have to do this if your MCU has a hardware stack or if you are programming in C language (the compiler makes this for you).
- **Write the interrupt service routine (ISR):** This is the code to be executed by the CPU to serve the interrupt. Avoid loops or calling subroutines from inside the ISR. Just write a simple and short code. Is important when you use ASM to write the ISR keeping register transparency (push all registers used in the ISR onto the stack at the beginning and pop them at the end of the ISR). If you are using C language the compiler does this for you.
- **Set-up the interrupt table:** Once your ISR is written, enter the ISR location in the interrupt table. This is how the CPU will know where the ISR is located. All MCUs have a table with entries for each interrupt source. In assembly, all you need to do is to take the label from the ISR and write an absolute jump instruction to this label in the corresponding table entry.
- **Interrupt enable:** Make sure that you have enabled the CPU global interrupt flag and the particular enable flag of the device. First set the flags corresponding to each device and then enable the CPU global interrupt flag. Many devices require to re-enable its interrupt flag at the end of the ISR to ensure it can be triggered again.

About Switch Bouncing

Switch contacts are usually made of springy metals that are forced into contact by an actuator. When the contacts strike together, their momentum and elasticity act together to cause bounce. The result is a rapidly pulsed electrical current instead of a clean transition from zero to a logical 1. This problem can be solved via hardware or via software.

For solving the bounce problem via hardware, an RC circuit is typically used. For solving problem via software you need to write an algorithm to ignore the switch input during the bouncing period.

Procedure

Part I: Read a key using Interrupts

1. Use the set-up developed in Experiment 1 that connects an LCD and a switch to your MCU. Make sure the I/O port where the switch is connected has interrupt capabilities. If necessary, move the switch to an input that has.
2. Modify the main program to setup the stack, and initialize the stack pointer (if necessary)
3. Write the code for your ISR. The ISR just needs to increment the value of a (global) variable each time it is executed. Remember to make your ISR register transparent. The flowchart in Figure 1 can guide in writing the ISR.

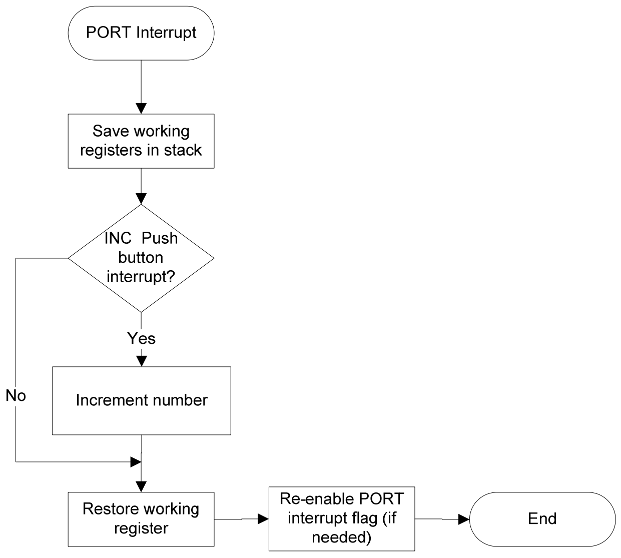
PORT Interrupt
Save working registers in stack
INC Push button interrupt?
Yes
No
Increment number
Restore working register
Re-enable PORT interrupt flag (if needed)
End

Figure 1: ISR flowchart.

4. Identify which entry in the MCU jump table corresponds to the port where the switch was connected. Use the label of your ISR for writing the jump instruction.
5. Insert instruction in your main to enable the interrupts. Recall enabling first the PORT flag and then the CPU global IF.
6. Modify further the main to make it a program that every time the number is incremented, its value gets displayed on the LCD. Use the flowchart in Figure 2 as a guide. Note: Use the LCD subroutines you created in experiment 1.

Exercise 1:

Modify your program to have a second key to decrement the count using interrupts.

Turn-in:

- Code for this and produce a demo to the TA.

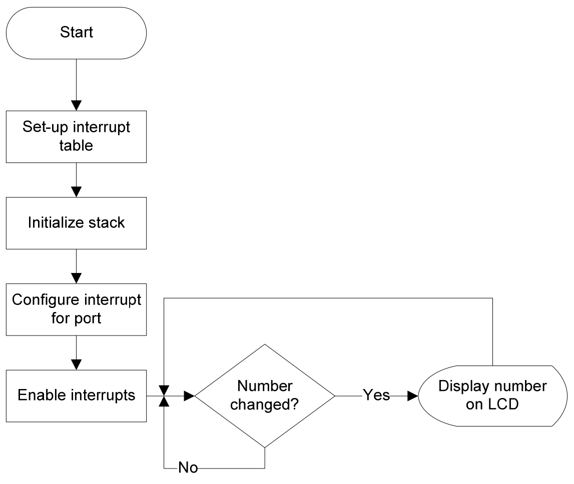

Figure 2: Flowchart for the main loop in the key-LCD incrementing program.

<u>Note</u>: You may notice that when you press the increment button the number might increase by more than one, even when the switch is depressed just once. This is a manifestation of the switch bouncing phenomena. The contact bounce produces pulses large enough to be detected by the MCU input port causing multiple interrupts.

Part II: Debouncing a Key via Hardware

1. Assemble the circuit in Figure 3, with R1= 2k2Ω, R2= 10kΩ, C=3.3uf. These values are chosen assuming a push button bounce of about 20ms. The inverting buffer has a Schmitt triggered input. If your MCU has Schmitt triggered input ports, the inverter is not needed. Otherwise provide one externally.
2. Replace the switch in the circuit used in *part I,* for this circuit.

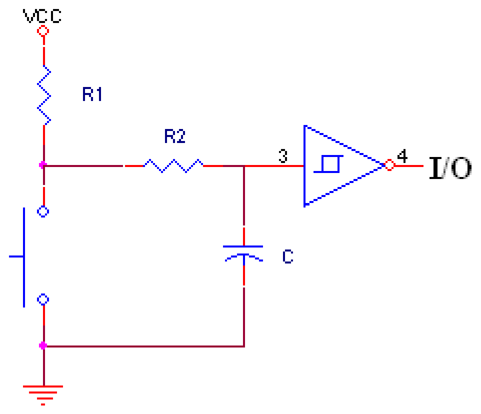

Figure 3: Schematic for hardware debouncing circuit.

3. Run the program created in part I, and see if the bounce effect disappears. Use a larger R1 if it does not.

Exercise 2:

Modify your circuit to have the second key debounced by hardware.

Turn-in:

- Produce a demo to the TA.

Part III: Debouncing a Key via Software

For software debounce activate a flag in the ISR to indicate that the switch has been closed. This flag will affect a counter in the main loop that will count until more than 20ms have passed (this depend on the push button bounce) making sure that the switch signal has stabilized and that the ISR will not accept any other interrupt. Then the flag needs to be reset so the ISR can continue accepting interrupts.

1. Restore the key push button to its original state without a hardware debouncing mechanism.
2. Modify the code of the ISR in *Part I* for setting a flag when an interrupt is generated. Follow the flow chart in Figure 4 as a guide.

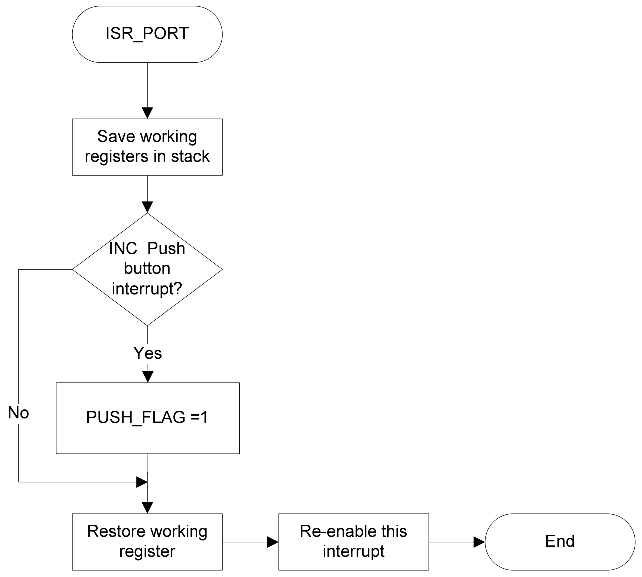

Figure 4: Software debouncing ISR flowchart.

3. Modify the code in *Part I* for every time the flag is set wait 30ms, reset the flag and if after the 30ms the flag is one, increment the value and display it in the LCD. See Figure 5 for a flowchart. Note this subroutine can be optimized using a timer interrupt to count the 30ms, you will learn this in the next experiment.
4. Test your program, and see that bounce effect is gone.

Exercise 3:

Modify your program to have both keys debounced by software.

Turn-in:

- Code for this and produce a demo to the TA.

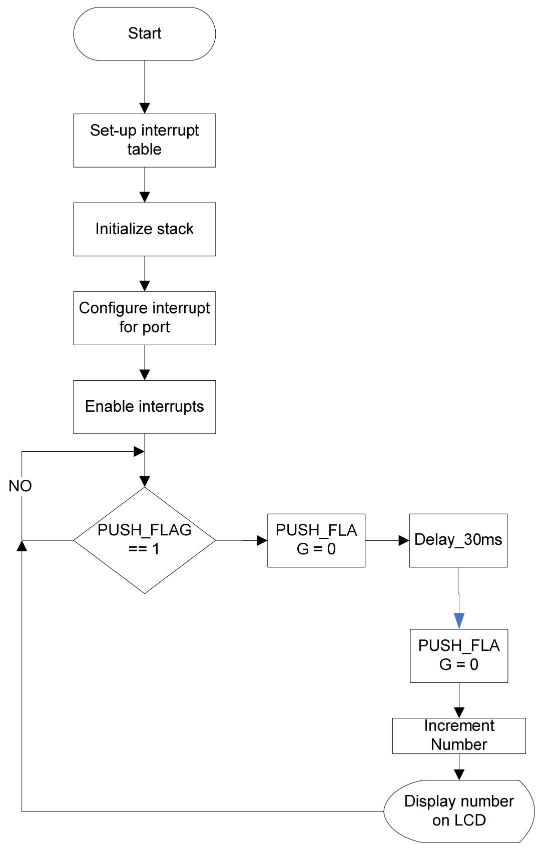

Figure 5: Software debouncing main flowchart.

Homework

Modify the scrolling message program in Homework 1 to work with a scrolling wheel instead of switches. To detect the movement direction, build an encoder wheel. An optical encoder can convert the angular movement and direction of the wheel into a set of digital pulses. For the wheel encoding pattern use the diagram shown in Figure 6. Affix it to one side of the wheel. Mount the wheel in a base as shown in Figure 7. Connect two opto- switches aligned vertically with the two sets of marks in the wheel. Note that the whole assembly can be built with cardboard. Investigate how to read a quadrate encoder using lookup table techniques. Figure 8 shows an example electrical connection of an opto-switch.

Turn-in:

- Software plan and explanation (pseudo code or flowchart)
- Connection schematic with components calculations and wheel assembly.
- Listing of code
- Have your circuit assembled and produce a demo to the TA or professor.

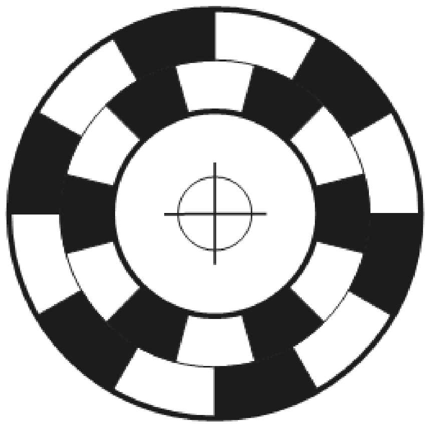
Figure 6: Layout of quadrature encoding patterns to be attached to the wheel.

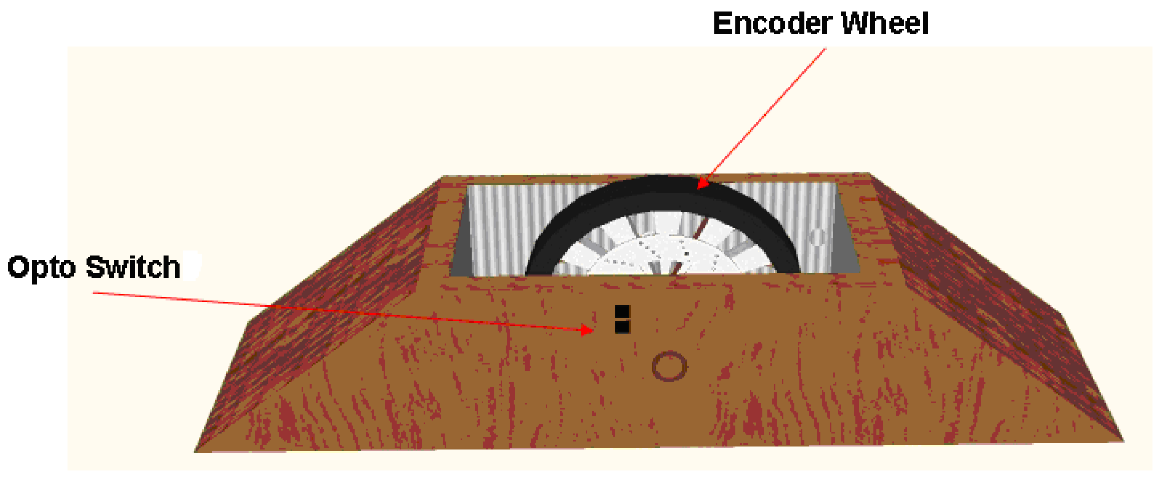

Figure 7: Suggested mounting base for encoding wheel.

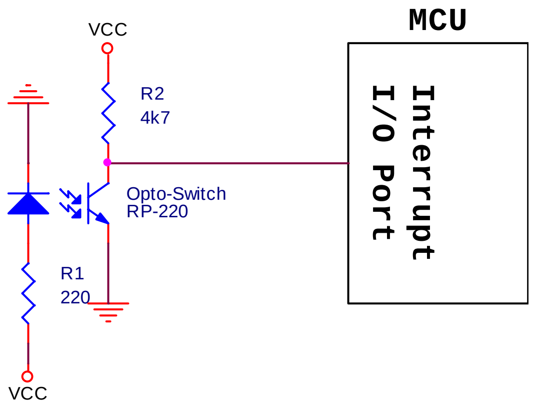

Figure 8: Opto-Switch connection schematic.