Embedded System Laboratory Course

Dr. Fangning Hu

Protected: Lab 2 – External Interrupt (Assembler)

Introduction:

Interrupts are events that require immediate response by the microcontroller. When an interrupt event occurs, the microcontroller pauses its current task and responses to the interrupt by executing an Interrupt Service Routine (ISR). After finishing the ISR, the microcontroller returns to the task it had paused and continue its normal operations.

The Atmega328 provides many internal and external interrupt sources. Internal interrupts include Timer Overflow Interrupt, A/D Converter Interrupt, etc. which we will learn later. In this lab we are going to use the external interrupt. External interrupts are triggered by external pins **INT0** or **INT1** (You can check the Pin Layout to find out which pins on the Arduino correspond to INT0 and INT1). Any voltage change on one of these pins will trigger the corresponding external interrupt.

A corresponding interrupt bit need to be set to one. Besides, the Global Interrupt Enabled bit, i.e., the I-bit in the Status Register SREG has to set to one by the assembly instruction **SEI**

in order to enable the interrupt in general.

If all the enable bits set to one and an interrupt is triggered, the address of the next instruction will be saved in the stack, the I-bit will be cleared. The address of the corresponding Interrupt Vector will be loaded into the program counter (PC). The microcontroller starts execution from that address until it reaches a RETI instruction. Upon the execution of the RETI instruction the address that was stored on the stack is reloaded in the PC and the I-bit is re-enabled. The
microcontroller then start executing instructions from that point. That is the point that it left off when the interrupt was triggered.

Each interrupt has its own Interrupt Vector and their addresses are usually reserved in the beginning of the Flash Memory (datasheet Interrupts). At those address, an rjmp instruction

RJMP *your_ISR_label*

should be written there to tell the program jump to your Interrupt Service Routine. You will write your Interrupt Service Routine after this label

your_ISR_label : ;your Interrupt Service Routine start here

The address 0×0000 is usually reserved for the Reset Interrupt Vector. A typical way to set up the Reset Interrupt Vector is:

```
.org 0x0000
          RJMP begin; jump to begin
.org0x0034          ;Initialize the stack pointer
 begin:
         LDI    R16,low(RAMEND)
           OUT    SPL,R16
         LDI    R16,high(RAMEND)
           OUT   SPH, R16
```

IMPORTANT NOTES:

-The Stack Pointer MUST be initialized in order to use interrupt.

- Each interrupt routine MUST ends with the **RETI** instruction.

- I-bits has to set to 1 by **SEI** in order to enable any interrupt.

PreLab Tasks:

1.Read the ATmega328 datasheet (Chapter External Interrupts) and find out the meaning of each pin in the necessary registers. You will need to set the corresponding bit in register **EMISK** to enable the external interrupt and set the corresponding bits in **EICRA** to control which types of voltage change will trigger the interrupt. Find the I/O addresses for registers EMISK and EICRA. Note that the command OUT can only access the lower 64 I/O registers. If you need to access I/O registers above these 64 I/O registers, you need to use the assembly instruction:

STS k Rd

to store the value stored in the working register Rd into the I/O address k.

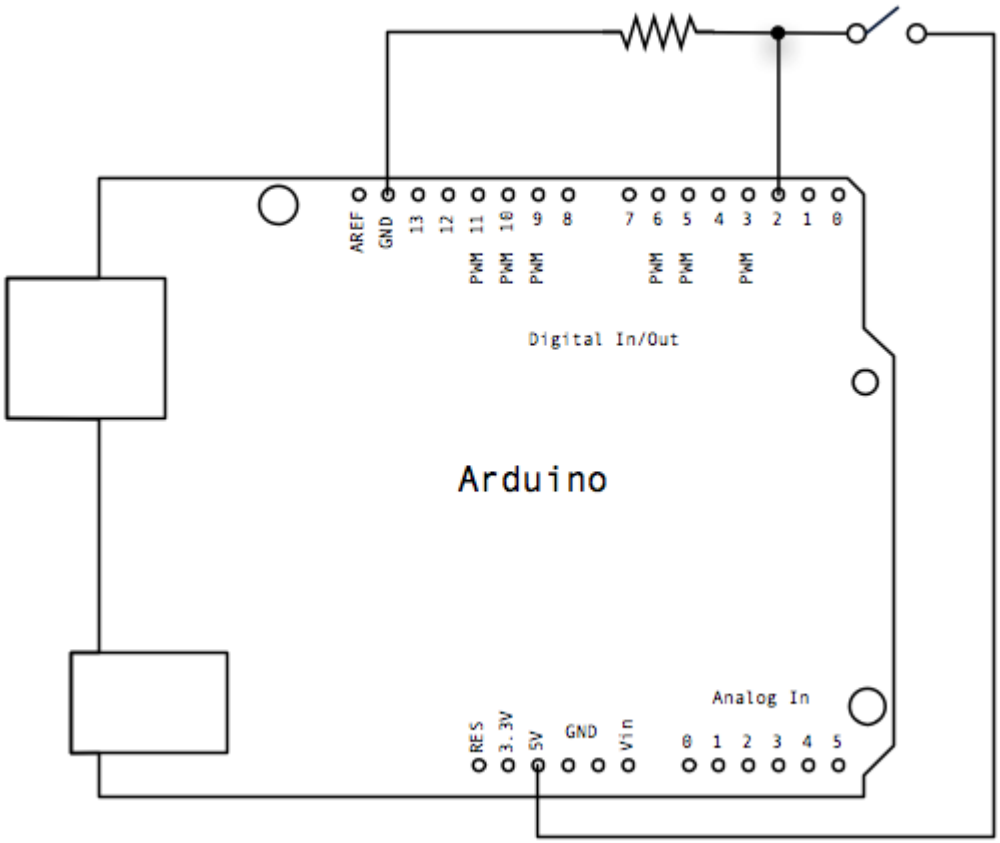
2.Read the ATmega328 datasheet (Chapter Interrupts) and find out the program address for the External Interrupt Vector in the Interrupt Vector Table. Be careful to the different devices in the datasheet, they have different Tables!!

Lab Assignments:

1. Check the Ardunino Schematic to find out which pin you should use to connect to the button. Design your circuit such that when pressing the button, an external interrupt will be triggered. A typical way to do that is using a 10K Ohm pull-up resistor as shown in the circuit diagram where the input voltage to the pin keeps high before pressing and becomes low after pressing the button.

2. Design what you want the LED does when pressing the button. In my case, I make it quickly blink twice when pressing the button. Otherwise it blink slowly in the normal situation. Write your Interrupt Service Routine codes to finish your design.

A Pressing Button with a Pull-Up Resistor



Lab Report: The requirements are the same as the previous lab.