Embedded Hardware Design

Monsoon 2017

Experiment 2

**Date : ** 17-08-2017
Topics: Interrupts, Switch Debouncing and Serial Display

Introduction to Interrupts on AVR Microcontrollers-

Interrupts are one of the most fundamental and powerful techniques present in modern embedded processors in which the processor is essentially diverted from the execution of the current task so that it may deal with some event that has occured. This event may be *hardware* or *software* generated. Once the interrupt request is served, the microcontroller resumes its original job. This way, instead of the processor continuously checking I/O devices whether they require service (called *polling* or *busy-wait* technique), the I/O device itself will notify (*interrupt request*) the processor when the service is required thereby freeing the processor to do other tasks in the meantime.

Atmega168/328 has only two external interrupt pins- INT0 on PD2 (digital pin 2 of Arduino) an INT1 on PD3 (digital pin 3 of Arduino). The following registers are used to set up interrupts-

1. **EIMSK (External Interrupt Mask Register):** This register controls whether the INT0 and/or INT1 interrupts are enabled.

Bit	7	6	5	4	3	2	1	0
							INT1	INT0
Access							R/W	R/W
Reset							0	0

   - EIMSK \|= 0x01 enables INT0
   - EIMSK \|= 0x02 enables INT1

2. **EICRA (External Interrupt Control Register A):** This register determines under which conditions (level triggering, falling/rising edge) INT0 or INT1 should be triggered.

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

The triggering condition can be set as follows-

ISC01	ISC00	Description
0	0	Low level generates interrupt
0	1	Any logical change
1	0	Falling Edge
1	1	Rising Edge

3. **SREG:** The global interrupt bit is present in this status register which can be set by executing the *sei()* function.

Switch debouncing-

In the previous lab, we had implemeted a circuit which changed the state of an LED based on the state of the push button, i.e, if the button is pressed, the LED glows and if it is released, the LED turns off. In this, we didn't experience switch bouncing effects which are associated with mechanical push buttons.
When you press a push button, the two metal parts come together. For the user, this seems like an instant action. That is, however, not true. When you push the button once, it initially makes contact with the other metal part, but just for a split second. Then it makes contact a little longer, and then again a little longer. In the end, the switch is fully closed. This is called bouncing of a switch. All of this happens in the order of microseconds but our microcontroller is capable of reading states that fast. In short, one press of a switch may result in the microcontroller reading multiple presses. There are two methods to solve this- software debouncing (using an appropriate delay between reads) and hardware debouncing.

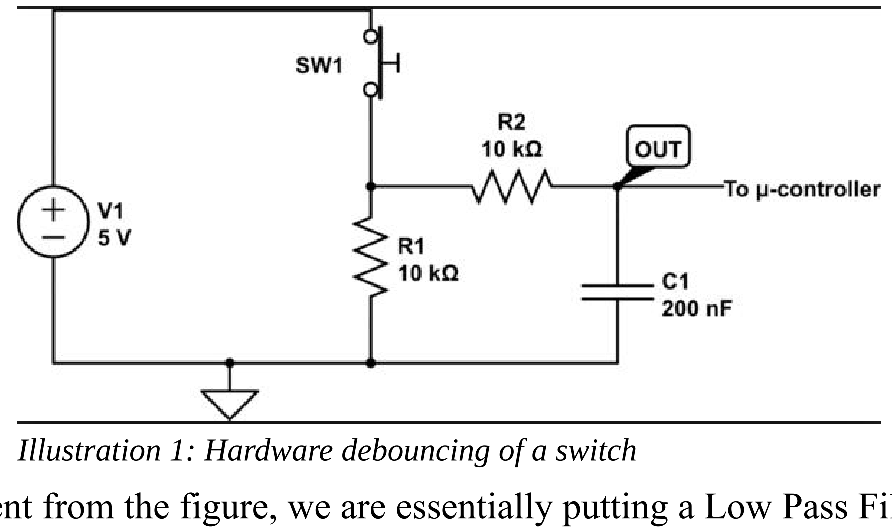

Illustration 1: Hardware debouncing of a switch

As it is evident from the figure, we are essentially putting a Low Pass Filter infront of the switch signal.

Experiment 2A: Toggle an LED

1. Connect a push button to INT0 pin of the Arduino with hardware debouncing circuit (can be for pull-up or pull-down).
2. Connect an LED to a pin in Port B or D.
3. Set the appropriate configurations for the switch and LED.
4. Configure interrupts on INT0 for rising/falling edge.
5. When the push button is pressed once, the LED should light up and remain that way.
6. The second push of the button should turn the LED off.
7. Repeat 5 and 6.

Serial Monitor-

Serial is used for 2 way communication between the Arduino board and a computer or other microcontroller. All Arduino boards have at least one serial port (also known as a UART or USART). It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pin 0 (PD0) and 1 (PD1) for digital input or output.
The Arduino IDE has a built in serial monitor on which you can display information from arduino or send inputs to Arduino. It is used extensively for debugging programs. Serial Monitor can be configured using *Serial.begin(9600)* command in the setup, where 9600 is the baud rate. *Serial.print()* is used to print a value from the Arduino to the monitor.

Experiment 2B: Implement a 4-bit counter

In this experiment, we'll use the switch connected to INT0 as clock to increment the counter and display the value on the Serial Monitor.

1. Connect a push button to INT0 pin of the Arduino with hardware debouncing circuit (can be for pull-up or pull-down).
2. Set the appropriate configurations for the switch to act as input.
3. Configure interrupts on INT0 for rising/falling edge.
4. If the push button is pressed, increment the value of the counter.
5. Display the current value on the Serial Monitor.
6. Repeat 4 and 5.

Pseudo Code-

```
int main(void)
{
    //configure I/O ports
    //configure registers for edge triggering on INT0
    while (1)
    {
    //do some task repeatedly
    }
}

ISR (INT0_vect) //Interrupt Service Routine for INT0
    {
    //code for what the interrupt should do
    }
```