Revised by Dr. Haroon Ahmed Khan, March 5th, 2020

Lab # 07 External Interrupts & Switch De-bouncing

Objectives

- To learn the concepts related to interrupts in AVR microcontroller.
- To configure and use the external interrupt or user input tasks

Software Requirement

- Atmel Studio or AVR Studio
- Proteus
- AVRDUDESS

Pre-Lab Reading & Literature

What is an Interrupt?

An interrupt refers to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from *(before the interrupt was received)*. The interrupts can be either internal interrupts or external interrupts.

Why need interrupts:

An application built around microcontrollers generally has the following structure. It takes input from devices like keypad, ADC etc., processes the input using certain algorithm and generates an output which is either displayed using devices like seven segment, LCD or used further to operate other devices like motors etc. In such designs, controllers interact with the inbuilt devices like timers and other interfaced peripherals like sensors, serial port etc. The programmer needs to monitor their status regularly like whether the sensor is giving output, whether a signal has been received or transmitted, whether timer has finished counting, or if an interfaced device needs service from the controller, and so on. This state of continuous monitoring is known as polling.

In polling, the microcontroller keeps checking the status of other devices and while doing so it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts. In interrupt method, the controller responds to only when an interruption occurs. Thus, in interrupt method, controller is not required to regularly monitor the status *(flags, signals etc.)* of interfaced and inbuilt devices.

To understand the difference better, consider the following. The polling method is very much similar to a salesperson. The salesman goes door-to-door requesting to buy its product or service. Like controller keeps monitoring the flags or signals one by one for all devices and caters to whichever needs its service. Interrupt, on the other hand, is very similar to a shopkeeper. Whosoever needs a service or product goes to him and apprises him of his/her needs. In our case, when the flags or signals are received, they notify the controller that they need its service.

External Interrupts:

The External Interrupts are triggered by the INT pins or any of the PCINT pins. The Pin Change Interrupt Request 2 (PCI2) will trigger if any enabled PCINT[23:16] pin toggles. The Pin Change Interrupt Request 1 (PCI1) will trigger if any enabled PCINT[14:8] pin toggles. The Pin Change Interrupt Request 0 (PCI0) will trigger if any enabled PCINT[7:0] pin toggles. The PCMSK2, PCMSK1 and PCMSK0 Registers control which pins contribute to the pin change interrupts. Pin change interrupts on PCINT are detected asynchronously.

The External Interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the External Interrupt Control Register A (EICRA).

In-Lab

How to use interrupts:

- First, we have to configure and enable the global interrupts. The most significant bit of status register called ‘**I**’ bit is used for this purpose. If ‘**I**’ is set 1 using register SREG this means interrupt is enabled otherwise disabled.
- We write functions called Interrupt Service Routine (ISR) to handle interrupts. These functions are defined outside the main function because the event that causes interrupt is not known by the programmer; hence the function can’t be called inside the main function.
- Enable the external interrupts locally in External Interrupt Mask Register (EIMSK). Then configure the interrupts for falling edge, rising edge, low level or any logical change by using EICRA register.

Advantages of Interrupt method:

- Priority can be assigned.
- Controller does not waste time checking if a device needs service or not.

Steps involved executing an interrupt:

Upon activation of an interrupt the microcontroller goes through the following steps as shown in figure 7.1:

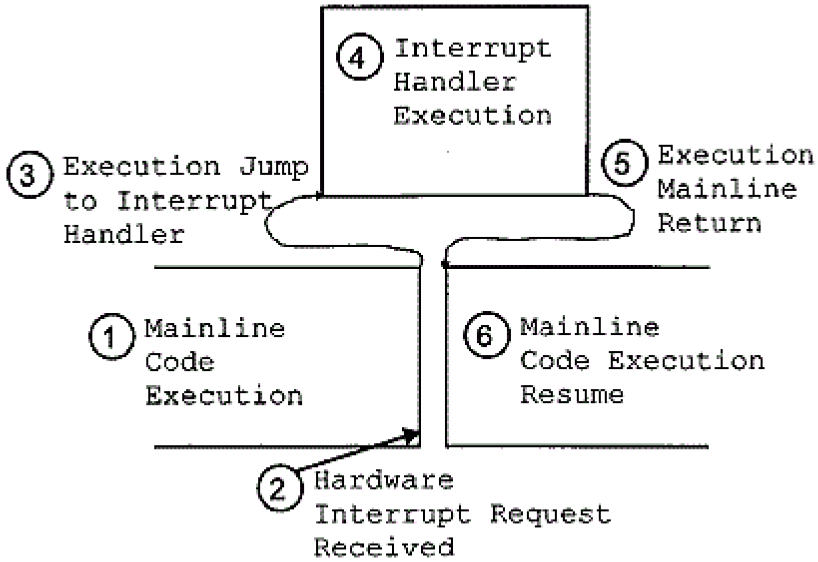

Figure 7.1: Interrupt Service Routine

1. It finishes the instruction that it is executing and saves the address of next instruction on the stack.

2. The program branches to the address of corresponding interrupt in the interrupt vector table. The code starting here is called interrupt handler.

3. Check which source generated the interrupt using interrupt flags.

4. Executes the corresponding interrupt service subroutine.

5. Upon executing the last instruction in ISR the microcontroller returns to the place where it was interrupted using RETI instruction. First it gets the program counter address by popping it from the stack. Then it starts to execute from that address.

Available interrupts in ATMEGA328P micro controller:

There are 26 interrupts in ATmega328P (Table 7.1) with 21 internal and 5 external interrupts. The external interrupts are **RESET**, **INT0** *(pin16)* and **INT1** *(pin17)*. All 21 interrupts are listed in table below in descending order of priority. In this lab we will focus on external interrupts.

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Table 7.1: Interrupt Vector table available on ATmega328P microcontroller

Register Description:

The registers involved in configuring the external interrupts are shown in below.

External Interrupt Control Register A(EICRA):

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

External Interrupt Mask Register (EIMSK):

In this register if INTn bit is set and the I-bit in the Status Register (SREG) is set, the external pin interrupt is enabled.

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

External Interrupt Flag Register (EIFR):

When an edge or logic change on the INTn pin triggers an interrupt request, INTFn will be set. If the I-bit in SREG and the INTn bit in EIMSK are set, the MCU will jump to the corresponding Interrupt Vector. The flag is cleared when the interrupt routine is executed. Alternatively, the flag can be cleared bywriting '1' to it. This flag is always cleared when INTn is configured as a level interrupt.

Bit	7	6	5	4	3	2	1	0
							INTF1	INTF0
Access							R/W	R/W
Reset							0	0

Registers for Pin change interrupt:

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

In addition to above registers you will need to set Global Interrupt Enable (GIE) bit. **GIE** is bit7 of status register **SREG**.

For interrupt handling we need to include following header file into our project:

#include <avr/interrupt.h>

The following format is used to declare interrupt service routine:

```
ISR(ISR_Vect){
    //Interrupt handling code goes here ...
}
```

Pre-Lab Task 1:

Generate binary counting with the help of LED's interfaced by MCU and controlled by 2 Switches. One for enabling the circuit and the other is to reset it. Switch pressing is an external event, that's why we use external interrupts.

Task: Write the C code for Interrupts and simulate in Proteus

Registers used in this task:

- EICRA
- EIMSK
- SREG (Status register)

Code:

```
#include<avr/interrupt.h>

#include<avr/io.h>

#define F_CPU 16000000UL


unsigned char counter=0;

ISR(INT0_vect){

     // WRITE YOUR CODE HERE

}

ISR(INT1_vect){

     // WRITE YOUR CODE HERE

}

int main(){

     DDRB= 0XFF;

     counter=0;

     //Enable interrupts globally

     //Enable INT0 and INT1 interrupt locally

     //configure EICRA for falling edge INT0 and INT1

     while(1){

     }
}
```

Proteus schematic:

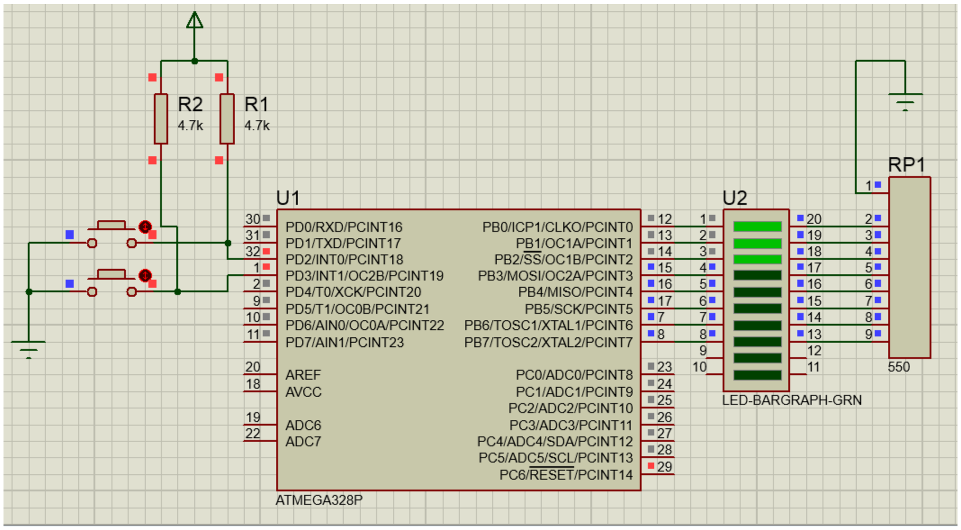

Figure 1.3: Proteus Schematic

In-Lab Task 1:

Implement the pre-lab task on hardware.

Code:

Simulation:

Post-Lab Task 1:

Design a controller for a microwave oven described as follows:

- The controller can initiate the start and stop of (i) the microwave emission, (ii) the turn table rotation, (iii) one light and (iv) ring a bell.
- Imagine your controller is attached to a register that contains the "duration" between 0 and 255.
- Whenever the user hits the "**start button**", the microwave (emitter, turn table and light) should start **if** the "duration" value is more than 0. The microwave should remain in operation for cycles equal to the duration (i.e. if duration is 25, it should remain on for 25 cycles).
- Whenever the user hits the "**pause button**", the microwave should stop (emitter and turn table only, light will remain on), till the time the **pause button** is pressed again
- Whenever the user hits the "**stop button**", the microwave should stop entirely, and the system should reset
- Once the time elapses, the microwave should stop (emitter, turn table and light) and a single bell should be rung.

You are required to finish this task on Proteus. Use LEDs to signify the emitter, turn table, light and bell. Use an 8-bit dip switch array to represent the "duration" input.

Post-Lab Task 2:

What is switch debouncing? Explain software and hardware methods for switch debouncing.

Critical Analysis / Conclusion

(By Student about Learning from the Lab)

Lab Assessment		
Pre-Lab	**/5**	**/25**
Performance	**/5**	
Results	**/5**	
Viva	**/5**	
Critical Analysis	**/5**	
Instructor Signature and Comments		