Interrupts in Atmel AVR using Assembly Language Programming

EE2016 Microprocessors

Sanjeev Subrahmaniyan S B

EE23B102

Abstract

Interrupts are one of the most useful and important functionalities of microcontrollers. Interrupts are signals to the CPU, which can be triggered either due to a hardware state change, like that of the potential of an input pin, or triggered internally at fixed intervals or when specific conditions are met. In this experiment, the concept of interrupts is used with an Atmel ATMega microcontroller. The functionality shown is to trigger the blinking of an LED for 10 seconds when the interrupt is triggered, after which the LED stops blinking. The practical application of such an idea would be in elevator doors, which have to decide when to stay open and close, in which case they wait for a duration after the last person was sensed. The interrupts are implemented by simulation using assembly langugage and demonstrated on a breadboard based circuit. As an extension, the same objective is also attained using the 16 bit timer in the microcontroller.

1 Introduction

Interrupts are exactly what they mean - they are commands to the CPU to stop whatever it is doing, perform another important action, and resume its current function. In microcontrollers, interrupts work in a similar way. When the CPU receives an interrupt instruction, it completes whatever task it is performing at the moment, stores the current state, also called the context on the stack and services the interrupt. This service is typically an interrupt service routine, which is a function that is called when the interrupt is triggered. When the ISR is done executing, the CPU picks off at where it left off and continues operating.

Interrupts are very important due to the immense scope of application. Most systems which use an embedded microcontroller would certainly be using interrupts in one or the other ways and is an indispensible element of microprocessors. In this experiment, interrupts are demonstrated in a simple lab setup with a circuit, helping one gain hands-on understanding of their functioning at an assembly level.

2 Objectives

The objective of this lab session is:

1. Program and wire a microcontroller to blink an LED with a cycle of 1 seconds(toggles every 0.5 seconds) for 10 seconds when a push button switch is pressed and turn off at the end of the time duration.
2. **As an additional task, it is asked to perform the same operation using a 16 bit timer in the AVR microcontroller. This was also performed.**

3 Assembly Language Programs

The assembly program is written with the following steps of execution:

1. Setup the interrupt vector table by writing the jump statement to the ISR corresponding to that interrupt
2. Write the ISR, which are the instructions that must be executed when the interrupt is triggered
3. Program the microcontroller to wait for the interrupt, execute the interrupt when called and return back to the original state when completed. There is a possibility of having another interrupt when the current ISR is being called, which I have chosen to ignore. This could alternately be handled as well.

Please note that the assembly program for this part was written entirely by us and did not use the template provided on Moodle. The program is explained in the comments that accompany it.

3.1 Program

```
;The interrupt pin chosen to be used is INT0 = PD2
.include "m8def.inc"

.def ledR = r17
.def ledtrig = r16
.def outloopR = r18
.def inloopRL = r24
.def inloopRH = r25
```

```
.def mloopR = r20; This is used to control the led blinking

.equ inloopC = 1760;
.equ outloopC = 71; These two together will make the led with a frequency of 1Hz
.equ mloopC = 20; this makes sure the led toggles 20 times, for a total of 10 secs

.cseg
.org 0
rjmp reset; Jumps the interrupt vector table to the main program

.org 0x02
rjmp isr; Calls the ISR when the interrupt is triggered

.org 0x0100
reset:
    clr ledR
    ldi ledtrig, 0x01
    out DDRB, ledtrig; Sets port B0 as output for the led to be triggered
    ldi r31, 0x00
    out PORTB, r31
    ldi r19, HIGH(RAMEND)
    out SPH, r19
    ldi r19, LOW(RAMEND)
    out SPL, r19; Sets the stack pointer to hold the ISR
    sbi PORTD, 2
    ldi r19, 0x00; Sets portD as input only, which can then trigger the interrupt
    out DDRD, r19

    IN R30, MCUCR;Load MCUCR register
    ORI R30, 0x00;
    OUT MCUCR, R30

    ldi r19, 0x40; can enable int0
    out GICR, r19; enables the interrupt pin
    sei; enables the interrupts

forever:
    rjmp forever ; puts the microcontroller in a loop waiting for the interrupt

```

```
isr: ; isr, which accomplishes the led blink as required
    clr ledR
    mloop:
        ldi mloopR, 20

    l1:
        ldi ledtrig, 0x01
        eor ledR, ledtrig
        out PORTB, ledR
        ldi outloopR, outloopC
    l2:
        ldi inloopRH, HIGH(inloopC)
        ldi inloopRL, LOW(inloopC)

    l3:
        sbiw inloopRL, 1
        brne l3

        dec outloopR
        brne l2

        dec mloopR
        brne l1

        ldi r31, 0x00
        out PORTB, r31 ; turns the led off after the ISR execution
        reti : return of control to the previously executing process
```

A snippet from the video we recording showing the image is attached. In this breadboard, the circuit is in the middle with the push button connected to the interrupt pin as required. The led blinking is shown in the picture shown:

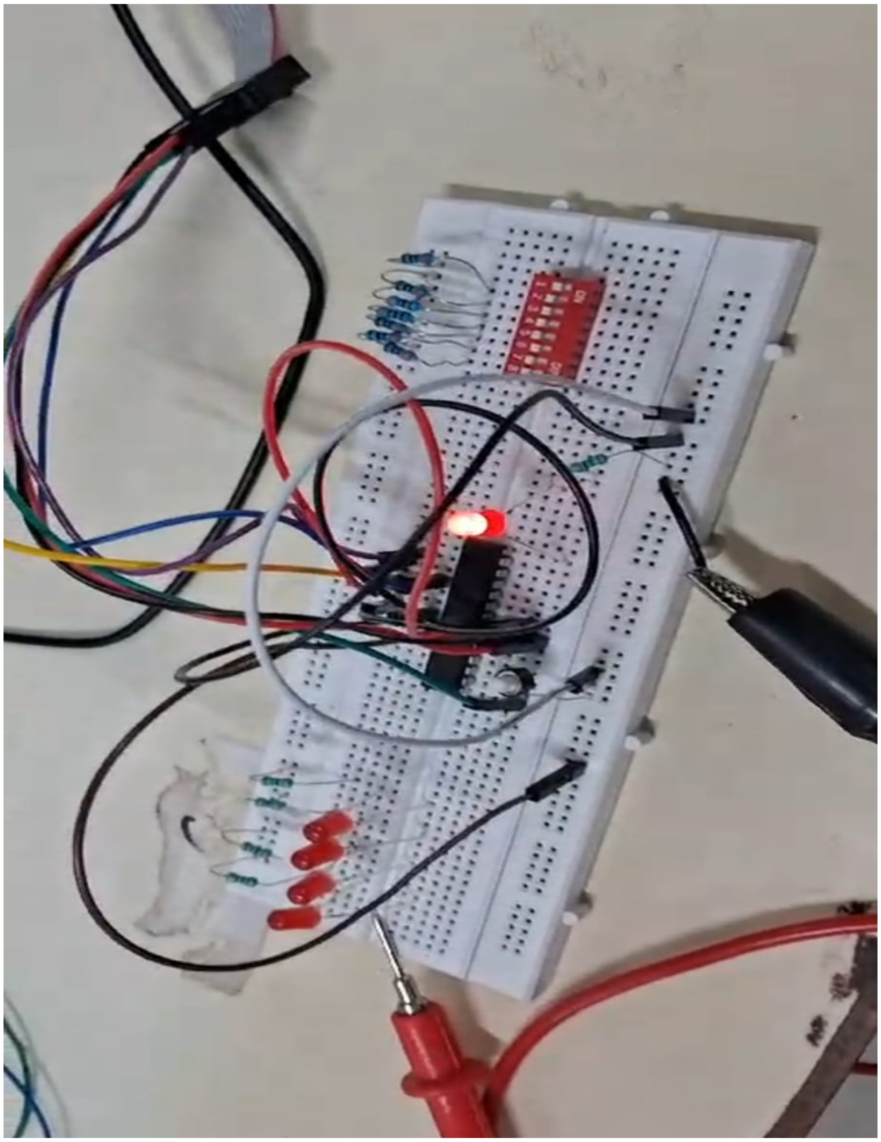

Figure 1: The LED glowing when the interrupt has been triggered

3.2 Selection of control loop variables

As was seen in the previous experiment, the delay required to make the LED blink at a frequency of 1Hz is achieved using two control variables. Additionally, to ensure the LED blinks only for 10 seconds after which it turns off is handled using another extra control variable as seen in the program. All of these in tandem ensure the interrupt behaves as expected.

A note on why mloopR was chosen as 20. Note that the LED toggles its state every 0.5 seconds. This means, for the led to blink for 10 seconds at the given rate requires a total of 20 toggles. This is accomplished using the mloopR control.

3.3 Additional Work - Blinking with timer

The assignment also had an extension task of using a 16 bit timer to create the same delay. The program for it as written by me follows.

```
.include "m8def.inc"
.cseg
.org 0
rjmp 0x0100

.org 0x0100
ldi r16, LOW(RAMEND)
out spl, r16
ldi r16, HIGH(RAMEND)
out sph, r16

; configure pb5 as output
ldi r24, 0x20
sbi ddrb, 5 ; sets pb5 as the led pin
ldi r17, 0
out portb, r17 ; initially sets the led off

; initialize timer1 with a start value
ldi r20, 0x5f ; high byte of initial counter value
out tcnt1h, r20
ldi r20, 0xf8 ; low byte of initial counter value
out tcnt1l, r20

; configure timer1 in normal mode with a prescaler of 256
ldi r16, 0x00
out tccr1a, r16 ; timer1 in normal mode
ldi r16, 0x04
out tccr1b, r16 ; prescaler set to 256

; enable overflow interrupt for timer1
ldi r16, (1<<TOIE1)
out timsk, r16

; enable global interrupts
sei

start:
    rjmp start ; main loop, does nothing as led toggling is handled by isr

```

```
; interrupt service routine for timer1 overflow
.org OC1Aaddr
timer1_ovf_isr:
    eor r17, r24 ; toggle the state of pb5 (led)
    out portb, r17

    ; reset timer1 to initial value 1593 (high = 0x5f, low = 0xf8)
    ldi r20, 0x5f ; high byte of initial counter value
    out tcnt1h, r20
    ldi r20, 0xf8 ; low byte of initial counter value
    out tcnt1l, r20

    reti ; return from interrupt
```

Here, the delay is created by making the sixteen bit counter upcount from a fixed lower value. This ensures that it counts for a time that is as required by us, as mentioned clearly in the comments. Because the division is not exact, the product does not exactly evaluate to 0.5 secs, but a little(a few microseconds) lesser. This is more or less compensated by the other overhead in the program which is not usually counted in timer programming.

4 Procedure

1. Write the assembly program to handle the interrupt as broken down into steps given in the previous section
2. Verify the working of the program on the Microchip studio simulation environment by changing I/O memory registers and writing a corresponding test bench
3. Build the project and retrieve the .hex assembly file for flashing onto the microcontroller
4. Wire up the microcontroller with the LED and the push button connected to the appropriate ports
5. Connect the programmer to the microcontroller and burn the program into the flash memory using AVR BurnOMat software
6. Trigger the interrupt by pushing the button and verify the functionality
7. Report the observations and the steps taken in the experiment

5 Conclusion

This experiment involved understanding the concept of interrupts at a register level and how it is execute by the microprocessor's hardware. It was particularly helpful to gain a deeper understanding of how interrupts can be triggered in different ways, and how they can be handled. It also presented an idea of how interrupts are implemented in RISC based processors, providing vital programming experience. The importance of interrupts in embededded systems was understood. The additional task of using a timer to create the required time delay helps to gain a foundational understanding of how timers are implemented in microcontrollers. The difference between how 8 and 16 bit timers work is exemplified by this experiment.