AVR INTERRUPTS

By Aneesh Kandi
EE20B009

Aim

Using External Interrupts of AVR, blink an LED connected to the microcontroller for 10 times, with a delay of 1s and 50% duty cycle.

Code 1

```
.org 0x0000
rjmp reset

.org 0x0002        ; Interrupt Vector Table location of Interrupt 1
rjmp int1_ISR

.org 0x0100

reset:
        ;Loading stack pointer address
        LDI R16,0x70
        OUT SPL,R16
        LDI R16,0x00
        OUT SPH,R16

        ;Interface port B pin0 to be output so to view LED blinking
        SBI DDRB, 0

        LDI R16,0x00
        OUT DDRD,R16          ; Set PORTD as Input
        SBI PORTD, 3          ; Activate Pull-Up Resistor

        ;Set MCUCR register to enable low level interrupt
        OUT MCUCR,R16

        ;Set GICR register to enable interrupt 1
        LDI R16, 0x80
        OUT GICR, R16
        LDI R16,0x00
        OUT PORTB,R16

        SEI
; --------------Main Program-------------------------
ind_loop:    NOP
             rjmp ind_loop

; -------------------ISR starts here------------------
int1_ISR:    IN R16,SREG
             PUSH R16         ; Store the SREG value in Stack

             LDI R16,0x0A     ; LED blinks 10 times
             MOV R0,R16

      c1:    LDI R16,0x01
             OUT PORTB,R16    ;Switch ON LED

             ;1 second delay
             LDI R16,8
      a1:    LDI R17,125
      a2:    LDI R18,250
      a3:    NOP
             DEC R18
             BRNE a3
             DEC R17
             BRNE a2
             DEC R16
             BRNE a1

             LDI R16,0x00
             OUT PORTB,R16    ; Switch OFF LED

             ; 1 second delay
             LDI R16,8
      b1:    LDI R17,125
      b2:    LDI R18,250
      b3:    NOP
             DEC R18
             BRNE b3
             DEC R17
             BRNE b2
             DEC R16
             BRNE b1

             DEC R0
             BRNE c1    ; Repeat 10 times

             POP R16    ; Pop the Original Status Register value from stack
             OUT SREG, R16
             RETI
```

About the Code

This Code uses External Interrupt 1 for connecting the push button.

- Interrupt Vector location is 0x0002
- Pull-Up resistor to be activated for PIND bit 3
- GICR value is 0x80
- MCUCR is set to 0x00 for low-level trigger

After setting the above values, now when we receive a trigger, we enter into the ISR. In ISR, first we Push SREG value into Stack. Then we initiate a counter of 10 and start toggling the output (PORTB pin 0) for every 1 second. For the exact algorithm, please check the Flowchart given at the end.

Code 2

```
#define F_CPU 1000000  // clock frequency

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT1_vect)
{
      int i;
      for (i=1;i<=10;i++) // for 10 times LED blink
      {
            PORTB=0x01;
            _delay_ms(1000);   // delay of 1 sec
            PORTB=0x00;
            _delay_ms(1000);
      }
}
int main(void)
```

```
{
      //Set the input/output pins appropriately
      //To enable interrupt and port interfacing
      //For LED to blink
      DDRD=0x00;   //Set appropriate data direction for D
      DDRB=0x01;   //Make PB0 as output
      MCUCR=0x00;  //Set MCUCR to level triggered
      GICR=0x80;   //Enable interrupt 1
      PORTB=0x00;
      PORTD=0x08;  // Activating Pull-Up resistor
      sei();       // global interrupt flag

      while (1)   // infinite loop
      {

      }
}
```

About the Code

This code is the C language version of Code 1 where it utilizes Interrupt 1 (INT1) of AVR to blink an LED 10 times with a delay of 1 second. For delays, we used internal libraries available for C language.

Code 3

```
.org 0x0000
rjmp reset

.org 0x0001
rjmp int0_ISR

.org 0x0100

reset:
        ;Loading stack pointer address
        LDI R16,0x70
        OUT SPL,R16
        LDI R16,0x00
        OUT SPH,R16

        ;Interface port B pin0 to be output so to view LED blinking
        SBI DDRB, 0
        LDI R16,0x00
        OUT DDRD,R16
        SBI PORTD, 2    ; Activating Pull-Up Resistor

        ;Set MCUCR register to enable low level interrupt
        OUT MCUCR,R16

        ;Set GICR register to enable interrupt 0
        LDI R16, 0x40
        OUT GICR, R16

        LDI R16,0x00
        OUT PORTB,R16

        SEI

; -------------------Main Program------------------
ind_loop: NOP
        rjmp ind_loop

; ------------------ISR Starts Here--------------------
int0_ISR:IN R16,SREG
             PUSH R16

             LDI R16,0x0A
             MOV R0,R16
             ;Modify below loops to make LED blink for 1 sec
      c1:    LDI R16,0x01
             OUT PORTB,R16

             LDI R16,8
      a1:    LDI R17,125
      a2:    LDI R18,250
      a3:    NOP
             DEC R18
             BRNE a3
             DEC R17
             BRNE a2
             DEC R16
             BRNE a1
             LDI R16,0x00
             OUT PORTB,R16

             LDI R16,8
      b1:     LDI R17,125
      b2:     LDI R18,250
      b3:     NOP
             DEC R18
             BRNE b3
             DEC R17
             BRNE b2
             DEC R16
             BRNE b1

             DEC R0
             BRNE c1

             POP R16
             OUT SREG, R16

             RETI
```

About the Code

This Code uses Interrupt 0 instead of Interrupt 1 like in Code 1. The changes made for changing INT1 to INT0 are as follows:

- Interrupt Vector location is 0x0001
- Pull-Up resistor to be activated for PIND bit 2
- GICR value is 0x40

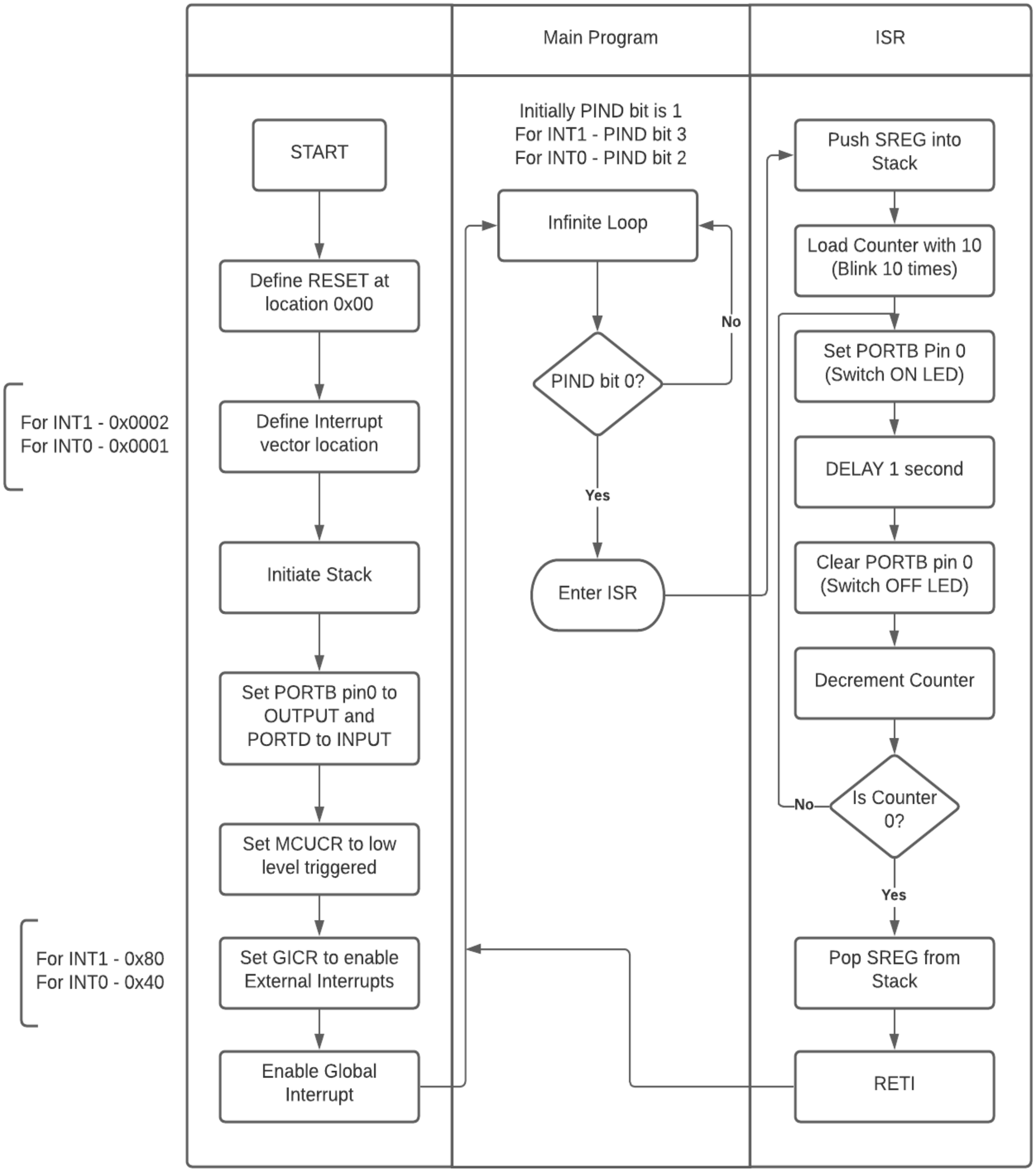
Code 4

```
#define F_CPU 1000000  // clock frequency

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT0_vect)
{
      int i;
      for (i=1;i<=10;i++) // for 10 times LED blink
      {
            PORTB=0x01;
            _delay_ms(1000);   // delay of 1 sec
            PORTB=0x00;
            _delay_ms(1000);
      }
}
int main(void)
{
      //Set the input/output pins appropriately
      //To enable interrupt and port interfacing
      //For LED to blink
      DDRD=0x00;   //Set appropriate data direction for D
      DDRB=0x01;   //Make PB0 as output
      MCUCR=0x00;  //Set MCUCR to level triggered
      GICR=0x40;   //Enable interrupt 0
      PORTB=0x00;
      PORTD=0x04;  // Activating Pull-Up resistor
      sei();       // global interrupt flag

      while (1) //wait
      {

      }
}
```

About the Code

This Code is the C-language version of Code 3.

Flowchart



<u>Note:</u> This flowchart was made by me using the Lucidchart software

Code Debugging

After building the solution of the code, follow these steps for Debugging:

1. Setup 4 breakpoints at - in main loop, beginning of ISR, beginning of first 1 sec delay and beginning of second 1 sec delay.
2. Start Debugging (Alt+F5).
3. Open the I/O window and the processor status window.
4. In the I/O window, make PIND bit 3 as 1. Now, try clicking on the continue button on the top. Notice that the control does not go into the ISR loop and stays in the main loop itself
5. Now, make the PIND bit 3 ‘0’ manually. This initiates the push button of the hardware. Again try clicking on the continue button and you can see that now the control switches to the ISR loop. Click on continue again and the control goes to the line before the first 1 sec delay.
6. Note the value of stopwatch in the processor status window and click on continue. Again, note the value in the stopwatch. You can see that the stopwatch is incremented by ~1 sec.
7. You can repeat the above step for the second 1 sec delay.

The One Second Delay

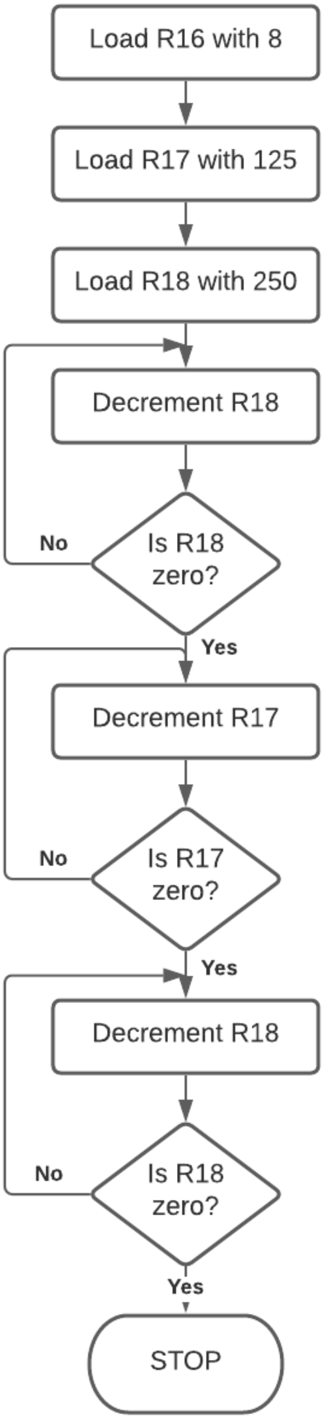
Here the one second delay is made using 3 nested loops. Since the simulator uses an oscillator frequency of 1MHz, we need to have 1000000 machine cycles to create a one second delay. Here’s how:

```
        LDI R16,8
a1:     LDI R17,125; 1
a2:     LDI R18,250; 1
a3:     NOP             ; 1
        DEC R18         ; 1
        BRNE a3         ; 2/1
        DEC R17         ; 1
        BRNE a2         ; 2/1
        DEC R16         ; 1
        BRNE a1         ; 2/1
```

For the inner loop, (4x250-1)x125x8x1 microseconds
For the middle loop, (4x125-1)x8x1 microseconds
For the outer loop, (4x8-1)x1 microseconds
Adding up all three we get, 1003023 microseconds ~ 1.003 seconds

To get a precise time delay, we have to use Timers of the AVR.

The Flowchart for the 1 second delay is given below:



Optional Code

```
.org 0x0000
rjmp reset

.org 0x0002
rjmp int1_ISR

.org 0x0100

reset:
        ;Loading stack pointer address
        LDI R16,0x70
        OUT SPL,R16
        LDI R16,0x00
        OUT SPH,R16

        ;Interface port B pin0 to be output
        ;so to view LED blinking
        SBI DDRB, 0

        LDI R16,0x00
        OUT DDRD,R16
        SBI PORTD, 3

        ;Set MCUCR register to enable low level interrupt
        OUT MCUCR,R16

        ;Set GICR register to enable interrupt 1
        LDI R16, 0x80
        OUT GICR, R16

        LDI R16,0x00
        OUT PORTB,R16

        SEI
ind_loop: NOP
        rjmp ind_loop

int1_ISR:IN R16,SREG
             PUSH R16
```

```
           LDI R16,0x02
           MOV R0,R16

           LDI R16,0x00

    c1:    LDI R17,HIGH(15625-1)
           OUT OCR1AH, R17
           LDI R17,LOW(15625-1)
           OUT OCR1AL, R17
           LDI R17,0
           OUT TCNT1H, R17
           OUT TCNT1L, R17

           OUT TCCR1A, R17
           LDI R17, 0x03
           OUT TCCR1B, R17

           COM R16
           OUT PORTB,R16          ; Toggle the OUTPUT

CHECK:     IN R17, TIFR
           SBRS R17,OCF1A
           RJMP CHECK
           LDI R17,1<<OCF1A
           OUT TIFR,R17
           LDI R17,0
           OUT TCCR1B,R17
           OUT TCCR1B,R17

           DEC R0
           BRNE c1

           POP R16
           OUT SREG, R16

           RETI
```

Inferences/Learnings from the Experiment

- Using External Interrupts of AVR
- Got a deeper understanding of the AVR I/O ports - DDRB register, PIN register, PORT register and using Pull-Up resistors
- Creating Delays using Nested loops and Timers. If we use Nested loops to create a delay, we'll get close to 1 second. To get an accurate delay, we use Timers
- Converting AVR assembly to C language. There are various internal libraries in C language which makes the program easier
- Learnt more about the Microchip Studio software
- Learnt how to make flowcharts for an AVR program

Link to all the Code Files

https://drive.google.com/file/d/1Pnuoa1tb1L559A7ab3mEHakDdfyJseBZ/view?usp=sharing

-----THANK YOU----