

EE2016 Lab

Experiment 2: Interrupts and Timers in Atmel
AVR Atmega

Manomukil T EE20B075

27th September 2021

Contents

1	<u>Aim:</u>	3
2	<u>Questions:</u>	3
3	<u>Solution:</u>	3
3.1	Question1:Fill in the blanks in the assembly code.	3
3.2	Question2:Redoing the above program with INT0	5
3.3	Rewrite the program in 'C' (int1). Rewrite the C program for int0.	7
3.3.1	C code with INT1	7
3.3.2	C code with INT0	8
3.4	Demonstrate both the cases (of assembly and C).	9
3.4.1	Flowchart:	10
3.4.2	Outputs:	11
4	<u>Conclusion:</u>	19

1 Aim:

Using Atmel AVR assembly language programming, implement interrupts and timers in Atmel Atmega microprocessor. To implement the blinking of an LED 10 times once switch is pressed in assembly and C programming using :

1. INT1 Interrupt
2. INT0 Interrupt

2 Questions:

1. Fill in the blanks in the assembly code
2. Use int0 to redo the same in the demo program (duely filled in). Once the switch is pressed the LED should blink 10 times (ON (or OFF) - 1 sec, duty cycle could be 50 %). Demonstrate both the cases.
3. Rewrite the program in 'C' (int1). Rewrite the C program for int0.
4. Demonstrate both the cases (of assembly and C).

3 Solution:

3.1 Question1:Fill in the blanks in the assembly code.

A textfile was given containing a program to enable interrupt1 with some of the code incomplete. The code has been filled and the entire code is displayed below:

```
;  
; INT1.asm  
;  
; Created: 26-09-2021 20:45:54  
; Author : ManomukilT  
;  
  
; Replace with your application code  
; Start  
.org 0;  
rjmp reset;  
  
.org 0x0002;  
rjmp int1_ISR;  
  
.org 0x0100;
```

```

reset:
    LDI R16,0x70; Loading Stack Point Address
    OUT SPL,R16;
    LDI R16,0x00;
    OUT SPH,R16;

    LDI R16,0x01; Interface PortB pin0 to be output
    OUT DDRB,R16; so to view LED Blinking

    LDI R16,0x00;
    OUT DDRD,R16;

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

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

    LDI R16, 0x00;
    OUT PORTB, R16;

    SEI;

ind_loop:
    RJMP ind_loop;

int1_ISR:
    IN R16, SREG;
    PUSH R16;

    LDI R16, 0x0A;
    MOV R0, R16;

    c1: LDI R16, 0x01;
    OUT PORTB, R16

    LDI R16, 0x05
    a1: LDI R17, 0xFF
    a2: LDI R18,0xFF
    a3: DEC R18
    BRNE a3
    DEC R17
    BRNE a2
    DEC R16
    BRNE a1

```

```

LDI R16, 0x00
OUT PORTB, R16

LDI R16, 0x05
b1: LDI R17, 0xFF
b2: LDI R18, 0xFF
    b3: DEC R18
    BRNE b3
    DEC R17
BRNE b2
DEC R16
BRNE b1

DEC R0
BRNE c1
POP R16
OUT SREG, R16

RETI

```

3.2 Question2:Redoing the above program with INT0

The same program has been implemented with interrupt0 and the code is displayed below:

```

;
; INT0_ASM.asm
;
; Created: 26-09-2021 20:45:54
; Author : ManomukilT
;

; Replace with your application code
;Start
.org 0;
rjmp reset;

.org 0x0001;
rjmp int0_ISR;

.org 0x0100;

reset:
    LDI R16,0x70; Loading Stack Point Address
    OUT SPL,R16;

```

```

LDI R16,0x00;
OUT SPH,R16;

LDI R16,0x01; Interface PortB pin0 to be output
OUT DDRB,R16; so to view LED Blinking

LDI R16,0x00;
OUT DDRD,R16;

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

LDI R16, 0x40; Set GICR Register to enable interrupt 1
OUT GICR, R16;

LDI R16, 0x00;
OUT PORTB, R16;

SEI;

ind_loop:
RJMP ind_loop;

int0_ISR:
IN R16, SREG;
PUSH R16;

LDI R16, 0x0A;
MOV R0, R16;

c1: LDI R16, 0x01;
OUT PORTB, R16

LDI R16, 0x05
a1: LDI R17, 0xFF
a2: LDI R18,0xFF
    a3:      DEC R18
    BRNE a3
    DEC R17
    BRNE a2
    DEC R16
    BRNE a1

LDI R16, 0x00
OUT PORTB, R16

LDI R16, 0x05

```

```

b1: LDI R17, 0xFF
b2: LDI R18, 0xFF
b3: DEC R18
BRNE b3
DEC R17
BRNE b2
DEC R16
BRNE b1

DEC R0
BRNE c1
POP R16
OUT SREG, R16

RETI

```

3.3 Rewrite the program in 'C' (int1). Rewrite the C program for int0.

3.3.1 C code with INT1

The code for the program with interrupt1:

```

/*
 * main.c
 *
 * Created: 27-09-2021 08:36:50
 * Author : ManomukilT
 */

#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=0x00;      //Make PB0 as output
    MCUCR=0x00;    //Set MCUCR to level triggered
    GICR=0x80;     //Enable interrupt 1
    PORTB=0x00;
    sei();         // global interrupt flag

    while (1) //wait
    {
        }
}

```

3.3.2 C code with INT0

The code for the program with interrupt0:

```

/*
 * main.c
 *
 * Created: 27-09-2021 08:36:50
 * Author : ManomukilT
 */

#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=0x00;      //Make PB0 as output
    MCUCR=0x00;     //Set MCUCR to level triggered
    GICR=0x80;      //Enable interrupt 1
    PORTB=0x00;
    sei();          // global interrupt flag

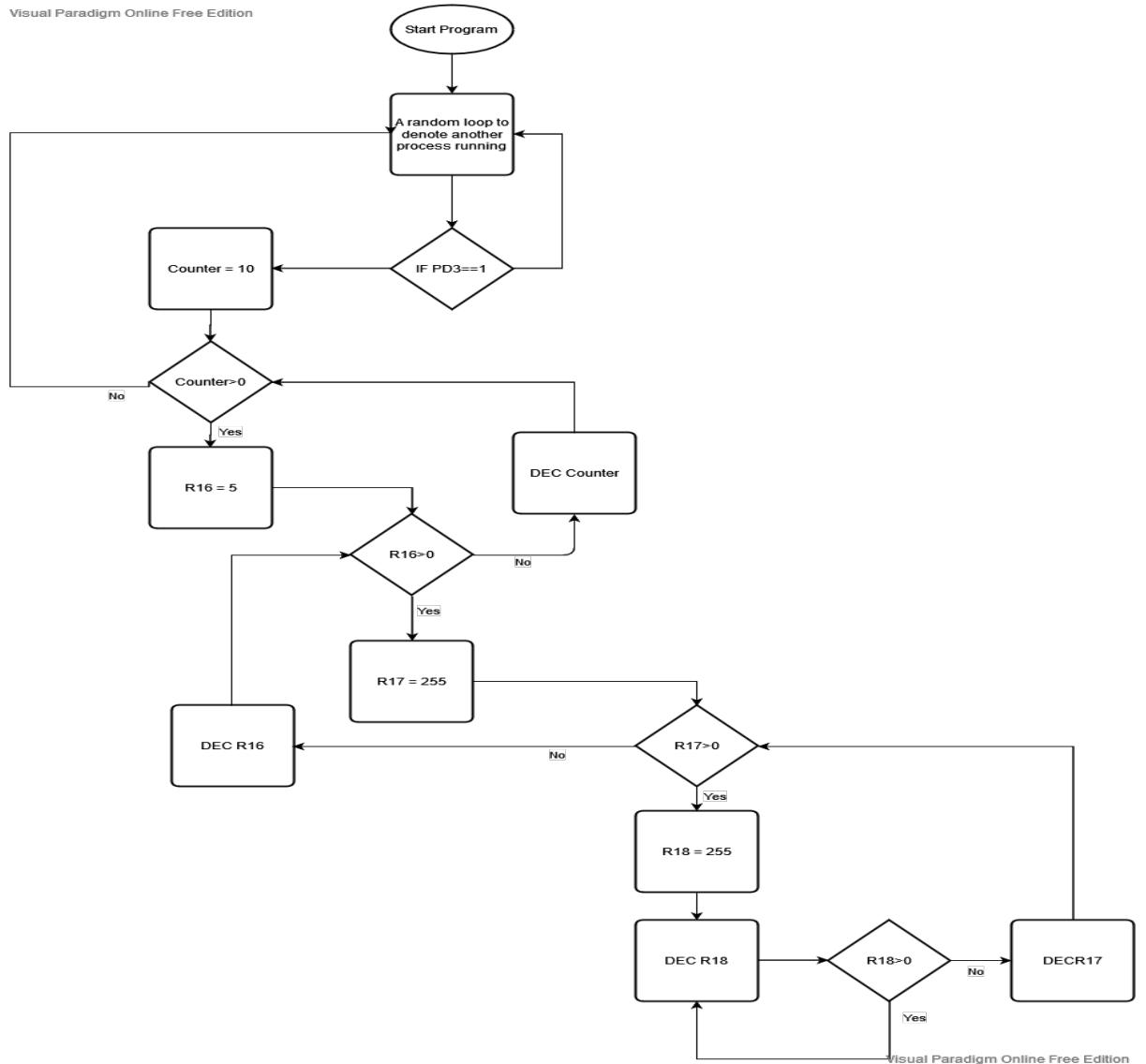
    while (1) //wait
    {
    }
}

```

3.4 Demonstrate both the cases (of assembly and C).

All the four programs have the same logic and flow. Only the functions for each program and language differs. The logic and flow of all the four programs is shown in the next page:

3.4.1 Flowchart:



3.4.2 Outputs:

The screenshot shows the Microchip Studio interface during debugging. The assembly code in the main window includes instructions like LDI R16, OUT MCUCR, and SEI. The Processor Status window shows the Program Counter at 0x0000010E, Stack Pointer at 0x0070, and various registers initialized to zero. The I/O window displays port pins PINE, DORB, and PORTB. The Memory window shows the program memory starting at address 0x0000.

```

    OUT MCUCR, R16;Set MCUCR register to enable low level interrupt
    LDI R16, 0x80;Set GICR Register to enable interrupt 1
    OUT GICR, R16;
    LDI R16, 0x00;
    OUT PORTB, R16;
    SEI;
    ind_loop:
    RJMP ind_loop;
int1_ISR:
    IN R16, SREG;
    PUSH R16;
    LDI R16, 0x0A;
    MOV R0, R16;
    C1: LDI R16, 0x01;
    OUT PORTB, R16

```

This screenshot shows the assembly code and processor status for a later point in the execution. The Program Counter is now at 0x00000114, indicating 29 cycles have passed. The registers show some changes, particularly R00 which is now 0x0A. The I/O and Memory windows remain similar to the first screenshot.

```

    OUT MCUCR, R16;Set MCUCR register to enable low level interrupt
    LDI R16, 0x80;Set GICR Register to enable interrupt 1
    OUT GICR, R16;
    LDI R16, 0x00;
    OUT PORTB, R16;
    SEI;
    ind_loop:
    RJMP ind_loop;
int1_ISR:
    IN R16, SREG;
    PUSH R16;
    LDI R16, 0x0A;
    MOV R0, R16;
    C1: LDI R16, 0x01;
    OUT PORTB, R16

```

The above outputs are for INT1.asm

The above outputs are for INT1.asm

INT1.C (Debugging) - Microchip Studio

```

PORTB=0x00; //Set appropriate data direction for D
DDRB=0x00; //Set appropriate data direction for D
PORTB=0x00; //Make PB0 as output
MCUCR=0x00; //Set MCUCR to level triggered
GICR=0x00; //Enable interrupt 1
PORTB=0x00;
sei(); // global interrupt flag
while (1) //wait
{
}

```

Processor Status

Name	Value
Program Counter	0x00000159
Stack Pointer	0x0458
X Register	0x0000
Y Register	0x0458
Z Register	0x0038
Status Register	0x0000
Cycle Counter	43
Frequency	1000 MHz
Stop Watch	43.00 µs

Registers

Reg	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00

I/O Properties

Memory 4

Call Stack Breakpoints Command Window Immediate Window Output

INT1.C (Debugging) - Microchip Studio

```

mainwhile: //For LED to blink
{
    PORTB=0x00; //Set appropriate data direction for D
    DDRB=0x00; //Set appropriate data direction for D
    PORTB=0x00; //Make PB0 as output
    MCUCR=0x00; //Set MCUCR to level triggered
    GICR=0x00; //Enable interrupt 1
    PORTB=0x00;
    sei(); // global interrupt flag
    while (1) //wait
    {
    }
}

```

Processor Status

Name	Value
Program Counter	0x0000015A
Stack Pointer	0x0458
X Register	0x0000
Y Register	0x0458
Z Register	0x0038
Status Register	0x0000
Cycle Counter	44
Frequency	1000 MHz
Stop Watch	44.00 µs

Registers

Reg	Value
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00

I/O Properties

Memory 4

Call Stack Breakpoints Command Window Immediate Window Output

The above outputs are for INT1.c

The screenshot shows the Microchip Studio IDE interface with the following details:

- Title Bar:** INT1_C (Debugging) - Microchip Studio
- Menu Bar:** File, Edit, View, VASIX, ASP, Project, Build, Debug, Tools, Window, Help
- Toolbar:** Standard icons for file operations, search, and build.
- Project Explorer:** Shows the project structure with files main.c and INT1_vect.c.
- Code Editor:** Displays the C code for the INT1 interrupt. The ISR function contains a delay loop of 10ms. The main function initializes pins.
- Processor Status Window:** Shows the processor state with the CPU Registers (PORTB) highlighted.
- Memory View:** Shows memory dump for the prog FLASH section.
- Call Stack:** Empty.
- Breakpoints:** One breakpoint is set at the start of the ISR.
- Registers:** Registers R00-R05 are listed with their current values.
- Watch Windows:** Watch 1 and Watch 2 are present.

The above outputs are for INT1.c

The screenshot shows the Microchip Studio interface with the following details:

- File Menu:** File, Edit, View, VASIX, ASF Project, Build, Debug, Tools, Window, Help.
- Toolbars:** Standard (File, Edit, View, Project, Build, Tools, Window), Debug, Hex, Registers, Stack, Memory, I/O, Properties.
- Registers Window:** Shows assembly code for the main.asm file. The code includes instructions like OUT MCUCR, LDRI, OUT GICR, LDRI, OUT PORTB, SEI, and a loop labeled `int_loop:`. The assembly code also includes comments: "Set MCUCR register to enable low level interrupt" and "Set GICR Register to enable interrupt 1".
- Processor Status Window:** Displays processor status information such as Program Counter (0x00000014), Stack Pointer (0x006D), X Register (0x0000), Y Register (0x0000), Z Register (0x0000), Status Register (bits 7-0: 11111111), Cycle Counter (29), Frequency (1.000 MHz), Stop Watch (29.00 µs), and Registers (R00-R06).
- Memory Window:** Shows a memory dump of the program FLASH starting at address 0x0000.prog. The memory contains several programs (prog_0x0000, prog_0x0016, prog_0x0032, prog_0x0048, prog_0x0064, prog_0x0080, prog_0x0096) each consisting of 16 bytes of FF (hex).
- Call Stack, Breakpoints, Command Window, Immediate Window, Output, Memory 4, Watch 1, Watch 2, Autos Locals, Stop, and Help windows:** These are standard windows found in most IDEs.

The above outputs are for INT0.asm

INT0_ASM (Debugging) - Microchip Studio

Registers

```
mainasm: .c
        LDI R16, 0x05
        LDI R17, 0xFF
        a2: LDI R18, 0xFF
        a3: DEC R18
        BRNE a3
        DEC R17
        BRNE a2
        DEC R16
        BRNE a1
        LDI R16, 0x00
        OUT PORTB, R16

        LDI R16, 0x05
        b1: LDI R17, 0xFF
        b2: LDI R18, 0xFF
        b3: DEC R18
        BRNE b3
        DEC R17
        BRNE b2
        DEC R16
        BRNE b1
```

Processor Status

Name	Value
Program Counter	0x0000011F
Stack Pointer	0x006D
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	0x0000
Cycle Counter	979246
Frequency	1.000 MHz
Stop Watch	979,246.00 µs
Registers	
R00	0x0A
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00

I/O

Memory 4

Watch 1

Call Stack Breakpoints Command Window Immediate Window Output Memory 4

Autos Locals Watch 1 Watch 2

Stopped

In 65 Col 1 Ch 1 INS

INT0_ASM (Debugging) - Microchip Studio

Registers

```
mainasm: .c
        ldi R16, 0x0d
        rjmp lnd_loop

        int0_ISR:
        in R16, SREG;
        push R16;

        ldi R16, 0x0A;
        mov R0, R16;

        c1: ldi R16, 0x01;
        out PORTB, R16

        ldi R16, 0x05
        a1: ldi R17, 0xFF
        a2: ldi R18, 0xFF
        a3: dec R18
        brne a3
        dec R17
        brne a2
        dec R16
        brne a1
```

Processor Status

Name	Value
Program Counter	0x00000114
Stack Pointer	0x006D
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	0x0000
Cycle Counter	1958466
Frequency	1.000 MHz
Stop Watch	19,584,666.00 µs
Registers	
R00	0x09
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00

I/O

Memory 4

Watch 1

Call Stack Breakpoints Command Window Immediate Window Output Memory 4

Autos Locals Watch 1 Watch 2

Stopped

In 65 Col 1 Ch 1 INS

The above outputs are for INT0.asm

INT0.C (Debugging) - Microchip Studio

```

1 int main(void)
{
    //Set the input/output pins appropriately
    //To enable interrupt and port interfacing
    //LED to blink
    DDRB=0x00; //Set appropriate data direction for D
    DDRB=0x00; //Make PBB as output
    MCUCR=0x00; //Set MCUCR to level triggered
    GICR=0x40; //Enable interrupt 0
    sei(); // global interrupt flag

    while (1) //wait
    {
    }
}

```

INT0.C (Debugging) - Microchip Studio

```

1 int main(void)
{
    //Set the input/output pins appropriately
    //To enable interrupt and port interfacing
    //LED to blink
    DDRB=0x00; //Set appropriate data direction for D
    DDRB=0x00; //Make PBB as output
    MCUCR=0x00; //Set MCUCR to level triggered
    GICR=0x40; //Enable interrupt 0
    sei(); // global interrupt flag

    while (1) //wait
    {
    }
}

```

The above outputs are for INT0.c

The above outputs are for INT0.c

4 Conclusion:

The following was learnt about through this experiment:

1. Using External Inputs with MCUCR and GICR
2. Interfacing Inputs and Outputs
3. Generating delays without the use of timers using nested loops
4. Programming Assembly code in C

Further an idea was established on how these concepts could be implemented for use in the real-world