CPE301 – SPRING 2019

Design Assignment 2C-Task2

Student Name: Ali Asadi

Student #: 5004200625

Student Email: aliasadivet@gmail.com

Primary Github address: https://github.com/aliasadivet/Submitions\_DA\_AliAsadi

Directory: DessignAssignments

The goal of the assignment is use GPIO and delays using Timers and Interrupts.

Programming

The following is the assembly code to implement Design Assignment 2C-Task1 using TIMER0\_OVF\_vect interrupt mechanism in normal mode.

;

; DA2C-T2.asm

;

; Created: 3/18/2019 5:32:23 AM

; Author : Ali Asadi

;

; Replace with your application code

.org 0

rjmp main

.org 0x010

rjmp Int\_int\_ASM ; ISR Handler

.org 0x100

Int\_int\_ASM:

in r18, SREG

push r18

ldi r18, 1

add r16, r18

clr r18

adc r17, r18 ; Increment registers r17:r16

pop r18

out SREG, r18

reti

main:

;stack initialization

ldi r16, high (RAMEND)

out SPH, r16

ldi r16, low (RAMEND)

out SPL, r16

rcall Timer\_Init\_int\_ASM

;rcall Pin\_Init\_int\_ASM ;make this line comment when executing the second part of the task

rcall Switch\_Init\_int\_ASM ;make this line instruction when executing the second part of the task

loop:

;rcall Pulse\_int\_ASM ;make this line comment when executing the second part of the task

rcall LED\_int\_ASM ;make this line instruction when executing the second part of the task

rjmp loop

Pin\_Init\_int\_ASM:

sbi DDRB, 2 ; sbi - set bit I/O - set bit 2 in DDRB register - Output Mode

sbi PORTB, 2 ; cbi - clear bit I/O - clear bit 2 in PORTB register - Output Low

ret ; return from the function

Timer\_Init\_int\_ASM:

ldi r16, 0x03 ; divide input clock by 64 => 8Mhz/64 ; CS02 = 0 CS01 = 1 CS00 = 1

out TCCR0B, r16 ; save it in register

ldi r16, 0x01 ; enable Overflow interrupt TOIE0

sts TIMSK0, r16 ; save it in register

sei ; Enable global interrupt

ret

Pulse\_int\_ASM:

ldi r24, low(435) ; ldi - load immediate load value of 87 in r24 - 435mS

ldi r25, high(435)

cbi PORTB, 2 ; set bit 2 in PORTB register - Output High

rcall Delay\_ASM ; rcall - relative call , call Delay\_ASM Function

sbi 0x05, 2 ; clear bit 2 in PORTB register - Output Low

ldi r24, low(290) ; load value of 58 in r24 - 145\*2 = 290mS

ldi r25, high(290)

rcall Delay\_ASM ; call Delay\_ASM Function

ret ; return from the function

Delay\_ASM:

lsr r25 ; r25:r24 contains delay in mS but clock generates a delay of 2.048mS

ror r24 ; so divide by 2 to get actual time of delays

sbiw r24, 4 ; subtract 4 from r25:r24 to compensate 0.048mS delay in each cycle

clr r16 ; clr r17:r16 which is the counter in ISR

clr r17

cp r24, r16 ; compare two resister r24-r16

cpc r25, r17 ; compare with carry r25-r17-C

brne PC-2 ; branch if not equal i.e r25:r24 and r17:r16 are not equal

ret ; exit or return

Switch\_Init\_int\_ASM:

cbi DDRD, 2 ; DDRD bit 2 = 0 INPUT Mode

sbi PORTD, 2 ; PORTD bit 2 = 1 PULLUP Enable

sbi DDRB, 2 ; PORTB.2 - Output

sbi PORTB, 2 ; PORTB.2 - Output Low

ret ; return from the function

LED\_int\_ASM:

sbis PIND, 2 ; skip if bit is set in I/O register, skip if HIGH state

rjmp pressed ; relative jump - jump to pressed label if State is low

brtc return ; branch if T bit is cleared - check the previous state

ldi r24, 10 ; load r24 with 2 to generate a delay of 10mS

ldi r25, 0

rcall Delay\_ASM ; call Delay\_ASM to generate debounce delay

sbis PIND, 2 ; check if the bit stays the same

rjmp return ; return if not

cbi PORTB, 2 ; PORTB.2 - Output High

ldi r24, low(1250) ; load r24 with 250 to generate a delay of 1250mS

ldi r25, high(1250)

sbiw r24, 10

rcall Delay\_ASM ; call delay

sbi PORTB, 2 ; PORTB.2 - Output Low

clt ; clear T , previous state is high

rjmp return ; return

pressed:

ldi r24, 10 ; if state is low, then generate a debounce delay of 10mS

ldi r25, 0

rcall Delay\_ASM ; call delay

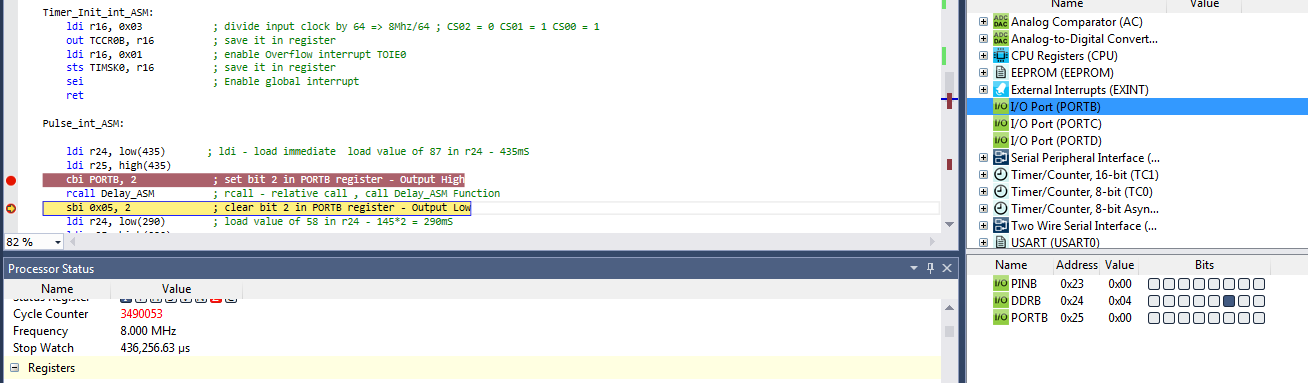
sbis PIND, 2 ; check if the bit stays the same

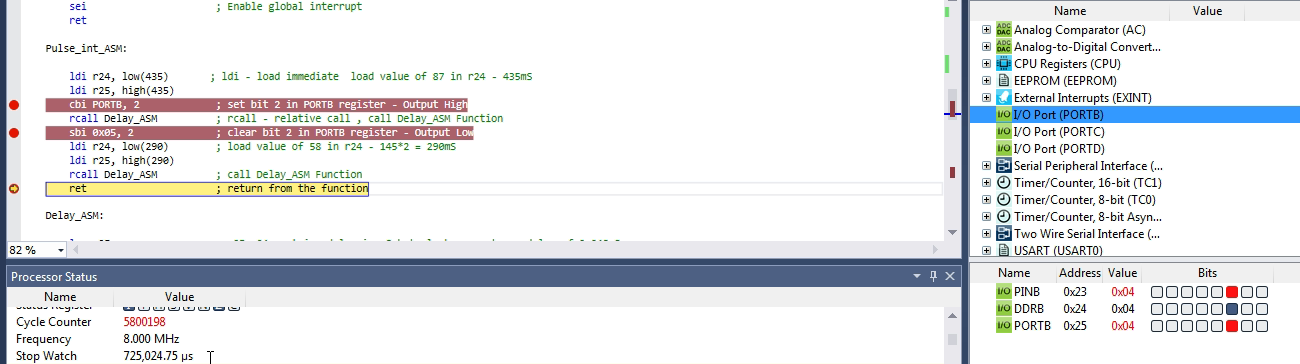
set ; set T bit to indicate previous state is Low

return:

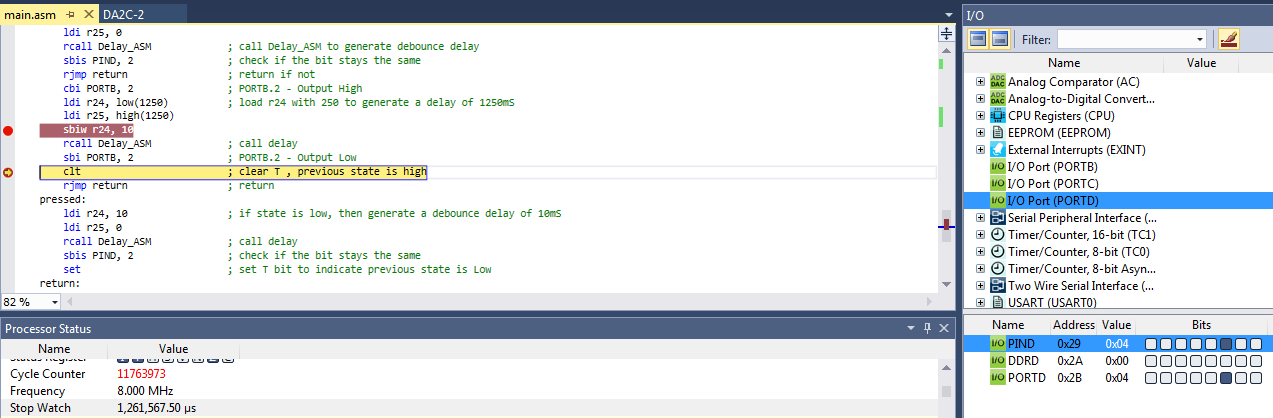
ret ; return from the function

The following screenshot demonstrates the simulation result of the code for the first part; 60% of the period and whole period respectively.





The following screenshot demonstrates the simulation result of the code for the second part; 1.25 sec delay.



The following is the C code to implement Design Assignment 2C-Task1 using

TIMER0\_OVF\_vect interrupt mechanism in normal mode.

/\*

\* DA2C-T2.c

\*

\* Created: 3/20/2019 4:13:12 AM

\* Author : Ali Asadi

\*/

#include <inttypes.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#include <avr/sleep.h>

#include <util/delay.h>

volatile *uint16\_t* tick;

ISR(TIMER0\_OVF\_vect)

{

tick++; // increment delay counter

}

void delay\_ovf(volatile *uint16\_t* ms)

{

tick = 0; // counter = 0

ms >>= 1; // divide by 2 since timer generates a delay of 2.048mS for each increment

ms -= 4; // subract 4 to compensate 0.048mS

while(tick < ms); // wait counter to complete given time delay

}

void Pulse\_Init\_ovf()

{

DDRB |= \_BV(2);

}

void Timer\_Init\_ovf()

{

TCCR0B = 0x03; // CS = 3 8Mhz/64

TIMSK0 = 0x01; // enble Timer Overflow interrupt

sei(); // enable global interrupt

}

void Pulse\_ovf()

{

PORTB |= \_BV(2);

delay\_ovf(435);

PORTB &= ~\_BV(2);

delay\_ovf(290);

}

void LED\_Init\_ovf()

{

DDRD &= ~\_BV(2);

PORTD |= \_BV(2);

DDRB |= \_BV(2);

PORTB |= \_BV(2);

}

void LED\_ovf()

{

static *uint8\_t* pressed = 0;

if((PIND & \_BV(2)) && pressed)

{

delay\_ovf(10);

if(PIND & \_BV(2))

{

PORTB &= ~\_BV(2);

delay\_ovf(1250);

PORTB |= \_BV(2);

}

pressed = 0;

}

else if(!(PIND & \_BV(2)))

{

delay\_ovf(10);

if(!(PIND & \_BV(2)))

pressed = 1;

}

}

void main()

{

Pulse\_Init\_ovf();

//LED\_Init\_ovf();

Timer\_Init\_ovf();

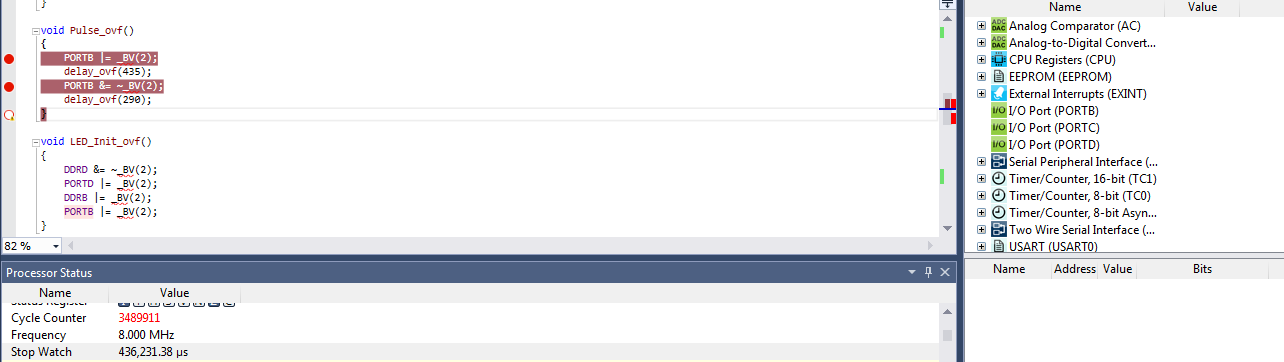
while(1)

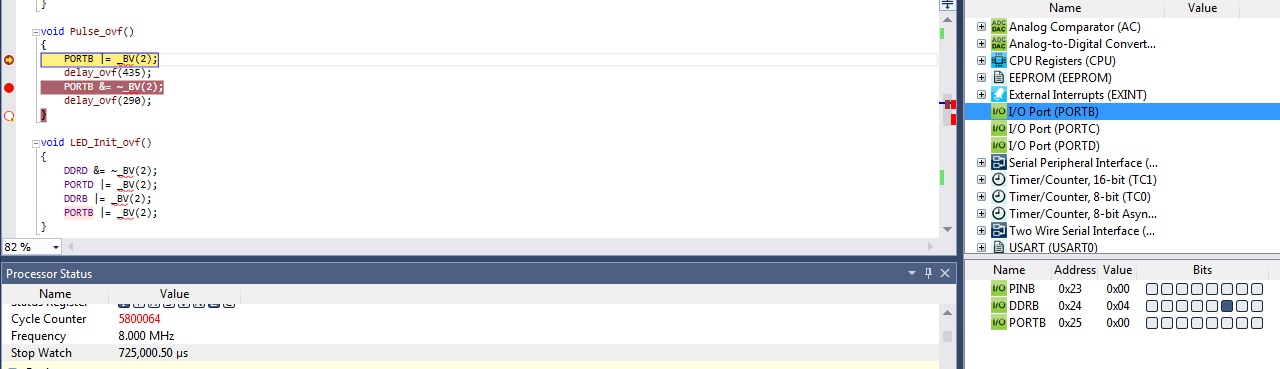
Pulse\_ovf();

//LED\_ovf();

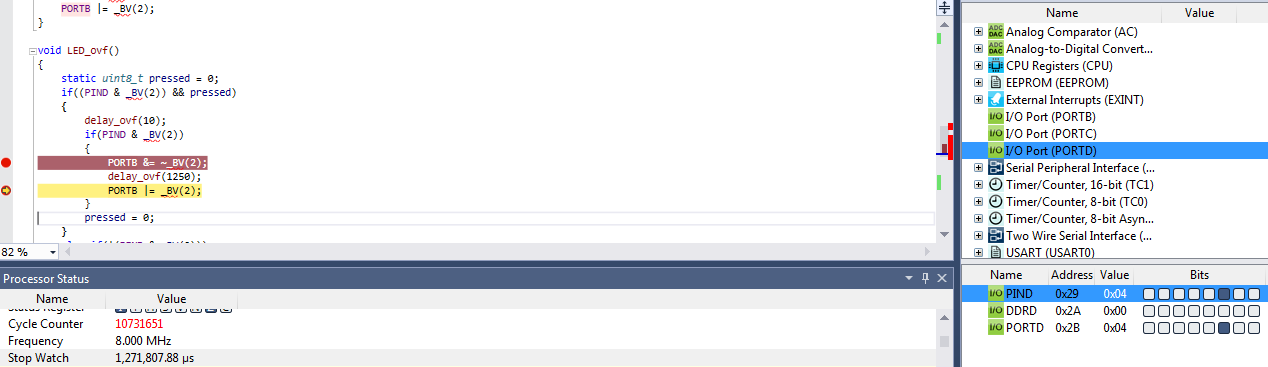
}

The following screenshot demonstrates the simulation result of the code for the first part; 60% of the period and whole period respectively.





As it is shown in the screenshots below, the delay time of 1.27 sec (slightly more than expexted time because of a small mistake in my calculations) has been executed.



The simulation and emulation tesults are posted on YouTube, and can be found on linkes below:

<https://www.youtube.com/watch?v=zTWkmLFCkgg>

<https://www.youtube.com/watch?v=TjDvhbE8t7U&t=9s>