ENEE 3587
HW 3

1. Write a subroutine that can generate a time delay from 1 to 100 s ~~using the modulus down counter~~. The number of seconds is passed to the subroutine. (Can't use the delay1ms function.)

   1s*16MHz = 16M >> 65535

Prescaler =	1024,	256,	64,	8
	16M/1024 = 15625	16M/256=62500	16M/64=250K	16M/8 = 20M

```
void delay1s(int k)
{
    TCCR1A = 0x00;            // configure Timer 1 to CTC mode with clock
    TCCR1B = 0b00001101;      // set prescaler to 1024, (CTC using OCR: WGM1:0 = 01)
    TCNT1 = 0;                // TCNT1 counts up from 0
    OCR1A = 15625-1;          // CTC TOP = OCR1A.
    TIFR1 = 0x02;             // clear OCF1A flag
    while(k)
     {
        while (!(TIFR1 & 2));     // wait for 1 s.
        TIFR1 = 0x02;             // clear OCF1A flag
        k--;                      // decrement count
    }
}
```

2. Write a program that measures the duty cycle of signal connected to ICP1. Assume that the signal period is less than 100ms.

   Note: ICP1 is on the chip but not on the board. You may use other Timer ICP (e.g ICP4).

   Period max = 100ms => period count = 100ms*16M = 1600K

Prescaler =	1024,	256,	**64**,	8
	1.6M/1024 = 1562.5	1.6M/256=6250	1.6M/64=**25000**	1.6M/8 = 200k

   Pmin = 100 cycles to be able to capture 1% increments of the duty cycle
   For a prescaler of 64 => 16M/fmax/64 = 100 => fmax = 16M/6400 = 2500Hz

   Idea: capture 2 rising edges then 1 falling edge.

   Duty cycle = ON time/Period
   = 100*(falling edge3 – rising edge 2)/(rising edge 2 – rising edge 1)

```
unsigned int re2, fe1, duty;
TCCR1A = 0;                 // Timer 1 to normal mode
DDRD &= ~0b00000100;        // ICP1 input (PD4)
TIFR1 = 0x2F;               // clear all flags related to Timer 1
TCCR1B = 0x43;              // capture rising edge, use prescaler = 64
while(!(TIFR1 & 0x20));     // wait for 1st rising edge
TCNT1 = 0;                  // rising edge1 = 0
TIFR1 = 0x21;               // clear ICF4 and OVF
while(!(TIFR1 & 0x20));     // wait for 2nd rising edge
re2 = ICR1;
TIFR1 = 0x21;               // clear ICF4 and OVF
TCCR1B = 0x03;              // capture falling edge, use prescaler = 64
while(!(TIFR1 & 0x20));     // wait for 2nd rising edge
fe1 = ICR1;
duty = ((fe1 - re2)*100)/re2;       // duty cycle percentage
while(1);
```

3. Assume that two signals having the same frequency are connected to the pins Timer1 and Timer3. Write a program to measure their phase difference. Phase difference = start of signal 1 - start of signal 2.

   Idea wait for rising edge of signal 1 then count until rising edge of signal 2.

```
unsigned int phase;
TCCR1A = 0;                 // Timer 1 to normal mode
DDRD &= ~0b00000100;        // ICP1 input (PD4)
TIFR1 = 0x2F;               // clear all flags related to Timer 1
TCCR1B = 0x41;              // capture rising edge, use prescaler = 1

TCCR3A = 0;                 // Timer 3 to normal mode
DDRE &= ~0b10000000;        // ICP1 input (PE7)
TIFR3 = 0x2F;               // clear all flags related to Timer 1
TCCR3B = 0x41;              // capture rising edge, use prescaler = 1


while(!(TIFR1 & 0x20));  // wait for rising edge of signal1
TCNT1 = 0;               // rising edge1 = 0
TIFR1 = 0x21;            // clear ICF4 and OVF

while(!(TIFR3 & 0x20));  // wait for rising edge of signal 2
phase = TCNT1;

while(0);
```

4. Write a program to generate a 10Hz, 20% duty cycle from OC5.

Non-PWM = OC
Period count = 16M/10 = 1.6M

Prescaler =	1024,	256,	64,	8
	1.6M/1024 = 1562.5	1.6M/256=6250	1.6M/64=25K	1.6M/8 = 200K

20% duty: 20% ON: HI count = 25k*.2 = 5k
80% OFF: LO count = 25*.8 = 20K

```
#define HIcnt 5000
#define LOcnt 20000

DDRL |= 0b0001000;     // configure OC1A pin for output (PL3)
TCCR5A = 0xC0;         // OC5A pin pull high on compare match
TCCR5B = 0x03;         // normal mode, prescaler 64
TCCR5C |= 0x80;        // force OC5A pin to high
TCCR5A = 0x40;         // select toggle as the OC5A pin action on compare match
OCR5A = TCNT5 + HIcnt;      // set HI count

while(1)                    // infinite loop
{    TIFR5 = 0x02;          // clear the OCF5A flag
     while (!(TIFR5 & 2));  // wait HI count, then toggle
     TIFR5 = 0x02;          // clear the OCF5A flag
     OCR5A = OCR5A + LOcnt; // set LO count
     while (!(TIFR5 & 2));  // wait LO count, then toggle
     OCR5A = OCR5A + HIcnt; // set HI count
}
```

5. Write a program to generate a 10KHz PWM signal that that starts at 0% duty cycle and increases to 100% in increments of 1% every 1ms.

   Fast-PWM based solution.

   User Timer 1, Fast PWM with OCR1A register for my count WGM1[3:0] = 1111
   OCR1A is used for duty cycle count => cant use OCA output => use OC1B output => PB6
   OC1B output should idle at LO so that count = duty cycle => COM1B[1:0] = 10

```
TCCR1A = 0b00100011;        // COM1B[1:0] = 10, WGM1[1:0]=11
TCCR1B = 0b00011001;        //WGM1[2:3] = 11
DDRB   = 0b01000000;        //PB6 -> OC1B
OCR1A  = 99;                //period  = 100
OCR1B  = 0;                 //0% duty
for (int i=0; i <= 100; i++)
{
    OCR1B = i;              //+1% duty
    _delay_1ms(1);
}
```

6. Write a program that uses the PWM to generate signal from 1Hz to 100KHz (50% duty cycle). The frequency should increase by 1Hz every second.

Prescaler:	1024,	256,	64,	8
1Hz:	16M/1024 = 15625	16M/256=62.5K	16M/64=250K	16M/8 = 2M
100KHz:	160/1024 = 0.15625	160/256=0.625	160/64=2.5	160/8= 2

```
long int period, duty;

TCCR1A = 0b00100011;          // COM1B[1:0] = 10, WGM1[1:0]=11
TCCR1B = 0b00011000;          //WGM1[2:3] = 11. Prescaler = 0
DDRB   = 0b01000000;          //PB6 -> OC1B

for (long int i=0; i < 100000; i++)
{
    period = 16000000/i
    if (period/1024 > 20)
    {   TCCR1B += 5;
        period /= 1024; }
    else if (period/256 > 20)
    {   TCCR1B += 4;
        period /= 256; }
    else if (period/64 > 20)
    (   TCCR1B += 3;
        period /= 64; }
    else
    {   TCCR1B += 2;
        period /= 8; }


    OCR1A  = (unsigned int) period - 1;
    OCR1B  = (unsigned int) (period+1)/2;

    _delay_1ms(1000);
}
```

7. Write a program that measures the duty cycle of a signal connected to the external interrupt pin 0.

   See discussion for algorithm:

```
long int edge2, edge3, state=0, OV;

void main()
{    int duty
     TCCR1A = 0;
     TCCR1B = 1;
     TCCR1C = 0;
     TIMSK1 = 1;                // enable TOV interrupt

     DDRD &= 0xFE;
     EICRA = 0x03;
     EIMSK = 1;
     sei();

     while (state < 3);
     duty = (100 * (edge3 - edge2))/edge2
     while(1);
}

ISR (TIMER1_OVF_vect)
{    tovCnt++;
}

ISR (INT0_vect)
{  if (state==0)                //first edge
     {          TCNT1=0;    //edge1 = 0
                TIMSK1=1;   //enable OV interrupts
                state = 1;      //look for second edge
     }
   else if (state==1)                           //second edge
     {          edge2 = TCNT1 + OV*65536;
                EICRA = 2;                      //next edge is a falling edge
                state = 2;
     }
   else                                         //state=2
     {          edge3 = TCNT1 + OV*65536;
                cli();                          //done. turn off interrupts globally
                state=3;
     }
}
```

8. Two buttons that generate pulses are connected to PC0 and PC20. Write a program that will increase the value in the variable frequency when PC0 pulse is detected and decrease the value in frequency when PC20 is detected.

PC0 -> PB0 => DDRB &= ~0x01
PC20 -> PK4 => DDRK &= ~0x10

INT0 detects toggles (rising edge or falling edge) on PC0-PC7
INT2 detects toggles on PC16-PC23

2 toggles = 1 pulse

```
long int frequency;
int edges0 =1, edges20=1;

void main()
{   DDRB &= ~0x01;          //PC0 (PB0)
    DDRK &= ~0x10;          //PC20 (PK4)
    PCICR = 3;              //enable INT0 and INT2
    PCMSK0 = 1;             //enable PC0
    PCMSK2 = 0x10           //enable PC20
    sei();
    while(1);
}

ISR (PCINT0_vect)
{   edges++;
    frequency += edges%2
}

ISR (PCINT2_vect)
{   edges20++;
    frequency -= edges20%2
}
```