TEAMMATES: Brandon Vo, Amy Qi, Brandon Marshall, Christian Crout
Due W 10/1

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.)

```
// HW 3 - Q1

#define F_CPU 16000000L //16MHz
#include <avr/io.h>
#include <util/delay.h>
#include <stdint.h>

// 1 sec delay
// 16 MHz / 1024 = 15.625 KHz = 15.625 cycles = 1 sec
// Period = 1/15.625 cycles = 0.000064 seconds per count
// 15.625 KHz * 0.000064 = 1 second


void delaySeconds(uint8_t seconds) {
    for (uint8_t i = 0; i < seconds; i++) {
        // 1. Configure Timer 1 with a prescaler of 1024
        TCCR1A = 0;
        TCCR1B = 0;
        TCCR1B = 0x05;
        TCNT1 = 0;

        // 2. Calculate the count value for 1 second delay
        // TCNT = bit max - (delay*F_CPU)/prescaler
        TCNT1 = 65536 - 15625;

        // 3. Clear overflow flag and Polling timer 1 for overflow flag se
        TIFR1 = 1;
        while (!(TIFR1 & 1));

        // 4. Clear the overflow flag
        TIFR1 = 1;
    }
}
```

2. Write a program that measures the duty cycle of signal connected to IC1. Assume that the signal period is less than 100ms.

```c
#include <avr/io.h>
#include <stdint.h>

uint16_t T1, T2, T3;
uint16_t duty_cycle;

int main(void) {

    DDRL &= 0xFE; // Set PL0 as input (ICP4)
    TCCR4A = 0x00; // Set Timer 4 to normal mode
    TCCR4B = 0x43; // Capture on rising edge, prescaler = 64
    TIFR4 = 0x2F;  // Clear all flags related to Timer 4
    TCNT4 = 0; // Count up from 0

    // Wait for 1st rising edge
    while(!(TIFR4 & 0x20));
    T1 = ICR4;
    TIFR4 = 0x20; // Clear input capture flag

    // ICES4=0 to capture falling edge
    TCCR4B &= ~0x40;
    TIFR4 = 0x20;

    // Wait for falling edge
    while(!(TIFR4 & 0x20));
    T2 = ICR4;
    TIFR4 = 0x20;

    // ICES4=1 to capture rising edge
    TCCR4B |= 0x40;
    TIFR4 = 0x20;

    // Wait for 2nd rising edge
    while(!(TIFR4 & 0x20));
    T3 = ICR4;
    TCCR4B = 0x00; // Stop Timer 4
    TIFR4 = 0x20;

    // Calculate duty cycle
    duty_cycle = ((uint16_t)(T2 - T1) / (uint16_t)(T3 - T1)) * 100;

    while(1);
}
```

3. Assume that two signals having the same frequency are connected to the pins PT1 and PT0. Write a program to measure their phase difference. Phase difference = start of signal 1 - start of signal 2.

```c
#include <avr/io.h>
#include <stdint.h>

uint16_t signal1, signal2;
int16_t phase_difference;

int main(void) {

    DDRL &= ~0x03; // Set PTL0 and PTL1 to 0 for input

    TCCR4A = 0x00; // Set Timer 4 to normal mode
    TCCR4B = 0x42; // Capture on rising edge, prescaler = 8

    TCCR5A = 0x00; // Set Timer 5 to normal mode
    TCCR5B = 0x42; // Capture on rising edge, prescaler = 8

    while (1) {

        while(!(TIFR4 & 0x20));  // Wait for rising edge for Timer 4
        signal1 = ICR4;
        TIFR4 = 0x20; // Clear flag

        while(!(TIFR5 & 0x20)); // Wait for rising edge for Timer 5
        signal2 = ICR5;
        TIFR5 = 0x20; // Clear flag

        phase_difference = signal1 - signal2; // Calculate phase difference

        if (phase_difference < 0) {
            phase_difference = -phase_difference; // Obtain the absolute value for phase difference
        }
    }
}
```

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

```c
#include <avr/io.h>
#include <stdint.h>

int main(void) {

    DDRL |= 0x08; // Set PTL3 for OC5B output

    TCCR5A |= 0x23; // Fast PWM mode, clear OC5B compare match, OCR5A as TOP (0b00100011)
    TCCR5B |= 0x1D; // Fast PWM mode, prescaler is 1024 (0b00011101)
    TCNT5 = 0; // Force TCONT5 to count up from 0

    OCR5A = 1561; // TOP = (16MHz/1024*10Hz) -1 = 1561.5
    OCR5B = 312; // Duty cycle = 1561*0.2 = 312.2

    while (1);
}
```

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.

```c
#include <avr/io.h>
#include <util/delay.h>

int main(void) {

    uint16_t duty_cycle = 0;
    DDRB |= 0x02;  // PTB1 is for OC1B output
    TCCR1A |= 0x23;  // Fast PWM mode, clear OC1B compare match, OCR1A as TOP (0b00100011)
    TCCR1B |= 0x19;  // Fast PWM mode, no prescaler (0b00011001)

    OCR1A = 1599; // TOP = 16MHz/(10kHz * 1) - 1 = 1599
    OCR1B = 0; // // Initial duty cycle set to 0%

    while (1) {
        _delay_ms(1);

        if (duty_cycle < 100) { // Increment duty cycle
            duty_cycle++;
            OCR1B = (uint16_t)(duty_cycle * OCR1A / 100 ); // Calculate duty cycle
        }
    }
}
```

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.

```c
#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 16000000UL

int main(void) {

    uint32_t frequency = 1;
    DDRB |= 0x02; // // PTB2 is for OC1B output
    TCCR1A = 0x23; // Fast PWM, clear OC1B compare match, OCR1A as TOP (0b00100011)
    TCCR1B = 0x19; // Fast PWM mode, initialized with no prescaler (0b00011001)

    while (1) {
        if(frequency <= 31) {
            TCCR1B = (TCCR1B & 0xF8) | 0x05; // Prescaler is 1024
            OCR1A = F_CPU / (1024 * frequency) - 1;}
        else if(frequency <= 250) {
            TCCR1B = (TCCR1B & 0xF8) | 0x04; // Prescaler is 256
            OCR1A = F_CPU / (256 * frequency) - 1;}
        else if(frequency <= 2000) {
            TCCR1B = (TCCR1B & 0xF8) | 0x03; // Prescaler is 64
            OCR1A = F_CPU / (64 * frequency) - 1;}
        else if(frequency <= 16000) {
            TCCR1B = (TCCR1B & 0xF8) | 0x02; // Prescaler is 8
            OCR1A = F_CPU / (8 * frequency) - 1;}
        else {
            TCCR1B = (TCCR1B & 0xF8) | 0x01; // Prescaler is 1
            OCR1A = F_CPU / frequency - 1;}

        OCR1B = OCR1A * 0.5; // Duty cycle = 50%
        _delay_ms(1000);

        if(frequency < 100000) {
            frequency++;  // Increment frequency until 100kHz
        }
    }
}
```

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

```c
// HW 3 - Q7

#define F_CPU 16000000L //16MHz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdint.h>

// Use volatile variables when dealing with interrupts and concurrent/critical section code
volatile uint16_t rising_edge_time = 0;
volatile uint16_t falling_edge_time = 0;
volatile uint16_t period_time = 0;
volatile uint16_t high_time = 0;
volatile uint8_t duty_cycle = 0;

// Pin 0 interrupt
ISR(INT0_vect) {
    if (PIND & 0x01) {

    // Rising edge detected
    TCNT1 = 0;
    rising_edge_time = TCNT1;

    } else {
    // Falling edge detected
    falling_edge_time = TCNT1;
    high_time = falling_edge_time - rising_edge_time;
    period_time = falling_edge_time;

    // Calculate Duty Cycle
    if (period_time > 0) {
        duty_cycle = (high_time * 100) / period_time;
    }
}

int main(void){

    // PD0 = Input
    DDRD &= ~0x01;

    // Pull-up PD0
    PORTD |= 0x01;

    // External interrupt on any edge
    EICRA |= 0x01;

    // Enable external interrupt INT0 by setting bit 0
    EIMSK |= 0x01;

    // Configure Timer1, Normal mode, no prescaler
    TCCR1A = 0x00;
    TCCR1B = 0x01;
    sei();


    _delay_ms(1000);
    cli();
    while(1);

}
```

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.

```c
// HW 3 - Q8

#define F_CPU 16000000L //16MHz
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdint.h>

// Use volatile variables when dealing with interrupts and concurrent/critical section code
volatile uint16_t frequency = 0;

// Interrupt for PB0 (PCINT0)
ISR(PCINT0_vect) {
    if (PINB & 0x01) {
        frequency++;
    }
}

// Interrupt for PB0 (PCINT20)
ISR(PCINT2_vect) {
    if (PIND & 0x10) {
        frequency--;
    }
}

int main(void){

    // PB0, PD4 = Input
    DDRB &= ~0x01;
    DDRD &= ~0x10;

    // Pull-ups
    PORTB |= 0x01;
    PORTD |= 0x10;

    // Enable PCINT 2
    PCICR |= 0x05;

    //  Pin change interrupt enable
    PCMSK0 |= 0x01;
    PCMSK2 |= 0x10;

    sei();
    _delay_ms(1000);
    cli();
    while(1);

}
```