Name: Lab Section:

CprE 288 Fall 2012 – Homework 8
Due Thu. Oct. 25 in the class

Notes:

- **Start early on homework.**
- Homework answers must be typed using a word editor. Hand in a hard copy in the class, before or at the end of the lecture.
- Late homework is accepted within three days from the due date. **E-mail late homework to both of the grading TAs, Min Sang Yoon (my222@iastate.edu) and Zhen Chen (zchen@iastate.edu)**. *Late penalty is 10% per day (counting from 10:45am of the due date if you are in the morning class or 2:10pm if you are in the afternoon class).*

Question 1: Output Compare Modes (5 pts)

Describe how TCNT changes under the Normal Mode, the CTC Modes, and the fast PWM Modes.

TCNT increments by one every clock cycle. When it reaches the TOP, it's reset to the BOTTOM (0) in the next clock cycle.

In the Normal Mode, the TOP is fixed at 0xFFFF.

In the CTC mode, the TOP is the value stored in OCRnA (if WGM3:0 = 4) or the ICRn (WGM3:0 = 12) of the Timer/Counter.

In the fast PWM mode, the TOP can be fixed at 0x00FF (WGM=0101, 8-bit fast PWM), 0x01FF (WGM=0110, 9-bit fast PWM), 0x03FF (WGM=0111, 10-bit PWD), or the value stored in the ICRn (WGM=1110) or OCRnA (WGM=1111).

Grading:

1. It's OK if one just means the TOP values but not the WGM value for each mode.
2. Give partial credit when appropriate.

Question 2: Output Compare Normal Mode (15 pts)

In lab 8 you are suggested to use a fast PWM mode of *Timer/Counter 3* to control the servo (WGM=0b1111). Assume now you are required, for certain reason, to do the same job using the **Normal Mode** (WGM=0b0000). The output channel is *channel A*.

Recall that the Normal Mode may be used to generate a waveform of any shape, but it incurs relatively high CPU overhead and may not work if the frequency of waveform events is close to the processor frequency. The waveform for controlling the servo is of low frequency (the minimum event interval is about 1.0ms), so it's doable to use the Normal Mode even though it's not efficient.

The general idea is to have two functions, move_servo() and ISR(TIMER3_COMPA_vect), to set the two PWM parameters and to generate the PWM waveform using the parameters, respectively. The two functions communicate with each other through two global variables, pulse_width and period_length, for the two PWM parameters.

Complete the following two functions, plus another function that initializes Timer/Counter 3. See the comments for additional details.

```
// PWM parameters: pulse width and period length
volatile unsigned short pulse_width;
volatile unsigned short period_length;

// YOU MAY ADD MICROS AND GLOGAL VARIABLES HERE
#define PRESCALAR           64
// GRADING: One may use a different prescalar value

// define the number of clock cycles for 1ms
#define ms_clocks           ((unsigned long)16000000/PRESCALAR/1000);

// state of PWM waveform generation
volatile enum {
   LOW,
   HIGH
} state;

/* Initialize timer/counter 3 to set servo at a desired
   angular position given by "degree". Assume that the pulse
   width should be 1.0ms + (degree/180)*1.0ms. Use the
   same prescaler value as you used in lab 8. The servo should
   be positioned at 90 degree initially. Enable interrupt(s)
   properly. */
void init_servo ()
{
  // YOUR CODE
  // HINT: First decide on WGM, COM, CS, and FOC bits

  // Initialize pulse_width and pulse_length
  pulse_length = 25*ms_clocks - 1;     // 25ms pulse period
  pulse_wdith = ms_clocks*3/2 - 1;     // 1.5ms pulse width

  // Set pin PE3 to be output, it's alternative function is OC3A
```

Name: Lab Section:

```
  DDRE |= _BV(3);
  // GRADING: -1 if one uses _BV(4) or similar. _BV(4) is for using
  // channel B.

  // Set up the first event
  // GRADING: OK if one doesn't do it. It's just a good practice.
  OCR3A = TCNT3 + 100;      // any short delay will work

  // Assume prescalar of 64 (CS=011)
  // WGM = 0000, COM3A = 01 (toggle), OCIE3A=1
  // No write to TCCR3C
  TCCR3A = _BV(COM3A0);
  TCCR3B = _BV(CS31) | _BV(CS30);
  ETIMSK |= _BV(OCIE3A);

  // GRADING: -1.5 if one uses the COM setting for non-inverting PWM
  // waveform. Recall that the meaning of COM setting is
  // dependent on the WGM selection; we studied two tables for COM.
  // GRADING: One may try to set the initial value of the OC pin
  // by writing FOC3A. The solution assumes the initial OC pin to
  // be at the low level.
  // GRADING: OK if one uses a different prescalar from lab 8 code.
}

/* The ISR generates the PWM waveform of pulse_width and
      period_length */
ISR (TIMER3_COMPA_vect)
{
   // YOUR CODE.
   // HINT: Use a finite state machine (FSM) of two states; you may use
   //       a switch statement to implement a FSM.

   switch (state)
     {
       case LOW:
         OCR3A = OCR3A + pulse_width;
         state = HIGH;
         break;

       case HIGH:
         OCR3A = OCR3A + (period_length - pulse_width);
         state = LOW;
         break;
     }
```

Name: Lab Section:

```
    // Grading: One may update the COM bits to ensure the right
    // low/high level. That's OK if done correctly.
}

/* Move the servo to the desired angular position */
void move_servo (unsigned degree)
{
  // YOUR CODE
  cli();
  pulse_wdith = ms_clocks
               + degree*ms_clocks/180 - 1;// 1.5ms pulse width
  sei();

  // GRADING: 1) There is no need to change pulse_width. If one does,
  // it's OK. 2) The use of cli() and sei() is a good practice
  // but it's OK if one doesn't use it for this
  // particular code. 3) The sequence of operations in
  // "degree*ms_clocks/180" is important, i.e. multiply first then
  // divide
}
```

Question 3: Interrupt Overhead in the Normal Mode (5 pts)

This question continues Question 2. Assume the ISR takes 150 cycles for each time of execution. What percentage of CPU time would have been spent in the ISR if your lab 8 were implemented as in Question 2?

The PWM period is 25.0ms in the previous code. Interrupt occurs twice per PWM period, and for each ISR execution the overhead is 150 X 1/16MHz = 9.375 microseconds. The overhead ratio is 9.375 microseconds over 25.0 milliseonds, which is 0.0375%.

GRADING: The PWM period can be different. Note that PRESCALAR is irrevelant.

Question 4: Output Compare CTC Mode (10 pts)

Assume that, for certain reason, you are required to do the programming of Question 1 but using the **CTC Mode** with OCRnA as the TOP (WGM=0b0100). Revise your code of Question 1. You only need to revise init_servo() and ISR(TIMER3_COMPB_vect). Put your revised code in the following code template.

Name: Lab Section:

HINT: The CTC Mode is supposed to generate square waveform without using interrupts. However, it can also be used with interrupts like the Normal Mode. The major difference is that in the CTC Mode, when a match event happens, the TCNT of the Timer/Counter is reset to zero. In the Normal Mode, the TCNT continues to increment without being reset.

```
/* Initialize timer/counter 1 to set servo at a desired
   angular position given by "degree". Assume that the pulse
   width should be 1.0ms + (degree/180) * 1.0ms. Use the
   same prescaler value as you used in lab 8. The servo should
   be positioned at 90 degree initially. Enable interrupt(s)
   properly. */
void init_servo ()
{
  // YOUR CODE:

  // Set pin PE3 to be output, it's alternative function is OC3A
  DDRE |= _BV(3);
  // GRADING: -1 if one uses _BV(4) or similar. _BV(4) is for using
  // channel B.

  // Set up the first event
  // GRADING: OK if one doesn't do it. It's just a good practice.
  OCR3A = TCNT3 + 100;      // any short delay will work

  // Assume prescalar of 64 (CS=011)
  // WGM = 0100, COM3A = 01 (toggle), OCIE3A=1
  // No write to TCCR3C
  TCCR3A = _BV(COM3A0);
  TCCR3B = _BV(WGM32) | _BV(CS31) | _BV(CS30);
  ETIMSK |= _BV(OCIE3A);

  // GRADING: One may try to set the initial value of the OC pin
  // by writing FOC3A. The solution assumes the initial OC pin to
  // be at the low level.
}

/* Move the servo to the desired angular position */
void move_servo (unsigned degree)
{
  // YOUR CODE
  // YOUR CODE
  cli();
  pulse_wdith = ms_clocks
```

Name: Lab Section:

```
                 + degree*ms_clocks/180 - 1;// 1.5ms pulse width
  sei();

  // GRADING: There is no need to change this code (from the
  // previous question).
}

/* The ISR generates the PWM waveform of pulse_width and
      period_length */
ISR (TIMER3_COMPA_vect)
{
   // YOUR CODE. You may assume the initial output is low.
   // HINT: Use a finite state machine of two states; you may use
   //       a switch statement to implement it.

   switch (state)
     {
       case LOW:
         OCR3A = pulse_width - 1;
         state = HIGH;
         break;

       case HIGH:
         OCR3A = (period_length - pulse_width) - 1;
         state = LOW;
         break;
     }
}
```

Grading:

1. Note the subtle difference in the ISR: "+=" vs. "=", with and without "-1"
2. One may try to update COM bits insider the ISR. That's OK if it's done correctly.
3. One may update the COM bits to ensure the right low/high level. That's OK if done correctly.

Question 5: Output Compare PWM Mode (15 pts)

Assume that you are programming Timer/Counter 1 of an ATmega128 to generate fast PWM waveforms to control the brightness of **three** LED (light-emitting diode) lights. The brightness is proportional to the duty cycle of the PWM waveform. The required resolution is 8-bit, e.g.

there should be 256 steps of brightness. Each LED light is controlled individually. The PWM period must be less than short than 1ms.

a. [2] What clock prescalar value(s) can you use for this application? Explain your choice. You don't have to give all valid choices.

The minimum PWD period is 256 cycles. Assume the default system clock of 16MHz. We can create the following table:

Prescalar	1	8	64	256	1024
Clock	16MHz	2MHz	250KHz	62.5KHz	15.625KHz
Cycle Time	1/16us	0.5us	4us	16us	64us
Min PWM Period of 256 cycles	16us	128us	1.024ms	4.096ms	16.384ms

Note: us is microseconds

Prescalar 1 and 8 are good choices.

Grading: One answer is good enough.

b. [3] What fast PWM mode(s) can you use for this application? Explain your choice. You don't have to give all valid choices.

There are five fast PWM mode:

WGM	Description	TOP
0101	8-bit fast PWM	0x00FF
0110	9-bit fast PWM	0x01FF
0111	10-bit fast PWM	0x03FF
1110	Fast PWM with TOP in ICRn	ICRn
1111	Fast PWM with TOP in OCRnA	OCRnA

WGM=1111 cannot be used: We have to use all three channels, but in that mode channel A cannot be used. All other modes can be used. A good choice is WGM=0101, whose resolution is sufficient for this application.

Grading: One choice is sufficient.

c. [10] Write C code for this application, using the following template. Function `init_timer()` initializes Timer/Counter 1, and function `set_brightness()` sets the brightness level of a given LED light.

Name: Lab Section:

```
// LED index
enum LED {LED_A, LED_B, LED_C};

// GRADING: 6 points init_timer(), 4 points on set_brightness().

// Initialize Timer/Counter 1 for controlling the three LEDs
init_timer()
{
  // YOUR CODE

  // Set pins PB5, PB6, PB7 to be output, their alternative functions
  // are OC1A, OC1B, OC1C.
  DDRB |= BV(5)|BV(6)|_BV(7);
  // GRADING: -1 if one sets wrong pins

  // Use prescalar of 8 (CS=010)
  // WGM = 0101, COM1A=COM1B=COM1C=10 (non-inverting mode)
  // NOTE: Set COM bits for all channels so that they run together
  // No write to TCCR1C and TIMSK
  TCCR1A = _BV(COM1A1)|_BV(COM1B1)|_BV(COM1C1)_BV(WGM30);
  TCCR1B = _BV(WGM32) | _BV(CS32);

  // Reset TCNT1. GRADING: OK without it, it's not critical.
  TCNT1 = 0;

  // GRADING:
  //  1. -2.5 if one sets only one set of COM bits, e.g only COM1A
  //  2. -1 if one enables interrupt. No need for that.
}

// Set the brightness of LED n to brightness level k
set_brightness(enum LED n, int k)
{
  // YOUE CODE
  switch (n)
    {
    case LED_A:
      OCR1A = k;
      break;
    case LED_B:
```

Name: Lab Section:

```
      OCR1B = k;
      break;
    case LED_C:
      OCR1C = k;
      break;
    }
    // Grading: If one writes “k-1” instead of “k”, accept it.
    // The question left a room on how to interpret k.
}
```