Name: Sean Gordon **Lab Section:** F

CprE 288 Fall 2018 – Homework 7
Due Sunday. October 28 (on Canvas 11:59pm)

Notes:

- Homework must be typed and submitted as a PDF or Word Document (i.e. .doc or .docx) only.
- If collaborating with others, you must document who you collaborate with, and specify in what way you collaborated (see last page of homework assignment), review the homework policy section of the syllabus: http://class.ece.iastate.edu/cpre288/syllabus.asp for further details.
- Review University policy relating to the integrity of scholarship. See ("Academic Dishonesty"): http://catalog.iastate.edu/academic_conduct/#academicdishonestytext
- Late homework is accepted within two days from the due date. *Late penalty is 10% per day.* ***Except on Exam weeks****, homework only accepted 1 day late.*
- ***Note:*** *Code that will not compile is a typo. Answering a question as "will not compile"* ***will be marked incorrect****. Contact the Professor if you think you have found a typo.*
- **Note: You are not allowed to use any MACROs in your code, except for register names.**
  - Example: You will lose points for: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | PIN1
  - Must use: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | 0b0000_0010; // or 0x02

Note: Unless otherwise specified, all problems assume the TM4C123 is being used

Question 1: General Timer questions (10 pts)

a) Briefly describe each of the Timer modes given in Table 9.1 of the textbook. (6pts)
One-shot mode counts up or down until it reaches the timeout value, where it clears the TEN bit.
Periodic mode does the same but upon timeout the timer starts counting again from initial value.
In capture mode, the timer holds count of the amount of input events while GPTMTnV and GPTMTnPV hold the free-running timer value and the free-running prescaler value.

b) For the GPTM Timer Mode Register (GPTMTnMR), under what Timer usage scenarios does it make sense to have the TAMRSU bit set to 0, how about set to 1? (2pts)
When creating a PWM with multiple different match locations, TAMRSU needs to be set to 0.
If said PWM should only be changed once per clock cycle, TAMRSU can be set to 1.

c) Under what condition will the TBTORIS bit be set in the GPTM Raw Interrupt Status Register (GPTMRIS)? (2pts)
TBORIS will be set once timer B hits timeout, such as 0 in count-down mode or the set cap in count-up.

Name: Sean Gordon **Lab Section:** F

Question 2: Don't Go into the Light! (25 pts)

Complete the program below to have a robot move away from a light source. The robot has two wheels, similar to the robot used in lab, and has a light sensor on each side. See figure.

Motor Control: Assume that you are programming Timer 1 module A (for left motor) and Timer1 module B (for right motor) to generate PWM waveforms to control the speed of each wheel's motor (**note, connect Timer 1 to Port B**). The speed of the motor is proportional to the percentage of time the PWM signal is high (i.e. PWM duty cycle).

Note: Your PWM wave must have a period of **1ms**. Note the system clock is 16 MHz.

Light Sensors: The light sensors are connected to **Channel 1** (left sensor) and **9** (right sensor) of the ADC as single channel inputs (i.e. not differential)

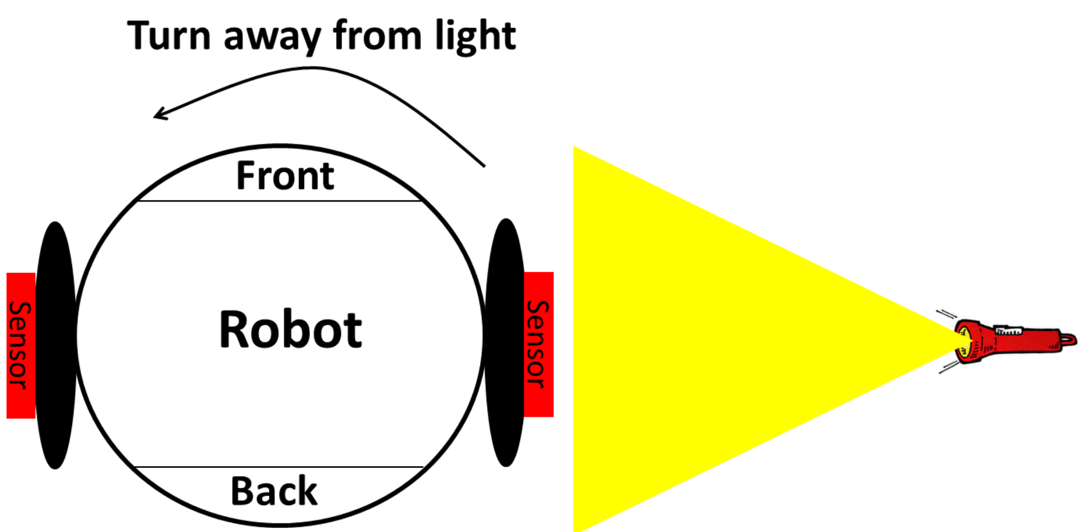
Robot behavior: The Robot should move away from the light in the following way. Where "Speed of motor" is the fraction of the motor's maximum speed.

- Speed of left motor = Intensity of left sensor / (Intensity of left sensor + Intensity of right sensor)
- Speed of right motor = Intensity of right sensor / (Intensity of left sensor + Intensity of right sensor)

Note: You are not allowed to use any MACROs in your code, except for register names.

- Example: You will lose points for: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | PIN1

- Must use: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | 0b0000_0010; // or 0x02



Name: Sean Gordon **Lab Section:** F

a. **Initialize TIMER 1 module A and B to meet the above requirements and so that both wheels initially move at 50% their maximum speed. (5 pts)**

```
void init_TIMER1_A_B()
{
     //----------------Configure GPIO Settings----------------\\

     SYSCTL_RCGCGPIO_R |= 0x2;          //Enable GPIO port B
  clock

     GPIO_PORTB_AFSEL_R |= 0x30;   //Set bits 4:5 to AF
     GPIO_PORTB_PCTL_R |= 0x0077_0000; //Set bits 4:5 to timer

     GPIO_PORTB_DEN_R |= 0x30;     //Set pin 4 digital
     GPIO_PORTB_DIR_R &= 0xCF;     //Set pin 4 input



     //----------------Configure Timer
       Settings----------------\\

     SYSCTL_RCGCTIMER_R |= 0x2;    //Turn on clock for port B

     TIMER1_CTL_R &= ~0x101;          //Disable Timer1 A and B

     TIMER1_TAMR_R &= 0xFFFF_F000;    //Clear non res bits
     TIMER1_TAMR_R |= 0x80A;
     TIMER1_TBMR_R &= 0xFFFF_F000;
     TIMER1_TBMR_R |= 0x80A;
     //11- Drive high on time-out, 3- PWM en, 1:0- Periodic timer

     TIMER1_CFG_R = 0x4;              //Set to 16-bit

     //16MHz clock, 16,000 ticks per millisecond
     unsigned int period = 16000;     //1ms period
     unsigned int match = period/2;   //50% duty cycle

     //Set timer values
     TIMER1_TAILR_R = period & 0xFFFF;//Lower 16 bits
     TIMER1_TAPR_R = period >> 16;    //upper 8 bits of period
     TIMER1_TBILR_R = period & 0xFFFF;
     TIMER1_TBPR_R = period >> 16;
```

Name: Sean Gordon **Lab Section:** F

```
    //Set match values
    TIMER1_TAMATCHR_R = match & 0xFFFF;    //Upper 16 bits
    TIMER1_TAPMR_R = match >> 16;    //upper 8 bits of match value
    TIMER1_TBMATCHR_R = match & 0xFFFF;
    TIMER1_TBPMR_R = match >> 16;

    TIMER1_CTL_R |= 0x101;          //Enable Timer1 A and B
}
```

b. **Initialize the ADC to meet the specification above. <u>No interrupts are to be used</u> (5 pts)**

```
void init_ADC()
{
    SYSCTL_RCGCGPIO_R |= 0x10;    //Enable GPIO port B clock


    GPIO_PORTE_AFSEL_R |= 0b0001_0100; //Set bits 2,4 to AF
    GPIO_PORTE_AMSEL_R |= 0b0001_0100; //Enable pins 2&4 analog

    GPIO_PORTE_DEN_R &= 0b1110_0111;   //Disable pin 2,4
      digital
    GPIO_PORTE_DIR_R &= 0b1110_0111;   //Set pins 2,4 input



    SYSCTL_RCGCADC_R |= 0x1;           //Enable ADC module 0
      clock

    ADC0_ACTSS_R &= ~0b1;         //Disable ADC SS0 Sampler

    GPIO_PORTE_ADCCTL_R &= 0b1110_0111;//Pins 2,4 !trigger ADC
    ADC0_SSMUX0_R |= 0x0091;           //Sample ADC channels
      1,9
    ADC0_SSCTL0_R |= 0x60;

    ADC0_ACTSS_R |= 0b1;               //Enable ADC SS0 Sampler
}
```

Name: Sean Gordon **Lab Section:** F

c. **Complete the following API function to read in the light sensor values. Use polling (i.e. no Interrupt Service Routines). (5 pts)**

```
void get_sensor_reading(int *left_sensor, int *right_sensor)
{
     ADC0_PSSI_R = 0b1              //Start conversion with SS0

     while((ADC0_RIS_R & ADC_RIS_INR0) == 0){

     }

     ADC0_ISC_R = 0b1;              //Clear SS0 interrupt

     &left_sensor = ADC0_SSFIFO0_R;
     &right_sensor = ADC0_SSFIFO0_R;
}
```

d. **Complete the following API function to set the speed of each motor. The inputs should be specified on a 100-point scale (e.g. 50 means 50% speed). Assume the input parameters are no less than 1 and no greater than 99. Also rounding errors are acceptable (i.e. do NOT use floating-point calculations) (5 pts)**

```
void set_motor_speed(int left_motor, int right_motor)
{
     TIMER1_TAMATCHR_R = 16000 - (16000 * left_motor)/100;
     TIMER1_TBMATCHR_R = 16000 - (16000 * right_right)/100;
}
```

Name: Sean Gordon **Lab Section:** F

e. **Complete `main()` (5pts)**

```
// Don't go into the light program
main()
{
  int left_sensor;
  int right_sensor;
  int left_motor;
  int right_motor;

  init_TIMER1_A_B();
  init_ADC();

  while(1)
  {
     get_sensor_reading(&left_sensor, &right_sensor);

     // Computed motor speed commands
     left_motor =
          (left_sensor * 100)/(left_sensor + right_sensor);
     right motor =
          (right_sensor * 100)/(left_sensor + right_sensor);


    set_motor_speed(left_motor, right_motor);
  }
}
```

Name: Sean Gordon **Lab Section:** F

Question 3: Square Waves (15 pts)

a) For Timer 1 module B using PWM mode, generate a symmetric square wave (i.e. 50% duty cycle) with a **<u>10 ms</u>** period. Assume the associated GPIO module has already been configured (5pts)

<u>Note: You are not allowed to use any MACROs in your code, except for register names.</u>

- Example: You will lose points for: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | PIN1

- Must use: GPIO_PORTA_DEN_R = GPIO_PORTA_DEN_R | 0b0000_0010; // or 0x02

```
void init_TIMER1()
{
	SYSCTL_RCGCTIMER_R |= 0x2;		//Enable clock for port B

	TIMER1_CTL_R &= ~0x100;			//Disable TimerB

	TIMER1_TBMR_R &= 0xFFFF_F000;	//Clear non res bits
	TIMER1_TBMR_R |= 0x80A;
	//11- Drive high on time-out, 3- PWM en, 1:0- Periodic timer

	TIMER1_CFG_R = 0x4;				//Set to 16-bit

	//16MHz clock, 16,000 ticks per millisecond
	unsigned int period = 160000;	//10ms period
	unsigned int match = period/2;	//50% duty cycle

	//Set timer values
	TIMER1_TBILR_R = period & 0xFFFF;//Lower 16 bits
	TIMER1_TBPR_R = period >> 16;	//upper 8 bits of period

	//Set match values
	TIMER1_TBMATCHR_R = match & 0xFFFF;	//Upper 16 bits
	TIMER1_TBPMR_R = match >> 16;	//upper 8 bits of match value

	TIMER1_CTL_R |= 0x100;			//Enable TimerB
}
```

Name: Sean Gordon **Lab Section:** F

b) Now assume there is no PWM mode and that you have to use a Generic Waveform Generation approach (i.e. using an Interrupt Service Routine) to generate a symmetric square wave. Also assume the time to setup and execute the code in your ISR takes 20 µs (i.e. the CPU overhead involved with the ISR). What CPU utilize (i.e. percent of the CPU time) would be spent handing interrupts for: (5 pts)

i) Generating a square wave with a 10ms period

As there are two points the wave would change state, the ISR would be called twice per 10ms

2*20us = 40us $(40 * 10^{-6} / 10 * 10^{-3})$ = .004 => .4% of CPU time

ii) Generating a square wave with a 100 µs period

2*20us = 40us $(40 * 10^{-6} / 100 * 10^{-6})$ = .4 => 40% of CPU time

iii) Generating a square wave with a 50 µs period.

2*20us = 40us $(40 * 10^{-6} / 50 * 10^{-6})$ = .8 => 80% of CPU time

c) What is the key trade-off between using a Generic Waveform Generation approach (i.e. using an Interrupt Service Routine) vs. a Timer in PWM Mode for generating a symmetric square wave? (5 pts)

When switching from ISR to a PWM timer, you lose a small amount of customization for a much smaller amount of operhead.

Name: Sean Gordon **Lab Section:** F

Collaboration Documentation

List the people (First and Last name) you collaborated with: __________________________.

For each collaborator, describe the manner in which you collaborated:

1)

2)