LAB 05
Pulse Width Modulation

Goals:

- Use Port and Timer Interrupts with PWM (Pulse Width Modulation) to sweep the single-bladed arm of the SG90 back and forth between 0 and 180 degrees.
- Use the on-board button to start and stop the sweep.
- Use the on-board LEDs to indicate the state of the motor; red for stop, green for start.

Bonus:

Use a Port Pin other than P1.2 to output the PWM and use a different output and count mode. Justify your reasoning to the TA and in your report. +10 Use only one timer. +10

Pre Lab Questions:

- What values of TA0CCR1 did you use to move the arm to -0°, 90° and 180°?
- What duty cycle are these? Give your answer in time in ms and percentage.
- What happens with duty cycles below or above those mentioned in the specification sheet?
A secondary use for the Timer peripheral is the generation of a continuous period signal, sometimes referred to as a pulse train or pulse wave; this is known as PWM (Pulse Width Modulation). We can generate various periodic signals using the Timer peripheral of the MSP430.

**Timer**

The registers that control the Timer Interrupts are 16 bit. There are two timer modules; Timer A0 and Timer A1. Below are the pertinent registers:

- **TAxCTL** - Timer A Control; \( x = 0 \) or 1 (Timer A0 or Timer A1)
- **TAxCCTLy** - Timer A Capture/Compare Control
- **TAxCC Ry** - Timer A Capture/Compare Module Register
- **TAxR** - Timer A Count Register
- **TAIV** - Timer A Interrupt Vector Value

Where \( x \) is the Timer A Module number, 0 or 1 (Timer A0 or Timer A1) and \( y \) is the Timer A Capture/Control Module number: 0, 1 or 2. For the MSP430G2553, we have 2 Timer A Modules with 3 Capture/Compare modules for a total of 8 interrupt trigger sources: 2 Timer Overflows and 6 Capture/Compare flags (there is no Capture function for Module 2 for Timer A0).

Shorthand to refer to a Timer module and its capture/compare submodule exists. For example, TA0.1 refers to Timer A0 – Capture/Control Module 1. In addition, the MSP430G2553 header file has Control Bits defined for the Timer Control Registers. See the Interrupts Guide for more details.
**SG90 Servo Motor**

The SG90 is a simple servo motor with a 180° position capability. You can use 3 base positions to determine your motors exact range: far left, middle and far right at -90°, 0° and 90° respectively. See the diagram below:

The datasheet is short and contains all the info you need. Connect the power wire to any VCC pin on the MSP430, same with the Ground pin, to any GND pin that is. The PWM wire should be connected to any Port Pin that is PWM capable, i.e. pins that are enabled using TAxCCR1 (See the Interrupts Guide for more details). You will gradually increase or decrease the duty cycle of the pulse width by manipulating this value.
Notes:

- Every motor is slightly different. Also, these motors run best under 5V, but the MSP430 puts out about 3.3V. This just affects the pulse width’s period, meaning that the duty cycle necessary for a given position will vary slightly than what is mentioned in the data sheet. A period of 20ms is still recommended.
- Depending of the position of the motor, you might cause a momentary short, making it a bit inconvenient to debug as it will severe communication to the PC. This is not cause for concern.
- You will notice that your motor may have a range slightly more (or less than, depending on how you look at it) than the ± 90° degrees. Once you reach a point where the motor will no longer move, you may stop increasing the pulse width duty cycle.
- The goal is to sweep back and forth through the whole 180° range of the motor, being able to start and stop with the push of the button. Depending on how you increment the pulse width duty cycles and set up the timing, you will either get a slow choppy motion or a swift and smooth swipe. Either is fine.
Pulse Width Modulation

Lab Hints

It is recommended that you examine the sample program provided. Read the comments and study the program flow. Note the use of TA0CCR1 for the Period and TA0CCR1 for the Duty Cycle.

Since we are using interrupts, we will have an empty main loop. Setup the onboard button like you normally would. You will use the Port 1 ISR to stop and start the clock/PWM as well as toggle the LEDs. It’s possible to accomplish this lab with only one Timer ISR, but you may use two. See the Interrupts Guide for help on doing so. If you decide, you may choose to use global variables in your C program. The variables must be global or you will not be able to access them from any of the ISRs. Below is the basic setup. You may use it as the basis for your code, just fill in the blanks:
**Main Program:**

```
#include <msp430g2553.h>

unsigned int go = 0, step = 0, up = 1, tic = 0; //Example of Global variables.

int main (void)
{
    //Setup
    WDTCTL = WDTPW + WDTHOLD; //Stop Watchdog Timer

    //Setup P1.3
    P1DIR &= ~BIT3;
P1REN |=  BIT3;
P1OUT |=  BIT3;

    //Setup LEDs
    P1DIR |=  BIT0 + BIT6;
P1OUT |=  BIT0;
P1OUT &= ~BIT6;

    //Setup PWM - Timer A0.1 - P1.2
    P1DIR    |=  //??
P1SEL    |=  //??
P1SEL2   &= ~//??
    TA0CTL   =  //Setup Timer
    TA0CTL   ^=  //Stop Timer Here (This and previous line can be combined).
    TA0CCCTL1 =  //??
    TA0CCR0  =  //Period of 20ms (50Hz)
    TA0CCR1  =  //Duty Cycle of 1ms (5%)

    //Setup Timer A1.0
    TA1CTL   =  //Setup Timer
    TA1CTL   ^=  //Start Timer Here (This and previous line can be combined).
    TA1CCCTL0 =  //Enable Timer Capture Compare Interrupt
    TA1CCR0  =  //50000 clock cycles

    //Setup Port 1 Interrupts
    P1IE    |=  BIT3;
P1IES    |=  BIT3;
P1IFG    = 0;
    _enable_interrupts(); //Enable General Interrupts. Best to do this last.

    while (1){} //Empty infinite while loop.
}
```
Interrupt Service Routine(s):

// Port 1 Interrupt Service Routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    // *Toggle Start/Stop of the Timer for the PWM. Stopping the Timer
    // stops the PWM and thus stops the motor.
    // *Stop Timer 1 or else the pulse width will not be the same as when you
    // stopped the PWM and thus neither will be the position of the motor.
    // *Toggle LEDs Green/Red to indicate Start or Stop.

    P1IFG &= ~BIT3; // Clear Button Flag
}

// Timer 1 A0 Interrupt Service Routine
#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer1_A0_ISR( void )
{
    /* Will jump in here when TA1R reaches value of TA1CCR0 stored during setup.
    Increment or toggle TA0CCR1 here to vary the position of the motor.
    MAKE SURE YOU TOGGLE *TA0CCR1* since it belongs to the PWM and DO NOT
    TOGGLE *TA1CCR1* since that belongs to *this* timer.
    */
}
Lab Notebook Questions:

- Answer the following questions in your lab notebook:
- What is the difference between a servo motor and a regular DC motor?
- What are the dis/advantages of each?
- What uses, besides a windshield wiper, can you think of for the SG90?
What’s inside a servo motor?

Inside the servo motor, we have a regular DC motor, a potentiometer, and a control circuit. The DC motor is connected to the control circuit through a series of gears and the potentiometer. As the DC motor rotates, the resistance increases and the motor decides where to stop depending on the pulse signal applied to the PWM wire.

Can I use the motor to push, pull or lower and raise something; I see ‘9g’ on the side of the SG90; does that mean it can only lift 9 grams?

Sure. Each motor has a Stall Torque rating; a weight and distance that the motor can safely stall at, resisting any motion. For example, the SG90 has a Stall Torque rating of 16.7oz/in or 1.2kg/cm at 4.8V, meaning that the SG90 can, for instance, hold an item that weighs 16.7oz suspended by a string about 1 inch from the pivot of the attached arm or center of the motor. However, since we are interfacing with a 3.3V micro controller, those numbers will be smaller. It would still be close to a pound though. The ‘9g’ on the side of the motor means that the motor itself weighs 9 grams!

My motor sometimes swings fast and other times it moves slow; how can I get it to move more consistently?

The speed of the motor is proportional to the current position and the desired position, meaning that the speed will be faster if the distance between the two positions is greater, and slower if the distance is smaller. This is known as Proportional Control and helps the motor be as efficient as possible, keeping the motor from doing more work than necessary. Unfortunately, this means precise speed control is not possible and so you must take this into account.