Hobby Servo Motor Control via PIC Pulse Width Modulation

Nathan Markey
April 4, 2003

Executive Summary:

Typical hobby servo motors are controlled by digital pulse width modulation. One way to produce a signal to control a motor is to program a PIC microprocessor. An advantageous feature of certain PICs is their ability to produce a pulse width modulation signal which would be desirable in controlling the typical hobby servo motor. This application note instructs you on how to use a Microchip PIC 16F874 to control a modified TS-53 servo motor.

Keywords:

- CCS Compiler
- Frequency
- ICD – In-Circuit-Debugger
- Modified Servo Motor
- MPLAB
- PIC – Programmable Integrated Circuit
- PWM – Pulse Width Modulation
- Oscillator
- Servo Motor

Introduction:
My team for ECE-480 was assigned to the Magnetic Minesweeper Robot project. Our main objective was to create an autonomously controlled robot that would navigate a four foot square playing field in search of small magnets. One of the requirements of the project was that we were to use simple modified servo motors to propel our robot, and control the robot using a PIC. In doing so, I had to learn how to learn how to control the motors using digital output signals from the PIC (known as pulse width modulation).

Objective:
The objective of this document is to give someone enough insight so that they may be able to use the pulse width modulation features of a PIC in order to control a similar motor.

Recommended Hardware:
The following is a list of hardware that was used to complete the project as noted in this document. Similar results may be obtained by using other parts, but due to specifications of any part, your results may vary.

- PIC 16F874 Microcontroller from Microchip
- Tower Hobbies TS-53 Servo S3K Standard U (Modified)
- ICD (In Circuit Debugger)
- Power Supply (Capable of +5V)
- Oscilloscope
- 1MHz Oscillator
- 10K ohm Resistor
- 0.2 micro-farad Capacitor
- Circuit Board and Wires

Recommended Software:
The following is a list of software that was used to complete the project as noted in this document. Other software and compilers can be used, but specific steps mentioned may change as well as the code listed.

- MPLAB
- CCS Compiler

Hobby Servo Basics:
Most servos have three leads corresponding to positive voltage, ground, and signal. The signal line is the one we are concerned with. Typical servos use a defacto standard pulse width modulation technique which references the position of the output shaft of the motor. See figure 1 for an example of how the circuitry inside the servo motor casing interprets the input square wave signal. The high part of the square wave represents a logic high of a digital circuit and the low part of the square wave represents the logic low. The typical minimum pulse length is 1ms with a 10-20ms delay.

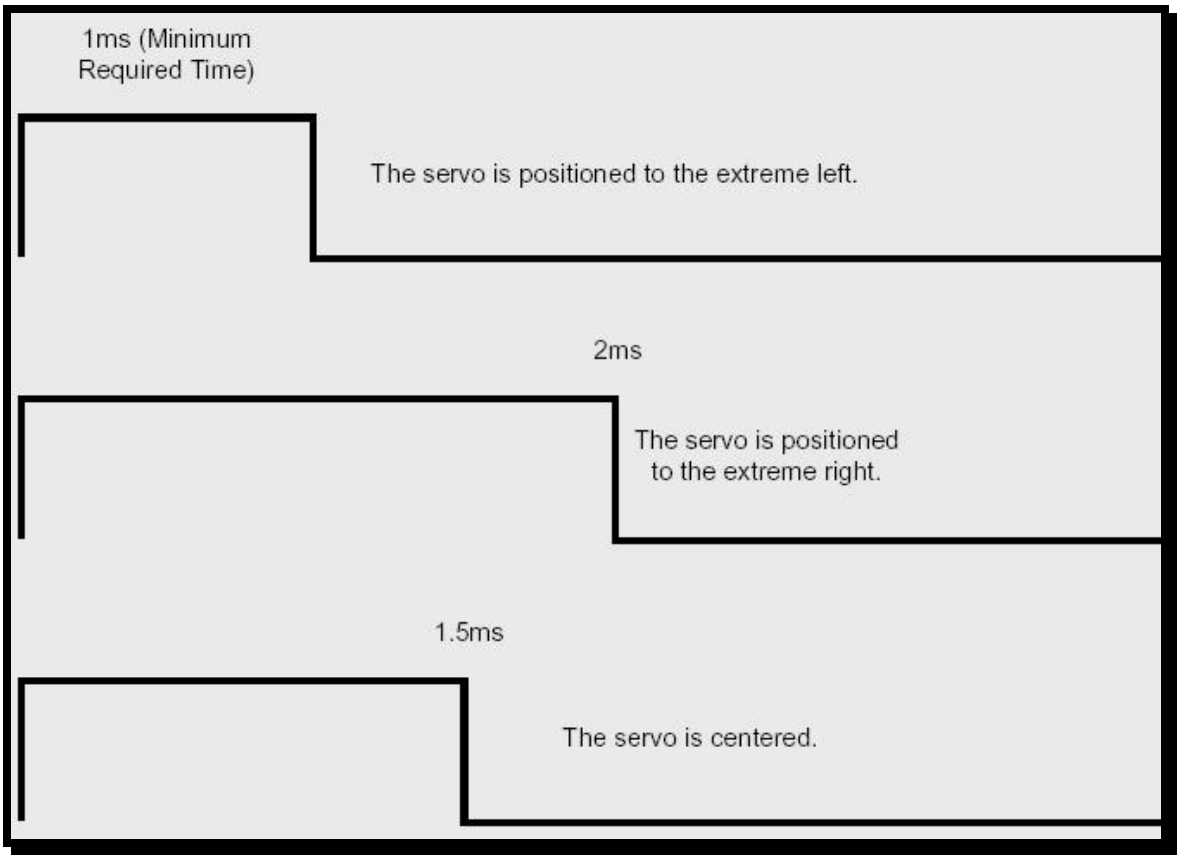

Figure 1

The information mentioned so far refers to an un-modified servo motor that only has just over a 180 degree range of motion. To have a motor rotate continuously, it needs to be "modified". To do this, you must remove the stop tab that is located on the main gear inside the motor. In addition, you must also remove the potentiometer as it provides feedback to the circuit inside the servo motor housing. The feedback from the potentiometer monitors the position of the motor. You must replace the potentiometer with two equal value resistors. The process to modifying a servo is somewhat more detailed, but you can easily find very detailed information on the web to help you do so. Our group was lucky enough to receive servo motors that had already been modified for use in our project. Once again looking at figure 1, you will notice that a pulse of about 1ms will cause the motor to rotate counterclockwise, a pulse of about 1.5ms will cause the motor to stop, and a pulse of about 2ms will cause the motor to rotate clockwise.

PIC Programming and Circuitry:
Microchip produces many programmable integrated circuits (or PICs) that may be used for various applications. Of the different types, one specific PIC that has the features needed for this application is the PIC 16F874 microprocessor. To program the PIC, I used a version of the CCS Compiler which is a compiler that creates a hex file for programming a PIC based using standard C code. To transmit the code to the PIC and to use in the debugging process, I used a version of MPLAB software along with a serially connected in-circuit-debugger (ICD). There are many different manufacturers for ICDs, but it is possible to construct your own as well. Figure 2 shows the circuit schematic for connecting the ICD to the PIC along with the output signal lines that will be used to control the motors. Use a 10 kilo-ohm resistor for the value of "R" and a 0.2 micro-farad capacitor for the value of "C". Figure 3 shows the pin-out for the PIC that corresponds to the connections in figure 2.

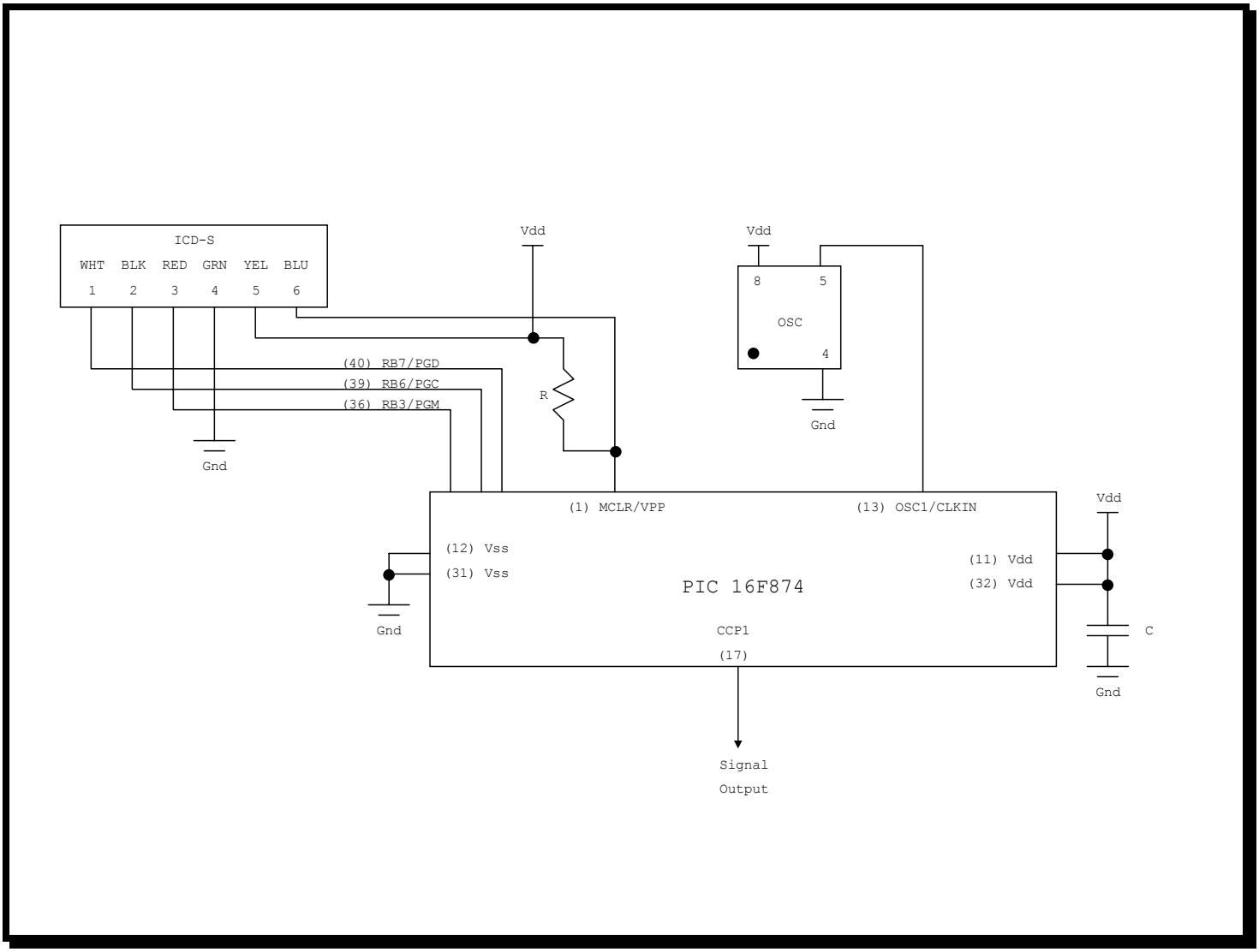

Figure 2

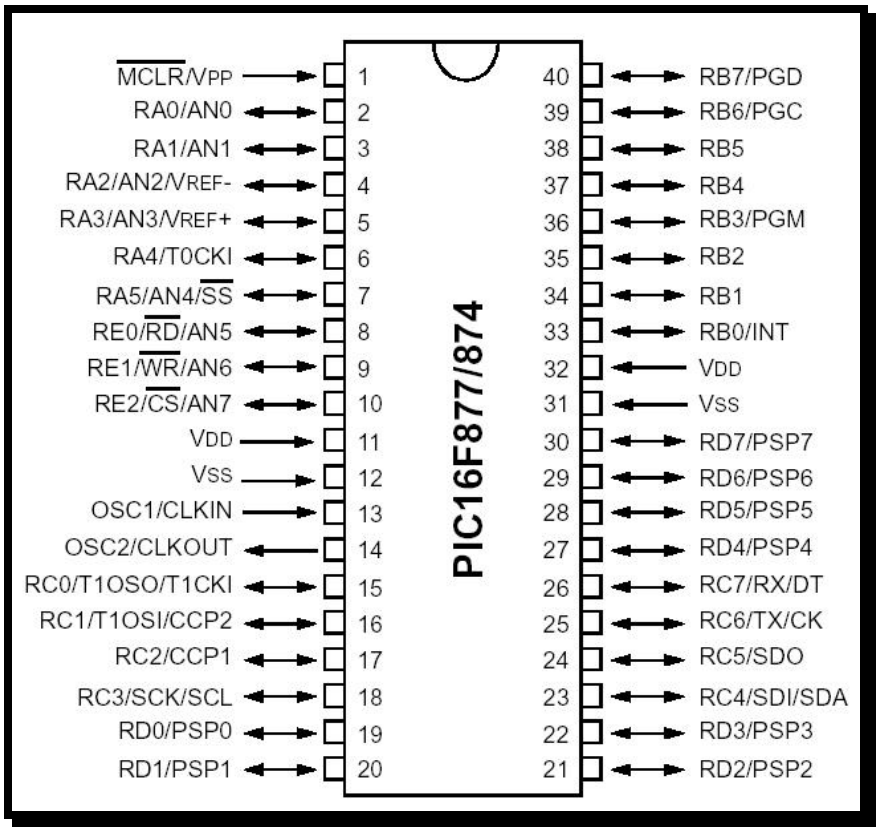

Figure 3

Pulse Width Modulation – PIC Programming:
To write the code for the PIC, you can use any standard text editor. The following is the code that uses pulse width modulation to rotate the motor. The code will rotate the motor clockwise, counterclockwise, and it will stop the motor.

```
#include <16F874.H>                  // For use with PIC
#fuses HS, NOWDT, NOPROTECT          // Set Fuses
#use delay(clock = 1000000)          // Synchronize with the Oscillator

main()
{
     long int pwm_value;
     setup_ccp1(CCP_PWM);            // Configure CCP1 as a PWM
     setup_timer_2(T2_DIV_BY_16, 200, 1);

     for (;;)  // Infinite Loop
     {
          // Rotate Clockwise
          pwm_value=125;                  // 20% Duty Cycle
          set_pwm1_duty(pwm_value);       // Set the PWM
          delay_ms(3000);                 // Delay 3 seconds

          // Stop
          pwm_value=94;                   // 15% Duty Cycle
```

```
            set_pwm1_duty(pwm_value);     // Set the PWM
            delay_ms(3000);               // Delay 3 seconds

            //Rotate Counterclockwise
            pwm_value=63;                 // 10% Duty Cycle
            set_pwm1_duty(pwm_value);     // Set the PWM
            delay_ms(3000);               // Delay 3 seconds

            // Stop
            pwm_value=94;                 // 15% Duty Cycle
            set_pwm1_duty(pwm_value);     // Set the PWM
            delay_ms(3000);               // Delay 3 seconds

      } // END FOR LOOP
} // END MAIN
```

The first line of the code references the library specific to your PIC. The next line is where you define your fuses. HS refers to the use of a high speed oscillator. NOWDT disables the watch dog timer of the PIC, and the NOPROTECT refers to "no program protect" allowing the PIC to be programmed by the ICD. The "#use delay" line is where you specify the frequency of your oscillator in order to synchronize the PWM and any calls to the Delay function. In this case the oscillator frequency is 1MHz or one million hertz. Inside the main function you must declare a variable to hold the calculated PWM value. The next line is where you configure the CCP1 pin on the PIC to be used for PWM output. Next you have to setup the timer to work your desired frequency base on the oscillator you are using. The equation for the cycle time is as follows:

(1/clock) * 4 * t2div * (period +1)

clock = 1000000
t2div can have three possible values (1, 4, or 16)
period must be an 8-bit value (due to the fact that the PIC uses an 8-bit divisor) 0-255

You must determine the frequency that you would like to run the PWM at. In the case of our servo, an optimal frequency is 100Hz. To solve for the period, we must set the equation above equal to 100Hz (or rather 10ms).

10ms = (1/1000000Hz) * 4 * t2div * (period + 1)
10ms = (1 us) * 4 * t2div * (period + 1)
10ms = 4us * t2div * (period + 1)
2500 = t2div * (period + 1)

…at this point we realize to get "period" to equal an 8-bit value, t2div must be 16 (since 2500/4 = 625 which is too big).

2500 = 16 * (period + 1)
period = (2500/16) -1 = 155.25

Since we have to use an integer value, we use 155 as the "period". The next part of the code changes the duty cycle of the PWM in order to rotate the motor (as shown in figure 1). It was determined that a 20% duty cycle would cause the motor to rotate clockwise, a 15% duty cycle would cause the motor to stop, and a 10% duty

cycle would cause the motor to spin counterclockwise. To calculate the pwm_value I needed to know the then width of the pulse (W), so I divided the duty cycle by the frequency. To finish the calculation I used the following equation.

pwm_value = (W * clock) / t2div

This resulted in the following:

- 20% duty cycle: pwm_value = 125
- 15% duty cycle: pwm_value = 94
- 10% duty cycle: pwm_value = 63

Results:

The next step (after compiling the program and programming the PIC via the ICD) was to connect the motor to the signal line and test it out. The results were as expected as the motors turned and stopped properly. Also, I connected the signal line to an oscilloscope to verify the output signal. Figure 4 shows the output for the 20% duty cycle, figure 5 shows 15% duty cycle, and figure 6 shows the 10% duty cycle. Notice how the width of the pulses decreases with the change in the duty cycle.

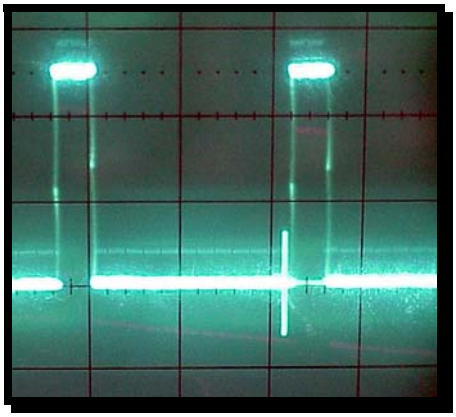

Figure 4

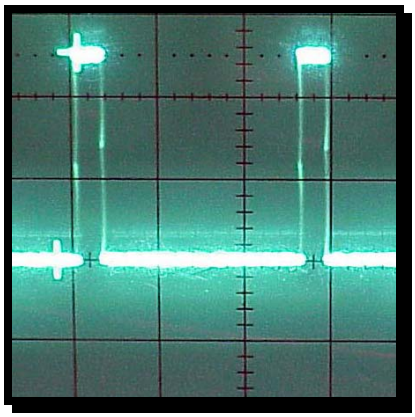

Figure 5

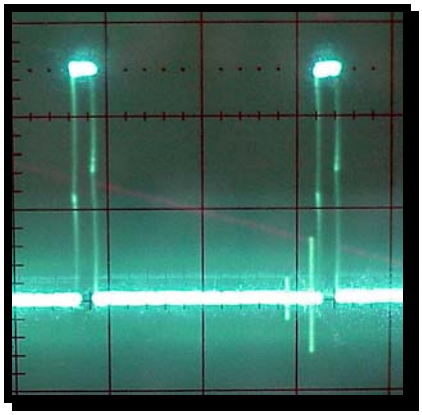

Figure 6

References:

PIC 16F974 Data Sheet
http://www.microchip.com/1010/pline/picmicro/category/embctrl/14kbytes/devices/16f874/610/index.htm
30292c.pdf

CCS Compiler Info
http://www.Ccsinfo.com

Hobby Servo Fundamental -- By: Darren Sawicz
http://www.winnipegrobotics.com/Papers/How%20To/servo.pdf

Tower Hobbies Servo Motor
http://www2.towerhobbies.com/cgi-bin/wti0001p?Q=1&I=LXUK84&P=3

CCS Compiler – PWM Basics – From: Dale Grove
(e-mail)