# Creating A PWM Signal Using A PIC 16F84

There are many small mechanisms, particularly servo motors, that use PWM coding as a means of input. PWM signals can also be used to vary the voltage applied to a device by achieving an effective average voltage. With so many applications, it is therefore necessary to have a reliable means of generating a PWM signal.

## MOTIVATION AND AUDIENCE

The focus of this tutorial is to demonstrate a method of generating a PWM signal using a PIC 16F84. This tutorial will teach you:

- ***What a PWM signal is.***
- ***How to write code to generate a PWM signal using a PIC 16F84.***

To do this, it is assumed that you already:

- ***Have completed "A Fast Track to PIC Programming".***

The rest of the tutorial is presented as follows:

- **Parts List and Sources**
- **Background**
- **Programming**
- **Applications**
- **Final Words**

## PARTS LIST AND SOURCES

In order to complete this tutorial you must have the circuit from the tutorial **"A Fast Track to PIC Programming"** (minus the dip switches and resistor LED circuits). This circuit will be the only part required for this tutorial. You will also need a DC power supply and access to an oscilloscope to observe the signal.

## BACKGROUND

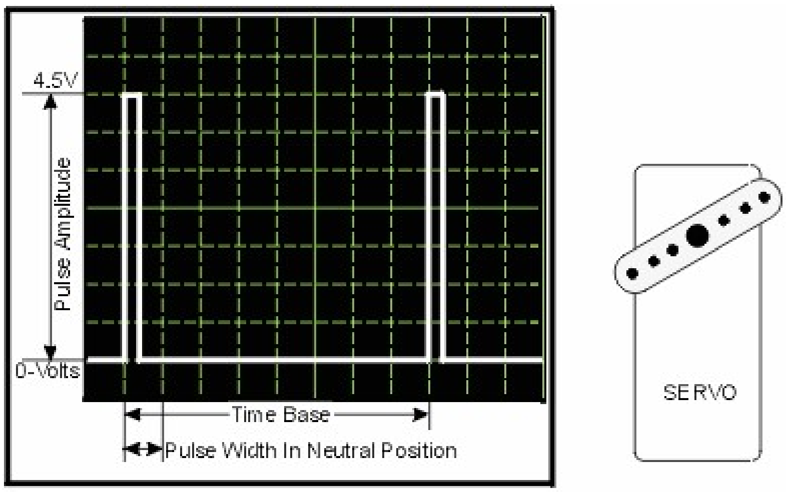


**Figure 1**

A PWM signal is simply a pulse of varying length, in effect a rectangular wave. This is illustrated in Figure 1, which also shows how a servo might react to different PWM inputs. For our circuit, the maximum voltage outputted will be +5 VDC, and the minimum will be 0 VDC. The length of the pulse generated is some times charcterized by a duty cycle. The duty cycle is the percentage of the signal that the output remains high. For instance, a constant +5V would be equivalent to a 100% duty cycle. A typical square wave output from a function generator has a 50% duty cycle. 0V would correspond to a 0% duty cycle.

## PROGRAMMING

**PWM.asm**

```
; FILE: PWM.asm
; AUTH: Keith Sevcik
; DATE: 5/21/03
; DESC: This program generates a PWM waveform.
; NOTE: Tested on PIC16F84-04/P

;-----------------------------------------------------------------------
;       cpu equates (memory map)

        list    p=16f84
        radix   hex

;-----------------------------------------------------------------------

portb   equ     0x06            ; port b equate
duty    equ     0x0c            ; length of duty cycle
temp    equ     0x0d            ; length of duty cycle

;----------------------------------------------------------------------

c       equ     0               ; status bit to check after subtraction

;----------------------------------------------------------------------
```

```
        org     0x000

        movlw   0x00            ; load W with 0x00 make port B output
        tris    portb           ; copy W tristate to port B outputs
        movlw   0x00            ; fill w with zeroes
        movwf   portb           ; set port b outputs to low
rstrt   movlw   d'0'
        movwf   portb
        movlw   d'157'          ; Duty cycle length
        movwf   duty
b0loop  movf    duty,w
        movwf   temp
        bsf     portb,0
pwma    nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwma
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,0
pwmb    nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwmb
        goto    rstrt

;----------------------------------------------------------------------

        end

;----------------------------------------------------------------------
; at burn time, select:
;       memory uprotected
;       watchdog timer disabled
;       standard crystal (4 MHz)
;       power-up timer on
```

## HEADER AND EQUATES

The first portion of code is the header and register equates. For more information about the meaning of the header see the previous tutorial.

```
        list    p=16f84
        radix   hex

;----------------------------------------------------------------------

portb   equ     0x06            ; port b equate
duty    equ     0x0c            ; length of duty cycle
temp    equ     0x0d            ; length of duty cycle

;---------------------------------------------------------------------

c       equ     0               ; status bit to check after subtraction

;---------------------------------------------------------------------

        org     0x000
```

The only equate of signifficance here is PWM. This register will be used to store the length of the PWM signal to be generated.

## INSTRUCTIONS

The next portion of code contains the actual instructions that tell the PIC what to do.

```
start   movlw   0x00            ; load W with 0x00 make port B output
        tris    portb           ; copy W tristate to port B outputs
        movlw   0x00            ; fill w with zeroes
        movwf   portb           ; set port b outputs to low
```

These lines set up port B as outputs. All outputs are then set to low.

```
rstrt   movlw   d'0'
        movwf   portb
        movlw   d'157'          ; Duty cycle length
        movwf   duty
```

After setting up the ports, the main loop is begun. At the beginning of the main loop, all port b pins are set to low just incase they are high when they shouldn't be. The duty cycle is then set to 157 (a 50% duty cycle. 255 corresponds to 100% and 0 corresponds to 0%).

```
b0loop  movf    duty,w
        movwf   temp
        bsf     portb,0
pwma    nop
        nop
        nop
        nop
        nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwma
```

The next bit of code is the loop for the PWM signal generated at pin B0. The pwm1a loop generates the high portion of the PWM signal. The duty cycle is stored in temp and then the pin is set high. after a pause, temp is decremented and so long as it doesnt reach zero the pause is repeated and temp is decremented again. After temp reaches zero, the code continues.

```
        movlw   d'255'
        movwf   temp
        movf    duty,w
        subwf   temp,f
        bcf     portb,0
pwmb    nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        decfsz  temp
        goto    pwmb
        goto    rstrt
```

The next portion of code generates the low part of the PWM signal. The value 255 is stored in temp, and the duty cycle is subtracted from this. This gives the remaining length of signal to be generated. Temp is then decremented in the same manner as above, this time with B0 set to low. Once the entire PWM signal has been generated, the code repeats.

This code causes a PWM signal to be generated with a duty cycle proportional to the value set. The frequency of the signal can also be adjusted by varying the delay (the number of nop's used).

## APPLICATIONS

One common application of pwm signals is motor control. By varying the duty cycle of a pwm signal sent to a motor, you can vary the effective power of the signal and thereby slow the motor down or speed the motor up depending on how long of a pulse you send to the motor. The signal generated by the PIC can not be directly connected to the motor, however, because the PIC is unable to handle the power required by the motor. It is therefore necessary to use a transistor to regulate the flow of current to the motor. A transistor is like an electric switch. When you send a logic high (+5V) to the transistor, it allows current to flow. When a logic low (0V) is sent, it restricts the flow of current. For digital signals, this means that the signal can be reproduced exactly, except the new signal is scaled up to a much larger current. Figure 2 shows a schematic for controlling a motor using a TIP31 NPN transistor.

**Figure 2**

As the schematic shows, the output from the pick is wired to the base. The negative terminal of the motor is then connected to the base and the collector is connected to ground. When the PWM otuput from the PIC is sent to the transistor, it will flip the transistor on and off and subsequently generate the same PWM signal to the motor, allowing you to control the motor with a PWM signal.

## FINAL WORDS

After completing this tutorial you should be familiar with PWM signals and how to program a PIC 16F84 to generate them.

If you have questions about this tutorial you can email me at **Keithicus@drexel.edu**.