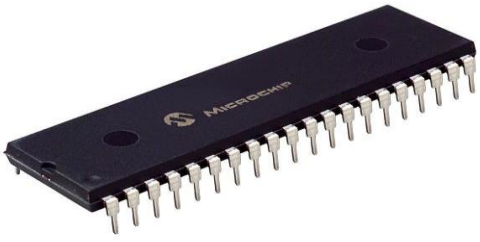
MICHIGAN STATE UNIVERSITY COLLEGE OF ENGINEERING

Embedded C on a PIC

Programming a Pulse Width Modulation on a PIC18F4520

Merav Nahoom

11/13/2009



Pulse Width Modulation (PWM) is the most effective mean to achieve constant battery voltage by controlling the amount of power to a load without having to dissipate any power in the load driver.

Programming a Pulse Width Modulation on PIC18F4520

Table of Contents

Programming a Pulse Width Modulation on PIC18F4520

Abstract

Some embedded systems run on battery power. For these systems, battery life is often a big issue. The primary method for preserving battery power is to turn off some parts or all of the system whenever possible. However when turning off all parts of the system is not possible, Pulse Width Modulation (PWM) plays a critical part in conserving energy.

Consider a light bulb that requires dimming. By switching the light bulb on and off very quickly, it is only on for half of the time. Hence, the average power drawn by the light bulb and the average power supplied by the battery will be half of the power if the light was not dimmed.

This on-off switching is called Pulse Width Modulation. The amount of power delivered to the load is proportional to the percentage of time that the load is switched on, which is called Duty Cycle.

Keywords

PIC, PIC18F4520, Pulse Width Modulation, PWM, Timer, Embedded C, Programming

Programming a Pulse Width Modulation on PIC18F4520

Objective

The purpose of this application note is to explain how to implement PWM with PIC18F4520. This document explains the different steps and methods taken in order to achieve the specification of the system.

Introduction

Pulse Width Modulation (PWM) is the preferred method to regulate motor speed because no additional heat is generated. In addition, it is energy efficient when compared to linear regulation. The PWM duty cycle is defined as Ton/Toff (%) in one period and the range is 0% - 100% (Figure 1). Controlling the Pulse Width Modulation of the system is an important aspect of this project. By controlling the Duty Cycle of the Peltier Junction, the heating or cooling rate of the water can be controlled. Moreover, implementing PWM will help in controlling and reducing the amount of current that is drawn from the battery, hence reducing power consumption.

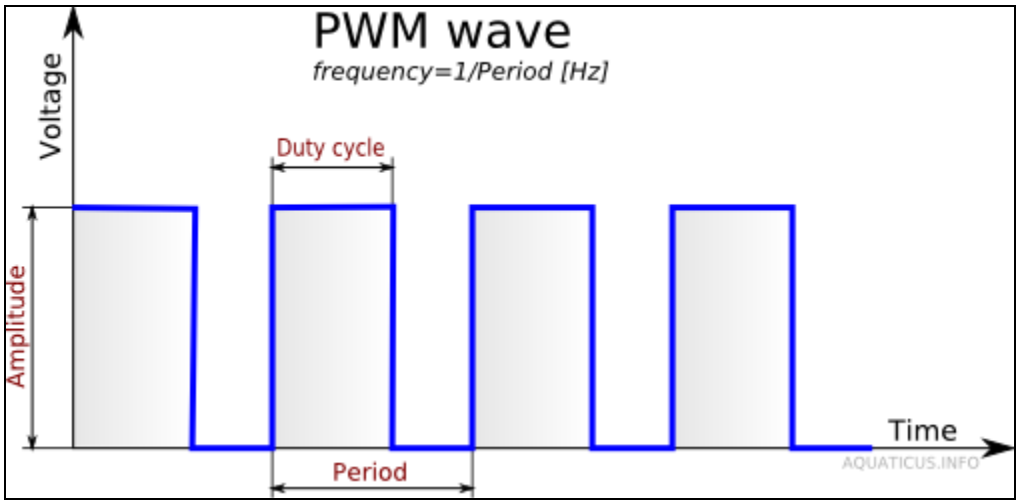


Figure 1– PWM Square Wave

PIC18F4520 has the capability of generating PWM signal with a period of approximately 400 µs. However, turning the Peltier Junction on and off this frequently will actually cause the system to draw more current from the battery. Therefore, PWM has to be generated in a way other than the internal capability of the PIC, utilizing the timer of the PIC.

PIC Programming

A. PIC PWM Functions

In order to use the PWM functions of the PIC when programming with MPLAB it is necessary to include two libraries in your code. The first library is *timers.h*. This library allows us to set the value of the TMR2 register which controls the frequency of the PWM generated.

The second library we must use is *pwm.h*. This library allows us to enable and control the PWM functions of the PIC. The commands used to control the PWM are OpenPWMx(Period) and SetDCPWMx(DutyCycle). The first command opens one of the two available PWM ports, CCP1 or CCP2 and sets the period of the PWM signal. The second command sets the duty cycle of the PWM. The value can be a 10-bit value with an equivalent decimal value of 1023 equal to 100% duty cycle. More details on how to set those values can be found on page 147 of the PIC data sheet, MPLAB C18 Libraries document, and the MPLAB C18 User's Guide.

As mentioned in the introduction, the period achieved by using the PIC's PWM is about 400 µs. The formula used to calculate it is as follows:

$$\text{PWM Period} = [(PR2) + 1] \bullet 4 \bullet TOSC \bullet (\text{TMR2 Prescale Value})$$

For a 40MHz crystal oscillator used in the system, TOSC is equal to 1/40M, PR2 is 255 and max TMR2 prescale is 16. Therefore, the maximum period will be

$$\text{PWM Period} = 256 \bullet 4 \bullet (1/40M) \bullet 16 = 409.6\ \mu s$$

For more details on how to calculate the period for PWM refer to the PIC data sheet on page 149.

A higher period was required for this system (1 minute period). Therefore, a different approach was taken in order to accommodate the necessary period.

B. Busy Waiting Approach

Busy Waiting (Figure 2) is a technique in which a process repeatedly checks to see if a condition is true. It can also be used to delay execution for some amount of time. This was necessary on old computers that had no method of waiting a specific length of time other than by repeating a loop a specific number of times.

```
for(time=0; time<pwmPeriod; time++)
{
	if (time < highDutyCycle)
	{
		for(loop=1; loop<50000; loop++);
			PORTDbits.RD3 = 1;
	}
	else if (time >= highDutyCycle)
	{
		for(loop=1; loop<50000; loop++);
			PORTDbits.RD3 = 0;
	}
}
```

Figure 2 – Creating PWM with Busy Waiting

Although this approach to generate delays for PWM is easy to understand and implement, it is not easy to produce precisely timed delays using this approach. However, Busy Waiting can be a valid strategy in certain special circumstances. In general it should be avoided because the CPU time spent waiting could have been reassigned to another task. In this case, the system was needed to utilize the CPU for processing instead of waiting.

C. *Final Approach*

Timers are a PIC peripheral function that can run in the background without interfering with the rest of the program. On the 18F4520 there are 4 timers that you can use, which run via hardware in the background. These timers are numbered Timer0, Timer1, Timer2, Timer3. You can setup the timer using the timer.h library functions. In this example code, I setup the timer 0, so I can use the internal clock:

```
// Setup the timer with a 1:256 prescaler with 16 bits resolution
OpenTimer0( TIMER_INT_OFF & T0_16BIT &
      T0_SOURCE_INT & T0_PS_1_256 );
```

```
#include <p18f4520.h>
#include <timers.h>

unsigned int tmr = 0;
unsigned int timeCnt = 0;

void DelaySec()
{
        while(ReadTimer0() < 39062)
        ;
        WriteTimer0(0);
        timeCnt++;
}

void main(void)
{
        ADCON1 = 0x0F;
        TRISD =0xF7;
        PORTDbits.RD3 = 0;
        OpenTimer0(TIMER_INT_OFF & T0_16BIT & T0_SOURCE_INT & T0_PS_1_256);
        while(1)
        {
                if(timeCnt > 30)
                {
                        timeCnt = 0;
                        PORTDbits.RD3 = !PORTDbits.RD3;
                }
                DelaySec();
        }
        CloseTimer0();
}
```

Figure 3 – PWM using internal clock

The frequency of the internal clock is based on the frequency of the external clock divided by 4. In this case the external clock is 40MHz; therefore the frequency of the internal clock would be 10MHz. In order to create a delay of one second, we need to consider the prescale value. Since the prescale is 256, we can divide the frequency of the internal clock by the prescale, which will come out to 39,062 . In the DelaySec() function in Figure 3 the clock gets incremented until it reaches the desired value (39,062). To create a period of 1 minute with a Duty Cycle of 50%, the function will count 30 seconds before it will change the state of the pin.

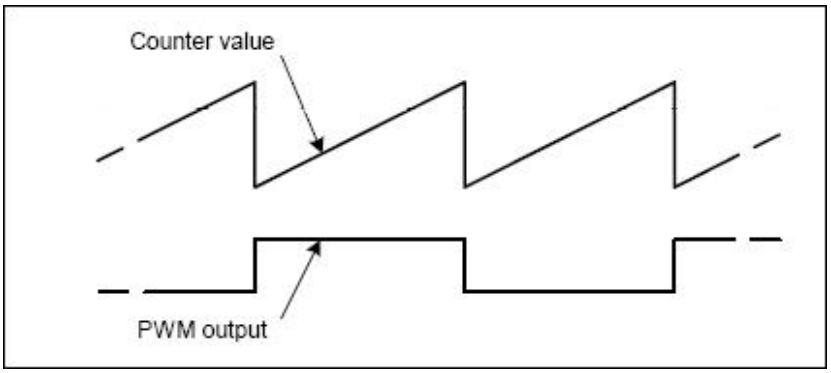


Figure 4 **- Counter Values and PWM Output**

Figure 4 shows how the counter works with the PWM that is implemented in Figure 3. The explanation for the calculation of the timing and the frequency of the internal clock can be obtained from the PIC data sheet.

Conclusion

Using PWM is a very important aspect of a system for power conservation and efficiency. For that reason there is a built in capability in the PIC to produce PWM. However, using the PIC's capability is not always sufficient to the specs of the system. Hence, a programmer needs to understand the requirements and the need for PWM in order to implement it. The advantages of implementing PWM, instead of using the PIC's function, include implementation on a wider range of pins as well as extending the range of the period.

References

PIC18F4520 Data Sheet

http://ww1.microchip.com/downloads/en/DeviceDoc/39631B.pdf

MPLAB C18 Libraries

http://ww1.microchip.com/downloads/en/devicedoc/MPLAB_C18_Libraries_51297f.pdf

MPLAB C18 User's Guide

http://ww1.microchip.com/downloads/en/devicedoc/MPLAB_C18_Users_Guide_51288j.pdf

Embedded C, Pont, Michael J. Print.

Aquaticus ROV, Homemade underwater robot

http://aquaticus.info/pic/pwm_wave.png