

Software PWM Generation for LED Dimming and RGB Color Applications

Author: Parthiv Pandya
Microchip Technology Inc.

INTRODUCTION

This application note describes a software solution that generates one or more pulse-width modulated (PWM) signals using a PIC[®] microcontroller (MCU). PWM control signals are widely used in embedded control applications for a variety of tasks that include light dimming, motor speed control, output voltage control, and communication between devices.

It is convenient and sometimes more practical to use a MCU that has dedicated PWM peripherals for these types of applications. However, a software PWM solution allows a less expensive MCU to be used if the frequency and resolution requirements for the application are not too high. Furthermore, the software solution could be used to provide additional PWM channels if the number of PWM peripherals on the MCU is not sufficient.

Pulse-width modulation is an effective way to control the brightness of LEDs. As a demonstration, the software PWM solution will be applied to control the color output of an RGB LED (OPTEK model OVSTRGBBCR8). By generating three software PWM signals red, green, blue or any mix of color, can be generated.

Recent advances in LED technology have allowed LEDs to be used as efficient sources of illumination. The RGB demonstration has a wide range of applications, including automotive interior lighting, architectural lighting, and LCD display backlighting.

A PIC12HV615 microcontroller device was chosen for the RGB demonstration. The PIC12HV615 is a very low-cost 8-pin MCU with an internal RC oscillator and a shunt voltage regulator. The device has many on-chip peripherals, including: a comparator, an ADC, a Capture/Compare/PWM (CCP) module, and three timers. The features, low cost, and small size of the PIC12HV615 allow a user to add the intelligence of a MCU to almost any application.

The first part of this application note will show you how to generate software PWM signals using the Timer0 peripheral, which is available on most PIC MCUs. The second part will show you how to use the PWM code for the RGB color demonstration.

HOW PULSE-WIDTH MODULATION WORKS

Pulse-width modulation is one of the most widely used output techniques available to the embedded designers. PWM can also convey analog information in a digital format. The amplitude of an analog signal is encoded in terms of the on-time and off-time ratio (duty cycle) of a PWM period.

A PWM signal has a fixed frequency. The width (W) of each pulse varies between 0 and the period (T). The duty cycle (D) of a signal is the ratio of pulse width to period. Figure 1 shows a PWM output waveform.

FIGURE 1: PWM OUTPUT

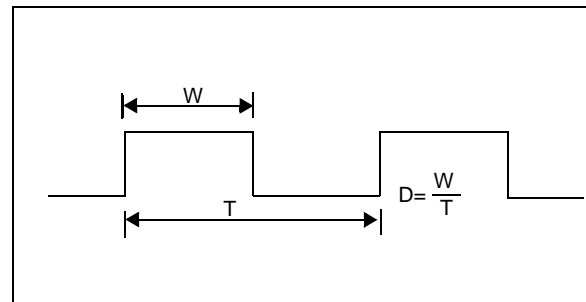


Figure 2 shows a PWM output at a 10% duty cycle (i.e., the signal is on for 10% of the period and off for the other 90%). Figures 3 and 4 show PWM output at 50% and 90% duty cycles, respectively.

AN1074

FIGURE 2: 10% DUTY CYCLE

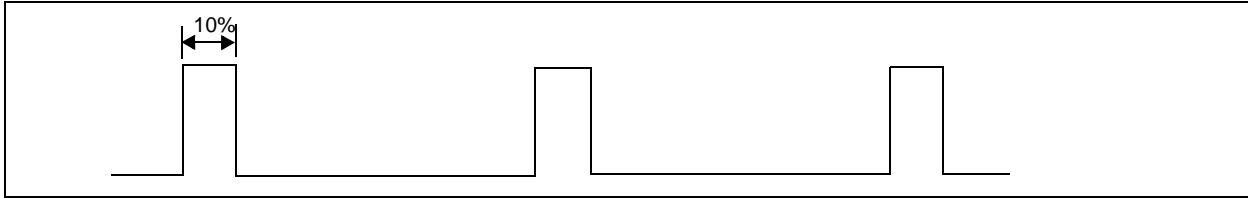


FIGURE 3: 50% DUTY CYCLE

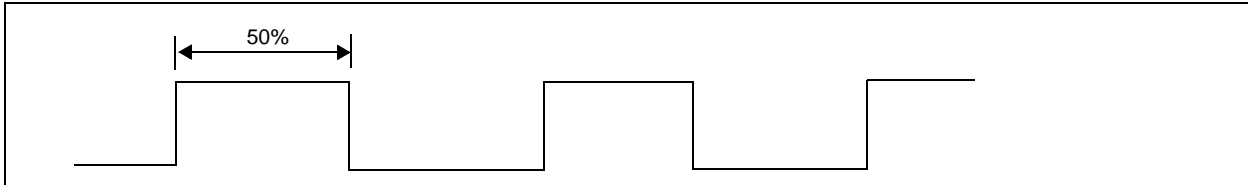
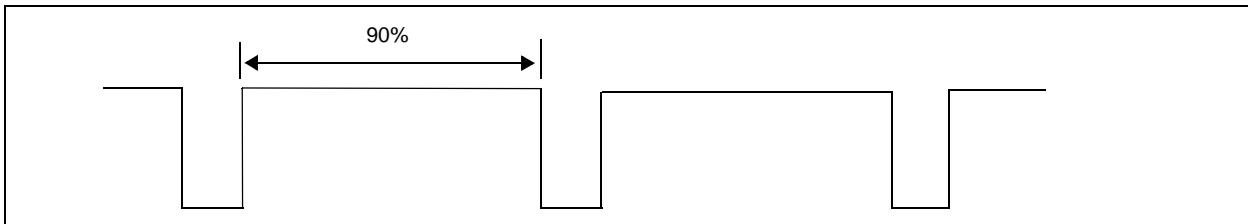


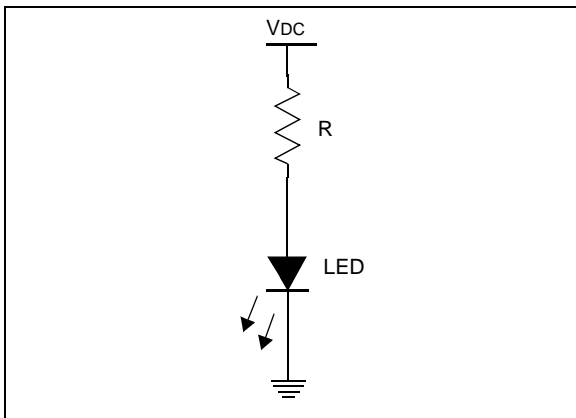
FIGURE 4: 90% DUTY CYCLE



USING PWM DIMMING TO CONTROL LED BRIGHTNESS

A LED is a current-controlled device. Figure 5 shows a simple LED application circuit.

FIGURE 5: CIRCUIT TO ILLUMINATE A LED



The light output from a LED is proportional to the current passing through it. There are two techniques to control the LED brightness in the circuit.

1. Vary the LED drive current. The LED drive current may be controlled using a variable resistor (R) or using a variable voltage power supply (V_{DC}).
2. Apply pulse-width modulation to the LED drive current.

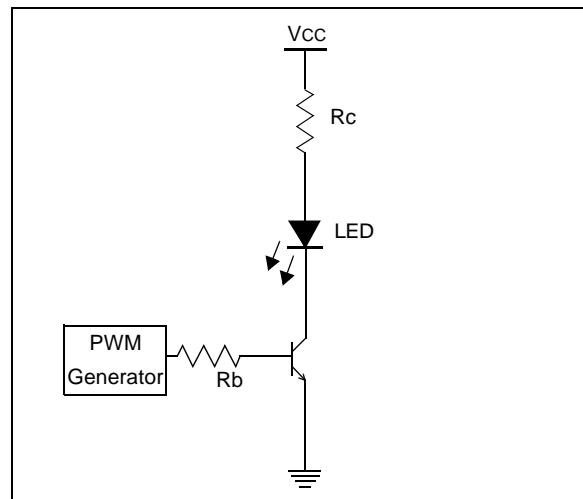
The first method has two major disadvantages:

1. As the current is reduced the LED efficiency may also be reduced.
2. In high power white LEDs, a color shift may take place with the reduction in current level.

The PWM dimming technique always drives the LED at full current. Therefore, problems such as reduced efficiency and color shifts can be eliminated.

Figure 6 shows a circuit to control the brightness with the PWM dimming technique.

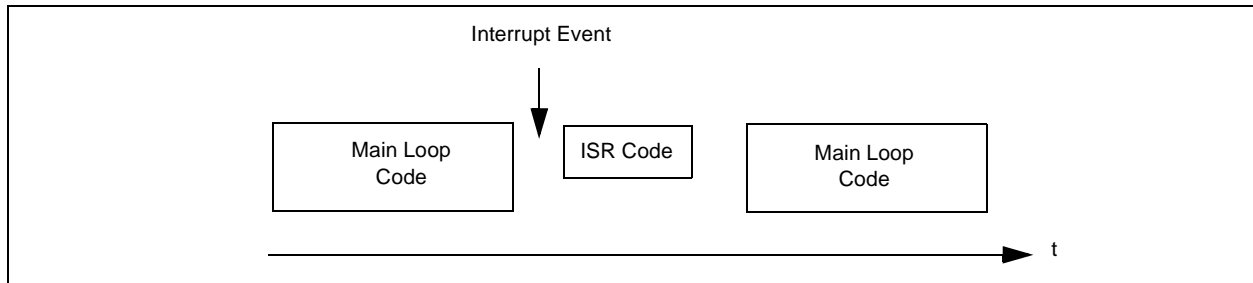
FIGURE 6: PWM DIMMING TECHNIQUE FOR BRIGHTNESS CONTROL



INTERRUPTS

Interrupts are defined as asynchronous events, generated by an external or internal hardware source. These events cause the CPU to interrupt the execution of the current program and to start a service routine, which is dedicated to these events. After the execution of this Interrupt Service Routine (ISR), the program that was interrupted will be resumed. Figure 7 shows the program flow when an interrupt is generated.

FIGURE 7: INTERRUPT OPERATION



GENERATING PWM WITH TIMER0

This software solution uses the Timer0 module to control the width of each PWM pulse.

To ensure that the PWM pulsing is not visible to the human eye, a 100 Hz PWM signal frequency was chosen. To allow for 32 discrete duty cycle values to be generated, the 10 mSec period was further divided in 32 intervals each 312 μ Sec long (10mSec/32 \sim 312 μ Sec). Therefore, the Timer0 module must be configured to generate an interrupt at 312 μ Sec intervals. If a greater duty cycle resolution is desired, a shorter Timer0 period must be used.

The Timer0 module counts from 0 (or a preloaded value) to 255 and rolls over. The interrupt occurs when the timer rolls over.

EQUATION 1:

$$\text{Interrupt Period} = \text{Prescale} \times \text{Tcy} \times (256 - \text{PreLoad})$$

Equation 1 indicates the relationship between the Timer0 pre-loaded value and the resulting interrupt period. Tcy is the microcontroller instruction cycle, obtained by dividing the microcontroller internal clock period by four. The microcontroller internal oscillator operates at 4 MHz frequency so Tcy is determined to be 1 μ Sec.

To obtain the desired 312 μ Sec interval and to ensure that the PreLoad is an 8-bit value (0-255) the Prescaler, an 8-bit clock divider that can be optionally assigned to the Timer0 module, is used to double the clock period. (Prescale = 2).

By solving Equation 1 for the PreLoad value, we obtain Equation 2.

EQUATION 2:

$$\begin{aligned} \text{PreLoad} &= 256 - (\text{InterruptPeriod} / \text{Prescale} \times \text{Tcy}) \\ \text{PreLoad} &= 256 - (312 \mu\text{Sec} / 2 \mu\text{Sec}) \\ \text{PreLoad} &= 100 \end{aligned}$$

Substituting the known values and rounding the result to the nearest integer value, we obtain the value to write into Timer0 register after each interrupt event.

SOFTWARE OPERATION

The source code file "SoftwarePWM.asm" supplied with this application note generates simultaneously three PWM signals of identical period (10 ms) but with independent duty cycles.

As described in the previous section, Timer0 generates an interrupt every 312 μ Sec. In the ISR, the state variable `IntCount` is decremented and tested for zero and for the desired duty cycle. If `IntCount` is zero then all output pins are set. This is the rising edge for all three PWM signals and the beginning of the PWM period. The `IntCount` value is then set to '32'.

If `IntCount` is not zero then it is compared to the three duty cycle values: `DutyCycle0`, `DutyCycle1` and `DutyCycle2`. If a match occurs with any one of the three duty cycles, the corresponding output pin is cleared. This decides the falling edge for the corresponding PWM signal.

AN1074

FIGURE 8: PWM SIGNAL

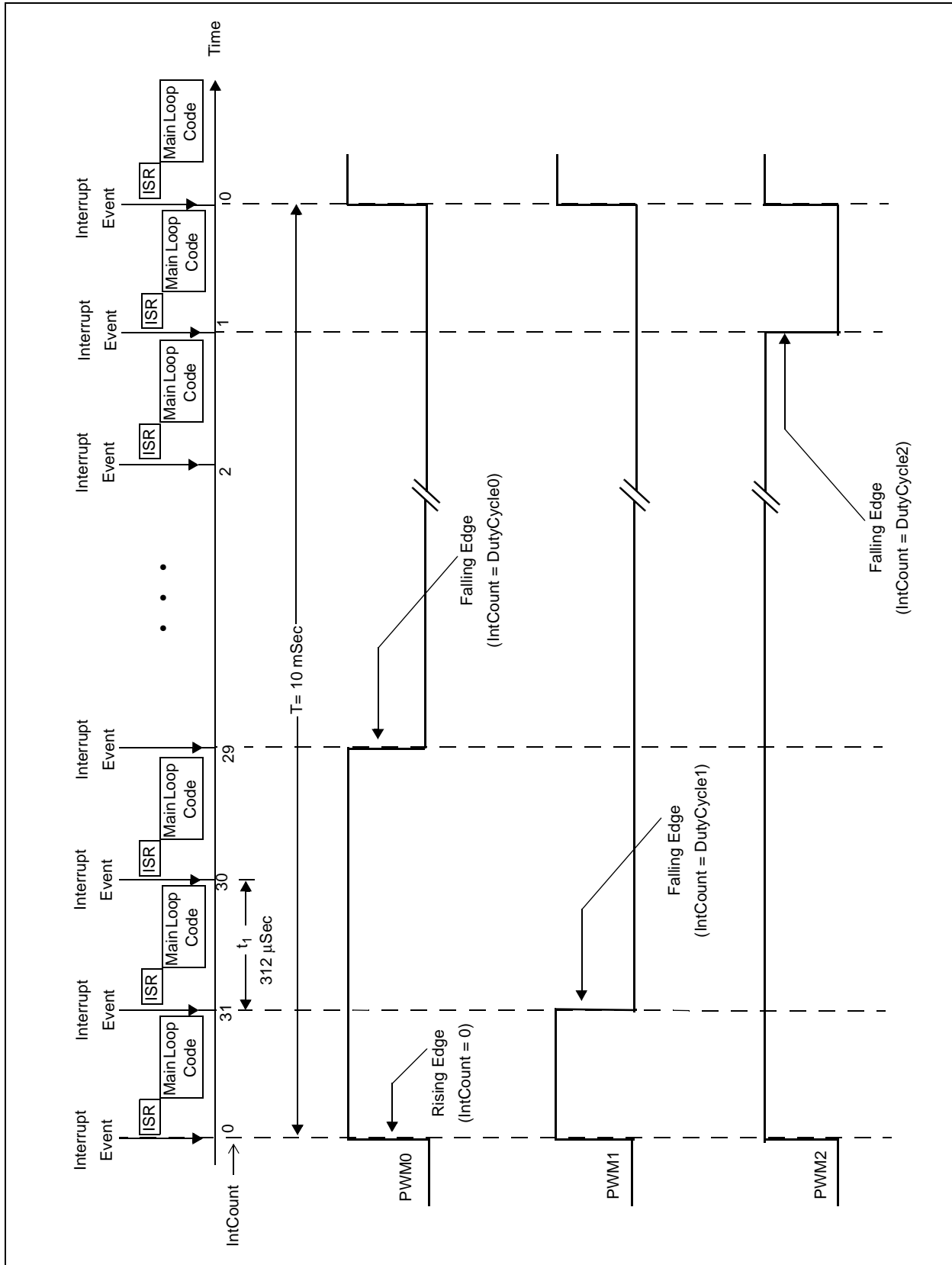


Figure 8 shows how the generated PWM period is divided into thirty intervals. Each interval corresponds to a Timer0 interrupt period of 312 μ Sec.

RESOURCES USED BY THE PWM MODULE

1. Program Memory:
The software uses 238 bytes of program memory out of 2048 bytes total. So, program memory usage is 11.62%.
2. Data Memory (RAM):
The software uses 11 bytes of RAM (Data Memory) out of 128 bytes total. So, RAM usage is 8.59%.
3. CPU Bandwidth:
The device operates at 4 MHz clock speed. The software takes 923 cycles per 10 mSec period for 100 Hz PWM frequency. Therefore, the CPU usage is less than 10%. The CPU usage will increase if the PWM frequency is increased.

USING AND CUSTOMIZING THE SOFTWARE PWM MODULE

To use the PWM module in your application, add three files "SoftwarePWM.asm", the device linker script and "SoftwarePWM.h" to the project directory. In the "SoftwarePWM.asm" source code file, three separate code sections (Channel0, Channel1 and Channel2) are responsible for the generation of the respective three outputs. You can add or remove these blocks according to the number of PWM signals required by your application. For example, if you need only one PWM signal, use only the Channel0 code block (and the corresponding DutyCycle0 variable) and change the I/O pins accordingly.

Comments are provided in the source code indicating which lines of code can be removed if you do not need the 2nd or 3rd PWM output. See the listing in the **Appendix D: "Source Code For Software PWM"**.

AN1074

MULTI-COLOR DEMO USING SOFTWARE PWM

In this section, we will use the software PWM code to create a demo that sweeps through different colors using a PIC12HV615 and a tri-color LED. The three PWM signal will be used to control the brightness of the Red, Green and Blue emitters.

MIXING COLORS

To create different color combinations we will have to change the duty cycles of the three PWM outputs over time. One way to do this is to control the three PWM duty cycles with a three phase sinusoidal profile. This will generate a rotating (color) vector that will sweep smoothly across the chromaticity plane generating a wide range of color combinations.

GENERATING A SINUSOIDAL WAVEFORM

The easiest way to generate a sinusoidal waveform is to use a look-up table that contains a selection of points in a sine wave cycle. The sine values are read from the table at periodic intervals, scaled to match the allowable range of duty cycles, and then written to the duty cycle variables (`Dutycycle0`, `Dutycycle1` and `Dutycycle2`).

A pointer variable is used to move through the table. This pointer has to be adjusted at periodic intervals, usually at the beginning of each PWM period. If a constant value is added to the pointer at each interval, a fixed frequency output sinusoid will be produced.

CHOOSING POINTS IN THE TABLE

One of the key issues in creating a look-up table is defining the number of points to be used. Too few points in the table will cause a visible 'stepping' effect in the light produced by the LED. Duty cycle steps can also be skipped during the steep portions of the sine wave profile, which further increase the stepping effects. On the other extreme, too many points will use up valuable memory in the MCU. As an approximation, you can multiply the number of duty cycle steps by 3 to get the minimum number of sine table points that will maximize the available duty cycle resolution. In this case, $3 \times 32 = 96$ steps are needed in the sine table.

The table length is usually set to a power of 2, such as 32, 64, 128 or 256. This way, the software does not have to check if the pointer has reached the end of the table, but a simple bit masking operation will produce an automatic roll over. The presented demo code uses a 128 value sine table and a 7-bit mask.

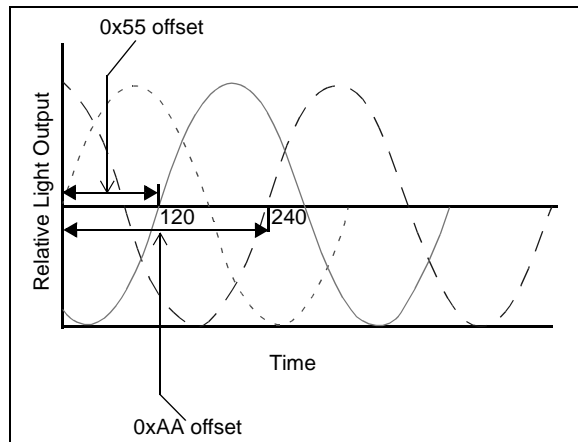
The resolution of the sine table is:

$$\text{Resolution} = 360^\circ/128 = 2.8^\circ/\text{bit}$$

THREE-PHASE OUTPUTS

Borrowing from motor control theory, the rotating color vector can be obtained by shifting by a constant 120 degrees the phase of three output sinusoids of the same frequency and amplitude. Binary numbering works well for three-phase systems. Assuming an 8-bit pointer size is used, values of 0x55 and 0xAA provide 120-degree and 240-degree offsets, respectively. The offset values are added to the sine Table Pointer at each PWM interrupt to provide two additional pointers for the 2nd and 3rd phase. The output three-phase sine wave with 120 degree phase shifts would look like Figure 9.

FIGURE 9: THREE PHASE SINE WAVE



CHANGING THE COLOR SWEEP SPEED

In order to make the demonstration more interactive we want the color sweep speed to be controlled by the user via a potentiometer. The PIC12HV615 has a built-in ADC module that can be used for the task. By scaling and adding the result of the ADC conversion to the sine Table Pointer at the beginning of each PWM cycle (every 10 mSec) we can control the frequency of the output sinusoidal brightness signals. The larger the ADC value, the faster the application sweeps through the color combinations.

To get a better idea of how the sine wave profile is generated by using a sine table, a section of the ISR code is shown below. This code will be executed every 10 mSec (i.e., once per every PWM period).

EXAMPLE 1: SINE WAVE GENERATION USING A SINE TABLE

```

ADCCConversion
BSF      ADCON0,GO      ;Start A2D conversion
BTFSC   ADCON0,GO      ;Wait until the conversion is completed
GOTO    $-1

RRF      ADRESH,W
ANDWF   0x7f
MOVWF   Temp
MOVLW   .25             ;ADRESH + 25
ADDWF   Temp,W
ADDWF   PhaseL,F       ;PhaseL+Temp
BTFSC   STATUS,C       ;Chk if the Carry is generated because of the addition of
                        ;PhaseL and Temp in the previous instruction

INCF    PhaseH,F

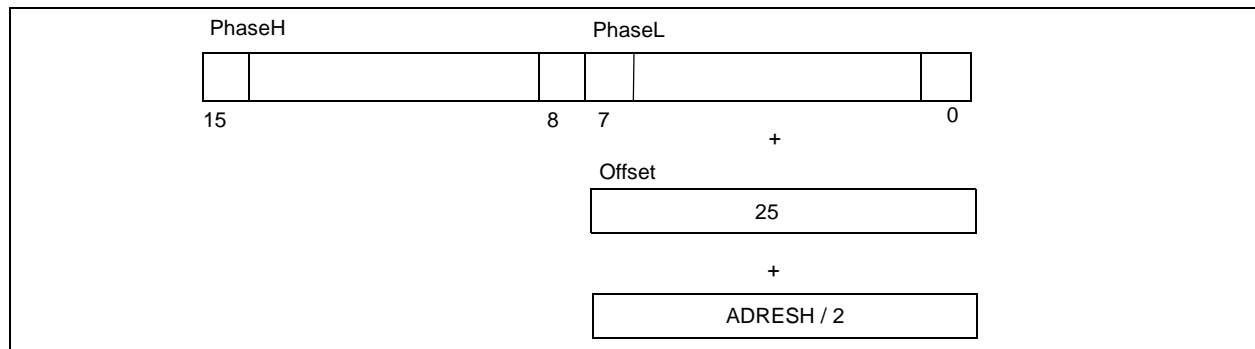
; ForPhase1
RRF      PhaseH,W       ;Sinetable has 128 values. Therefore, the pointer should be
                        ;7 bit wide

ANDLW   0x7f
CALL    SineTable
MOVWF   DutyCycle0
    
```

Let's see what the code does at a high level:

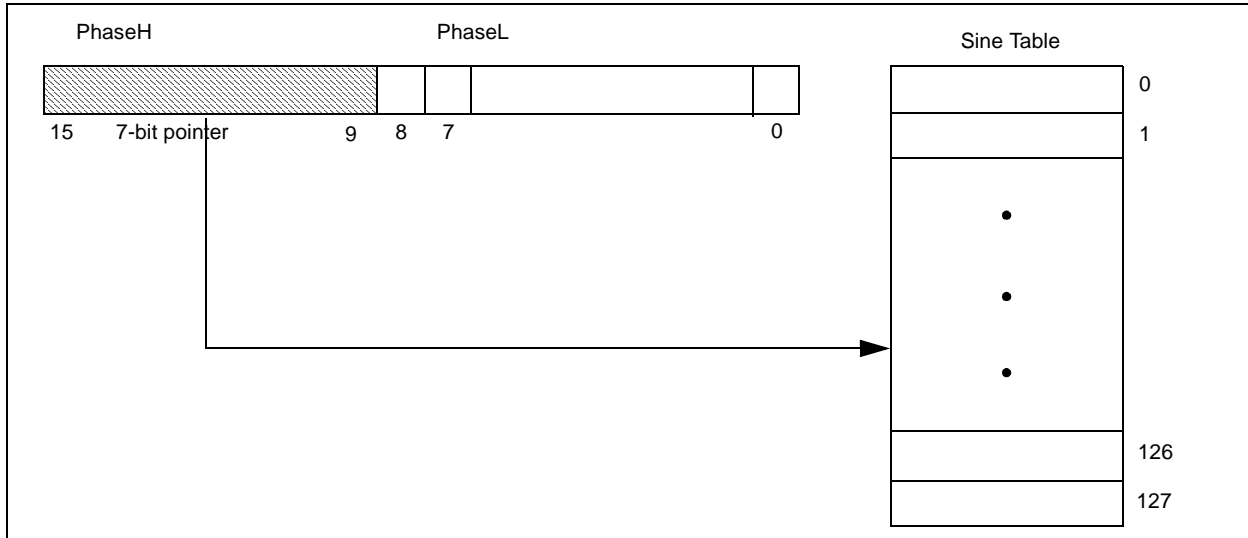
1. Start A/D conversion and wait for conversion to complete.
2. The upper 7 bits of the ADRESH (A/D conversion result) is retrieved.
3. An offset is added to ensure a minimum motion speed
4. The result is added to a 16-bit integer used to represent the sinusoid phase angle: Phase (PhaseH:PhaseL) where PhaseH is the MSB and PhaseL is the LSB.

FIGURE 10: 16-BIT ADDITION



AN1074

FIGURE 11: 7-BIT POINTER ACCESSES THE SINE TABLE



5. Only the upper 7-bits of the `PhaseH` register are used to access the table. Effectively scaling the result will achieve the desired range of frequencies.

used the software PWM to control a RGB LED, producing a smooth color sweep.

Equation 3 gives the output sinusoid period for any given ADC value, offset and PWM period.

EQUATION 3:

$$T_{\sin} = \left[\frac{2^{16}}{\text{Offset} + (\text{ADRESH}/2)} \right] \cdot T_{\text{PWM}}$$

T_{\sin} represents the time that it will take the demonstration program to complete one entire sweep of colors before repeating. Replacing in Equation 3 the value '25' for the offset and 10 mSec for the PWM period (T_{PWM}), we obtain a sinusoidal output period varying from 26.22 to 4.32 seconds.

MORE APPLICATION IDEAS

1. Write ADC value directly to the Phase variable. This way, the potentiometer can be used to directly select a (phase angle) color combination.
2. Use a random number generator to produce random color combinations.
3. Connect the application to an outside source of information. Use LED color to indicate stock market price, temperature, audio level etc.

CONCLUSION

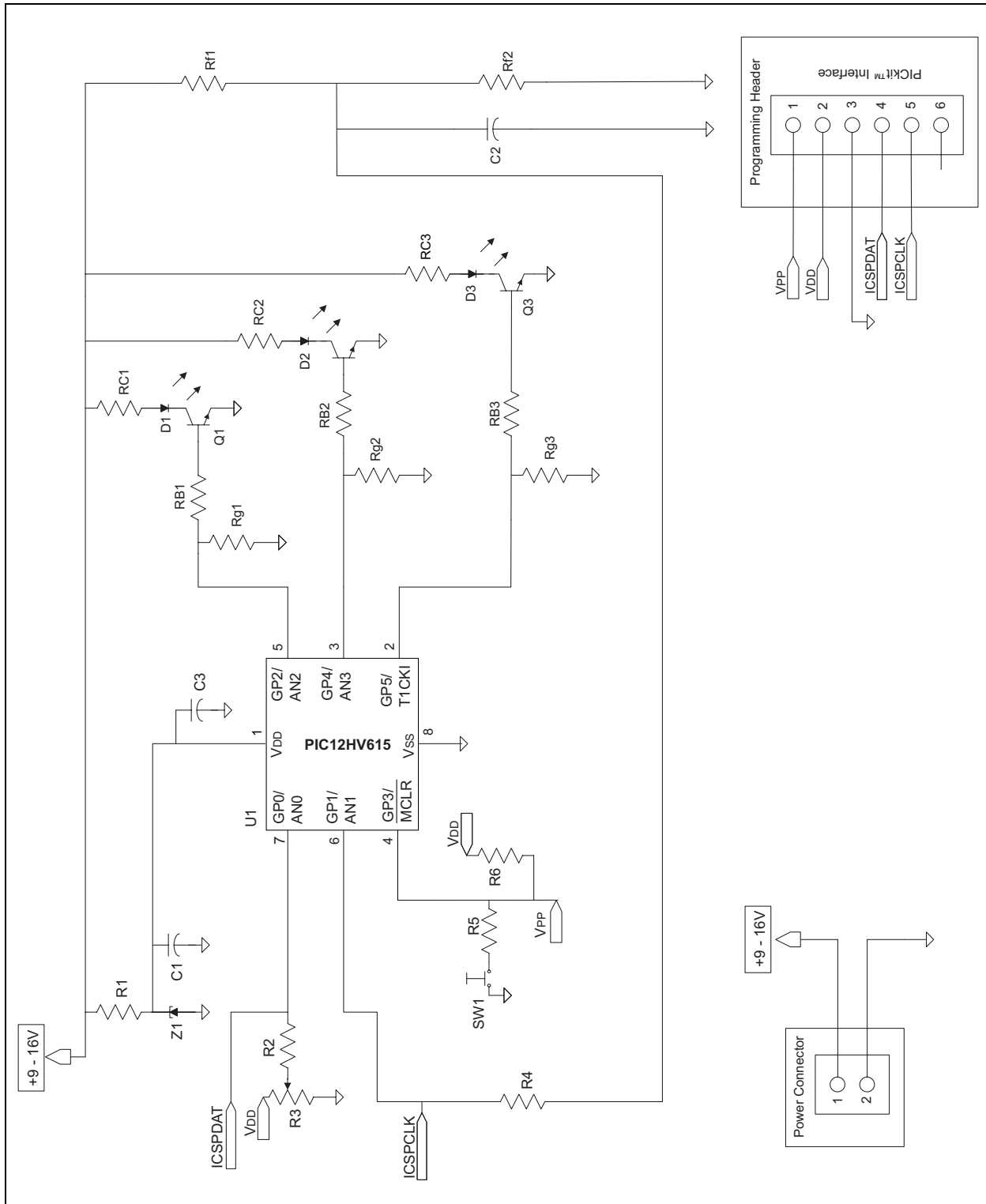
This application note has presented a simple code solution to generate multiple PWM signals in software using a low-cost MCU. The demo

REFERENCE

1. AN654: PWM, a Software Solution for the PIC16CXXX (DS00654)
This application note provides a software solution for a more accurate and flexible PWM output.
2. AN984: Generating a Sinusoidal Waveform (DS00984)
This application note demonstrates how to modulate a PWM signal to a Sinusoidal waveform and how to generate a three-phase sine wave signal in software.
3. PIC12F615 Data Sheet (DS41302)
PICKit™ 2 Microcontroller Programmer User's Guide (DS51553)

AN1074

APPENDIX A: THE DEMO CIRCUIT SCHEMATIC



APPENDIX B: CIRCUIT EXPLANATION

Description of the tri-color LED demonstration circuit schematic:

When using the PIC12HV615 (featuring an internal shunt regulator) the resistor R1 can be used to reduce the input voltage to 5V. The low pass filter composed by R1C1 reduces the ripples of the input voltage. The Z1 Zener diode can be left unpopulated.

On the contrary, if a PIC12F615 or other MCU that does not provide a built-in shunt voltage regulator circuit is used, a 5.1V Zener diode (Z1) will be required.

The PIC12HV615 I/O pins can source directly up to 25 mA each. In order to drive the LED at a rated current of approximately 50 mA, the Q1, Q2 and Q3 transistors are required.

Resistors Rf1 and Rf2 create a voltage divider circuit which provides a feedback path for the DC input voltage. This voltage feedback could be used to help maintain a constant LED brightness as the input voltage changes by changing the PWM duty cycles.

Red, Green and Blue LED emitters have different luminous efficiencies as they do not generate the same luminous intensity given the same input current. Also, the forward voltage across each LED changes so that the amount of current across each LED will be different given the same series resistor value (as in the application schematic). If a precise matching of the luminous output of the three LEDs is required, different collector resistors RC1, RC2 and RC3 might be required

Alternatively, the same resistor value can be used on all three diodes and the output intensity can be adjusted multiplying the duty cycle values of each PWM by an individual corrective constant in software. The advantage of this approach is that it is now possible to calibrate the color output of the tri-color LED without requiring any hardware modifications. This process could be automated to be performed at the end of the production and testing chain, or in a more advanced application, could be performed in a closed control loop provided a color photo sensor is added as a feedback element.

AN1074

APPENDIX C: BILL OF MATERIAL FOR THE DEMO

Item	Qty	Reference	Description
1	3	Resistor, RC1, RC2, RC3	320 Ohm, 1 W, SMT 2512
2	1	Resistor, R1	630 Ohm, 1 W, SMT 2512
3	1	Resistor, R2	470 Ohm, 0.25W, SMT 0805
4	2	Resistor, Rf1, R6	10K Ohm, 0.25W, SMT 0805
5	1	Resistor, Rf2	2.7K Ohm, 0.25W, SMT 0805
6	3	Resistor, RB1, RB2, RB3	470 Ohm, 0.25 W, SMT 0805
7	3	Resistor, Rg1, Rg2, Rg3	10K Ohm, 0.25W, SMT 0805
8	1	Resistor, R4	4.7K Ohm, 0.25W, SMT 0805
9	1	Resistor, R5	470 Ohm, 0.25W, SMT 0805
10	1	Resistor Potentiometer, R3	Bourns Series 3386P (Digi-Key 3386P-103-ND)
11	1	Capacitor, C1	25 μ F, SMT 3528, Tantalum Capacitor
12	2	Capacitor, C2, C3	0.1 μ F, SMT 0805
13	3	Transistor, Q1, Q2, Q3	MMBT3904, SOT23
14	1	Microcontroller	PIC12F683/PIC12HV615
15	1	Push Button, SW1	
16	1	Zener Diode, Z1	5.1V, 0.5W, SOD-123
17	1	Connector, J2	6 Pin Header, 0.1" Spacing
18	1	Connector, J1	Power Connector, Jack for wall transformer, 2.1 mm (Digi-Key CP-202A)
19	1	Three Color PLCC6 LED	OPTEK Technology - OVSTRGBBCR8

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX D: SOURCE CODE FOR SOFTWARE PWM

```
; File: SoftwarePWM.asm
; This file generates 3 software PWM signals using the Timer0 interrupt.
; ISR code is provided in this file that should be executed when
; a Timer0 interrupt occurs. An initialization routine is also provided
; that configures the I/O pins associated with the PWM signals and
; enables Timer0.

list      p=12F615          ; list directive to define processor
#include <p12F615.inc>      ; processor specific variable definitions
#include <SoftwarePWM.h>
errorlevel -302           ; suppress message 302 from list file

global Dutycycle0, Dutycycle1, Dutycycle2, PWMFlags, InitPWM, ISRPWM, IntCount

        UDATA
IntCount      res 1
Dutycycle0    res 1          ; Holds duty cycle for PWM channel 0
Dutycycle1    res 1          ; Holds duty cycle for PWM channel 1
Dutycycle2    res 1          ; Holds duty cycle for PWM channel 2
w_temp        res 1          ; variable used for context saving
status_temp   res 1          ; variable used for context saving
pclath_temp   res 1          ; variable used for context saving
PWMFlags      res 1          ; variable used for software flags

        CODE

ISRPWM
movwf        w_temp          ; save off current W register contents
movf         STATUS,w        ; move status register into W register
movwf        status_temp     ; save off contents of STATUS register

; Write TMR0 to setup next interrupt interval
movlw        .99
movwf        TMR0

;State Machine for PWM starts from here
DecIntCount          ;Decrement IntCount Register
decfsz          IntCount,F          ;decrement IntCount register and if it is zero then make the
                                ;output pin high
goto           Channel0          ;if IntCount register is not zero then go to chk the dutycycle
                                ;of the signal

; If IntCount is 0, then it is time to start a new PWM signal period.
BeginPeriod
bsf          PWM0          ; Set all PWM output pins high
bsf          PWM1
bsf          PWM2
```

AN1074

```
    movlw    .32                ; Initialize IntCount to 32
    movwf   IntCount
    bsf     BeginPWM           ; Set flag for main software loop

    goto    ExitISR           ; Goto end of ISR code

; If it is not the beginning of the PWM period, we need to compare each
; dutycycle to the value of IntCount. This is done by performing a
; subtraction and checking to see whether the result is 0. When a match
; occurs, the output pin for the PWM channel is set to 0.
Channel0
    movf    DutyCycle0,W
    subwf   IntCount,W
    IFNZ                    ; Is IntCount - DutyCycle0 = 0?
    goto    Channel1
    bcf     PWM0              ; Yes, set output pin to 0.

Channel1
    movf    DutyCycle1,W
    subwf   IntCount,W      ;Is IntCount - DutyCycle1 = 0?
    IFNZ
    goto    Channel2
    bcf     PWM1              ;Yes, set output pin to 0

Channel2
    movf    DutyCycle2,W
    subwf   IntCount,W      ;Is IntCount - DutyCycle2 = 0?
    IFZ
    bcf     PWM2              ;Yes, set output pin to 0

;ISR RestoreData
ExitISR
    bcf     INTCON,T0IF      ;clear the T0IF bit in the INTCON register
    movf    status_temp,w   ; retrieve copy of STATUS register
    movwf   STATUS          ; restore pre-isr STATUS register contents
    swapf   w_temp,f
    swapf   w_temp,w        ; restore pre-isr W register contents
    retfie                  ; return from interrupt

;*****
;
;
InitPWM                                ;Initialization for software PWM

    BANK1                        ;Select the RAM memory bank 1

                                ;Timer0 assignment
    movlw   b'01000001'        ;Select the internal clock & /4 prescaler
                                ;0----- GPIO Pull-up Enable bit
                                ;-1----- Interrupt on rising edge of INT pin
                                ;--0----- TOCS->0. Select Timer mode and
                                ;          Internal instruction cycle clock (CLKOUT)
                                ;---0---- T0SE->0. Select falling edge on TOCK1 pin
                                ;----0---- PSA->0. Select Timer0 module for Prescaler
                                ;-----001 PS2:PS0->001. Select 1:4 Prescaler rate
    movwf   OPTION_REG        ;move the value of work register into OPTION register

;GPIO assignment
    movlw   b'00001011'      ;Make GP0, GP1 and GP3 as input and GP2, GP4, GP5 as output
    movwf   TRISIO
```

```
CLRF    ANSEL           ;Initialize ANSEL register for I/O port

bcf     ANSEL,2         ;Digital I/O
bcf     ANSEL,3

BANK0                               ;Select the RAM memory bank 0

CLRF    GPIO           ;Initialize I/O port
clrf    CCP1CON

;Interrupts Initialization
movlw   b'10100000'    ;Enable global and Timer0 interrupts
movwf   INTCON

;Other Variable Initialization
movlw   .32
movwf   IntCount      ;Initialize IntCount to 32

;Variables to hold duty cycle values
movlw   .00
movwf   DutyCycle0
movwf   DutyCycle1
movwf   DutyCycle2

return

END
```

AN1074

APPENDIX E: SOURCE CODE FOR RGB LED DEMO APPLICATION

```
; Filename: Demo.asm
; This demo application uses PWM generation software to drive a RGB
; LED with a 12F615 device. The PWM code is in SoftwarePWM.asm.
; This file initializes the PWM routines and modulates 3 PWM duty
; cycles with a sinusoidal profile. The sinusoidal profile causes
; the RGB LED to sweep through a range of colors. A potentiometer
; voltage is sampled once per PWM period and the conversion result
; is used to set the color sweep speed.

list      p=12F615          ; list directive to define processor
#include <p12F615.inc>      ; processor specific variable definitions
#include <SoftwarePWM.h>
errorlevel -302           ; suppress message 302 from list file

__CONFIG _CP_OFF & _WDT_OFF & _BOR_ON & _MCLRE_ON & _PWRTE_ON & _INTRC_OSC_NOCLKOUT

#define      TABLESIZE      .128

        UDATA
Temp      res 1
PhaseL    res 1          ; PhaseL and PhaseH hold the sinusoidal
PhaseH    res 1          ; pointer location

extern     Dutycycle0, Dutycycle1, Dutycycle2, PWMFlags, ISRPWM
extern     InitPWM, IntCount

;*****
;
;Code starts to run from here
;This will be placed by the linker inside the memory location bet'n 0 and 4

STARTUP CODE

Reset                                ;Reset Vector
    goto      Main

    nop
    nop
    nop

Interrupt                             ;Interrupt Vector
    goto      ISRPWM

;*****
;
; Main code section starts to run from here
PROG CODE                             ;For the rest of the code

Main

    call      InitPWM

    BANK1                                ;Select the RAM memory bank 1

;Configure the input mode of an I/O pin to analog.
    bsf      ANSEL,0                    ;Pot input
```



```

bsf          ANSEL,1          ;For bus voltage feedback

; Setup ANSEL bits for ADC clock period
bsf          ANSEL,4          ;Clock derived from the internal oscillator
bsf          ANSEL,5
bcf          ANSEL,6

BANK0
;A/D Module setup
movlw       b'00000001'
;0----- ADFM->0: Left justified
;-0----- VCFG->0: VDD as Voltage Reference
;--00---- Unimplemented bits
;----00-- CHS1:CHS0->00: AN0 as Analog channel select
;-----0- GO;!DONE->0: A/D conversion not in progress or done
;-----1  ADON->1: A/D conversion enabled

movwf       ADCON0

clrf        PhaseH          ; Initialize the sinusoidal pointer to 0
clrf        PhaseL

; The main program loop starts here. The process of PWM signal
; generation is driven by Timer0 interrupts. A software flag
; called 'BeginBWM' is set in the Timer0 ISR at the start of
; a new PWM period. When the flag is detected in the main loop,
; the potentiometer is sampled, the sinusoidal pointer is incremented,
; and new duty cycle values are determined for the three PWM signals.
MainLoop
    btfss    BeginPWM
    goto     MainLoop
    bcf      BeginPWM

ADCCConversion
    bsf      ADCON0,GO          ;Start A2D conversion
    btfsc    ADCON0,GO          ;Wait until the conversion is completed
    goto     $-1

; The upper 8 bits of the ADC result is added to the 16-bit sine pointer,
; PhaseH:PhaseL. An offset of 25 is also added to make sure the sweep
; frequency never goes to 0.
    rrf      ADRESH,W          ; use only the upper 7 bits of ADC result
    andwf    0x7f              ; mask off upper bit to 0 after rotate
    movwf    Temp
    movlw    .25                ;ADRESH + 25
    addwf    Temp,W
    addwf    PhaseL,F          ;PhaseL+Temp
    btfsc    STATUS,C          ;Chk if the Carry is generated because of the
    ;addition of PhaseL and Temp in the previous
    ;instruction
    incf     PhaseH,F

; To get the three duty cycle values, The upper 7 bits of PhaseH are
; used as a lookup index to the sine table. Offsets of 0x55 (120
; degrees) and 0xAA (240 degrees are used for the 2nd and 3rd duty
; cycles.
ForPhase1
    rrf      PhaseH,W
    andlw    0x7f
    call     SineTable
    movwf    DutyCycle0

ForPhase2
    rrf      PhaseH,W
    addlw    0x55                ;Phase2 = Phase1 + 0x55 for the first phase shift of 120
    ;degree

```

AN1074

```
    andlw    0x7F
    call     SineTable
    movwf    Dutycycle1

ForPhase3
    rrf      PhaseH,W
    addlw    0xAA                ;Phase3 = Phase1 + 0xAA for the second phase shift of 240
degree
    andlw    0x7F
    call     SineTable
    movwf    Dutycycle2

    goto     MainLoop
```

```
; This is a 128 entry table containing 1 cycle of a sine wave.
; A 7-bit table index value is required.
```

```
SineTable
    andlw    TABLESIZE-1      ;Mask off invalid entries
    movwf    Temp
    movlw    high (TableStart) ;get upper address bits for start of table
    movwf    PCLATH
    movlw    low (TableStart)  ;load lower address bits of table
    addwf    Temp,w            ;add offset.
    btfsc    STATUS,C          ;did it overflow?
    incf     PCLATH,f          ;yes:increment PCLATH
    movwf    PCL               ;perform a complete jump
```

```
TableStart
    retlw    .16
    retlw    .17
    retlw    .17
    retlw    .18
    retlw    .19
    retlw    .20
    retlw    .20
    retlw    .21
    retlw    .22
    retlw    .22
    retlw    .23
    retlw    .24
    retlw    .24
    retlw    .25
    retlw    .26
    retlw    .26
    retlw    .27
    retlw    .27
    retlw    .28
    retlw    .28
    retlw    .28
    retlw    .29
    retlw    .29
    retlw    .30
    retlw    .30
    retlw    .30
    retlw    .30
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
    retlw    .31
```

retlw	.31
retlw	.31
retlw	.30
retlw	.30
retlw	.30
retlw	.30
retlw	.29
retlw	.29
retlw	.28
retlw	.28
retlw	.28
retlw	.27
retlw	.27
retlw	.26
retlw	.26
retlw	.25
retlw	.24
retlw	.24
retlw	.23
retlw	.22
retlw	.22
retlw	.21
retlw	.20
retlw	.20
retlw	.19
retlw	.18
retlw	.17
retlw	.17
retlw	.16
retlw	.15
retlw	.15
retlw	.14
retlw	.13
retlw	.12
retlw	.12
retlw	.11
retlw	.10
retlw	.10
retlw	.09
retlw	.08
retlw	.08
retlw	.07
retlw	.06
retlw	.06
retlw	.05
retlw	.05
retlw	.04
retlw	.04
retlw	.04
retlw	.03
retlw	.03
retlw	.02
retlw	.02
retlw	.02
retlw	.02
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01
retlw	.01

AN1074

```
retlw    .01
retlw    .02
retlw    .02
retlw    .02
retlw    .03
retlw    .03
retlw    .04
retlw    .04
retlw    .04
retlw    .05
retlw    .05
retlw    .06
retlw    .06
retlw    .07
retlw    .08
retlw    .08
retlw    .09
retlw    .10
retlw    .10
retlw    .11
retlw    .12
retlw    .12
retlw    .13
retlw    .14
retlw    .15
retlw    .15
```

```
END
```

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


AmpLab, FilterLab, Linear Active Thermistor, Migratable Memory, MXDEV, MXLAB, PS logo, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2007, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona, Gresham, Oregon and Mountain View, California. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

**QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==**



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471-6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820