PWM research and implementation on MCS-51

PWM approach provides an efficient way for gaining output control, as well as another approach named PFM is the other popular way. The principle of PWM is very easy to realize for its operation.

(a) PWM Wave Output

(b) PFM Wave Output

Figure 1. PWM and PFM Waveform

One may hear the terminology like “30% Duty” of squire wave. It means only 30% cycle width remains at high level vs. the whole cycle. Please see Figure 2 to see how it works at 30% Duty of one pulse cycle.

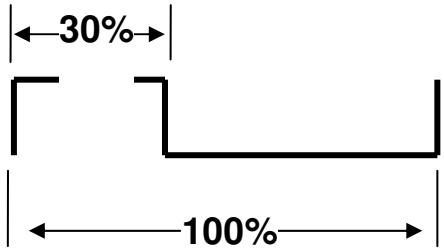


Figure 2. One “Pulse” output with 30% duty.

Before talking about the method, we should introduce some parameters in PWM. The first parameter is the "frequency of PWM" in another word “The Cycle of one PWM pulse”. And the second parameter is "Duty of a cycle". Figure 3 shows the waveform of a PWM output. T_{DH} means time of high duty of a squire wave, T_{DL} is the low duty width and T_C is the period of wave. In general design of PWM, T_C is a constant for the purpose of fixing the PWM output frequency. For a control of PWM, to remain setting T_C and T_{DH} is the main work in a PWM control routine.

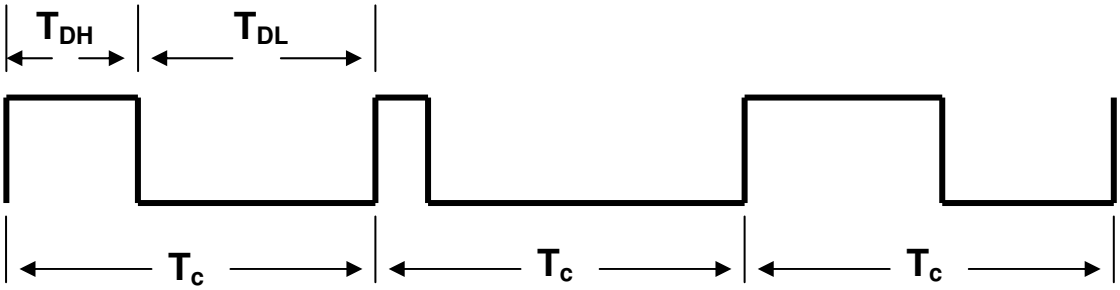


Figure 3. Timing diagram for PWM mechanism

By the way, it is regardless of the frequency of the PWM for how many percentage of the PWM output duty. For some application, the frequency was required under some demand, such as 2Khz frequency for a PWM output, etc. Here, we will introduce some implementations for the design of PWM in MCS-51, and explain how to code for a good control of the output.

First of all, the basic stuff we should notice is how to generate a good pulse in shape. Generally speaking, a good squire pulse means a 50% duty pulse.

Figure 4. Regular Standard Squire Wave (50% duty)

The following phrase of assembly code provides a simple pulse generator and the output waveform will like that of mention above.

```
Loop:
    Setb PWMOUT
    Mov R1,#DutyValue
    Djnz R1,$
    Clr  PWMOUT
    Mov R1,#DutyValue
    Djnz R1,$
    Jmp Loop
```

However, the code cannot provide a good control for changing the duty of a pulse in one cycle. It just can be seen as a square wave regenerator. The cycle of one wave is 2 times of DutyValue. We can add some code to improve the result.

```
Loop:
    Setb PWMOUT
```

```
Mov R1,DutyHigh
Djnz R1,$
Clr  PWMOUT
Mov R1,#DutyValue
Djnz R1,$
Jmp Loop
```

But there still exists a problem; even we can control the high duty via setting the value of **DutyHigh** register, the low duty of a cycle still remains a fix time interval which is refereed to the value of the constant **DutyValue**. For compensation of a pulse in one cycle, the equation shown below should be afforded to remain a fix length pulse width.

$$T_{DL} = T_C - T_{DH}$$

So, with the same delay code phrase (refer to the code above), R1 for high duty is from the value of DutyHigh, if R1 for duty low is from DutyLow, the value of DutyLow of a pulse generation should be

DutyLow = TC - DutyHigh

The pseudo code will be re-written as follow:

```
Loop:
        Setb    PWMOUT
        Mov     R1,DutyHigh
        Djnz    R1,$
        Mov     A,#TC
        Sub     A,DutyHigh
        Mov     DutyLow,A
        Clr     PWMOUT
        Mov     R1,#DutyValue
        Djnz    R1,$
        Jmp     Loop
```

It looks so perfect for a PWM output with good control by only setting DutyHigh, if there is no any consideration about the margin on the side of 0(DutyHigh or DutyLow) .

By now, the code can generate a controllable duty of a PWM output. But the PWM generation code segment is not the whole thing in one application. Let's consider some cases of applications using PWM like motor speed controller, light dimmer... etc.

If the application program structure is like this:

```
        MainLoop:
                ...
                ...
                Call    GetSpeedSetting
                Call    SetDuty
                Call    OnePWMOut
                ...
                ...
                Jmp     Main
;
```

Notice the subroutine 'OnePWMOut' just outputs one pulse wave while calling once. Even a good control for 'ONE' pulse can't be thought a good control on whole PWM output. The reason is the consuming time on other code segments (such as GetSppdSetting and SetDetDuty) is varied, and which will affect the cycle of PWM pulse and then do the same thing on the output of square wave. The following figure shows the improper effect of PWM output result.

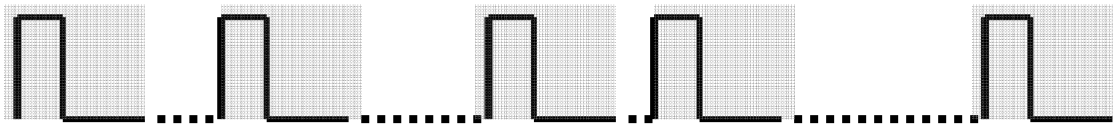


Figure 5. Unsteady PWM wave output

The dot line parts are the uncertain determined under the execution time of the portion of codes except OnePWMOut procedure.

How can we design a good and steady output of PWM? The characteristics of the MPU (single chips) will be a very important indicator. If the chip can just provides the simplest I/O functions, the concept of “State Machine” may generally be applied. This problem will be a little hard to solve. The chip such as PIC16C5X, the approach should be considered to make it work for generating a good PWM wave.

Fortunately, MCS-51 provides the interrupt functions. The timer interrupt subroutine will be applied in the case to make a steady output. Let’s consider the following pseudo code segment working in TIMER mode 2:

Before showing the code, let's take a look at the algorithm first:

Algorithm TMR0_ISR

1. If PWMFlag ==0 then
2. PWMFlag ← 1
3. TMR0 ← -(#Cycle – DutyHigh)
4. Else
5. PWMFlag ← 0
6. TMR0 ← -DutyHigh
7. Return

Pseudo Code List as follow:

```
            JB      PWMFlag,PWMHigh
PWMLow:     Setb    PWMFlag
            Clr     PWMOUT
            Mov     A, DutyHigh
            Sub     A, #PWMCycle
            Mov     TH0,A
            Mov     TL0,A
            Reti
PWMHigh:    Clr     PWMFlag
            Setb    PWMOUT
            Mov     A,#00
            Sub     A,DutyHigh
            Mov     TH0,A
            Mov     TL0,A
            Reti
```

The timer mode 2 provides a reload function for setting timer count. The reload value for timer0 is stored in TH0. So don't forget setting the TH0 in the ISR. And please notice that the timer is a count up timer in msc-51, we should set the value as minus for proper timer counter. The code can work for a steady PWM output under at least 12-mc's cycle. If #PWMCycle is 100 as a constant, then the value in DutyHigh can be 1 to 99. But in actual situation, DutyHigh only work a steady condition while the value is between 15 and 85. Why ? That leaves a room for reader to think about. However, it can work to set the value of DutyHigh register from 1 to 99 to control PWM output. The PWM cycle is determined by the constant PWMCycle. If PWMCycle is 100 then the cycle is 100mc.
The maximum value for PWMCycle is 255.

Notice that only one PWM channel can be afforded for one timer if you use the code above. 8051 only supports 2 timers inside. It means only 2 PWM output can be designed in your application. But in this way, it can provide a good control for changing the duty or output. Like 1% - 100% speed control. The same approach can be used in timer mode 1 or 3 for longer or shorter cycle of PWM output. Readers can develop their idea via the way.

But, if you just have limited usage for timer, say only one timer can be used in PWM generating for more than 1 channel, how can a designer do for the purpose? The follow pseudo code can give an example.

```
PWMPROC:    Djnz    PWMCount,PWM_01
            Mov     PWMCount,#PWMCycle
            Mov     DutyCount,DutyCycle
PWM_01:     Cjne    DutyCount,#00,PWM_02
            Clr     PWMOUT
            Jmp     PWM_03
PWM_02:     Setb    PWMOUT
PWM_03:     Dec     DutyCount
            Ret
```

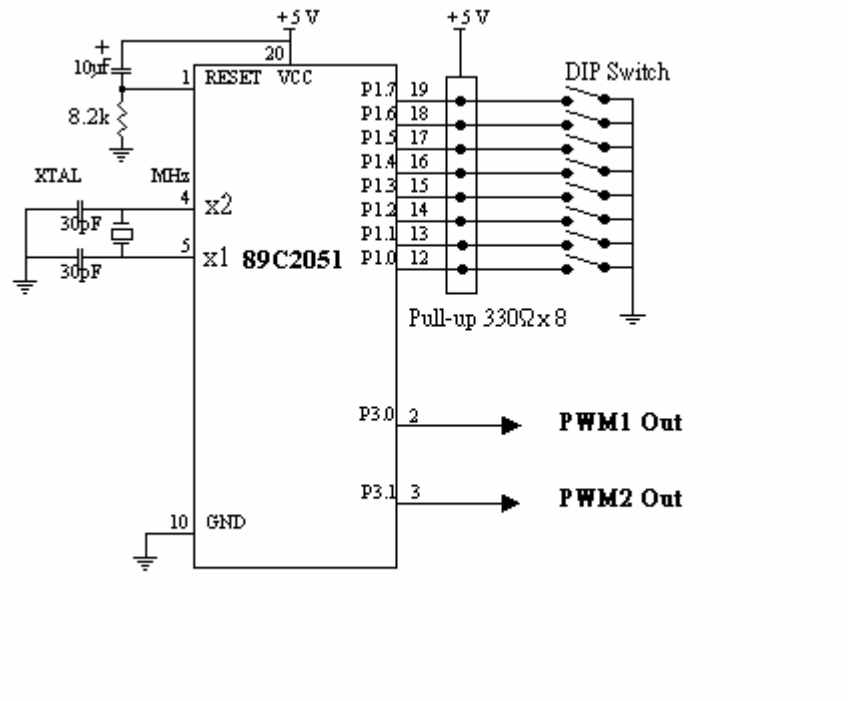
Here, PWMCycle is a constant for fixing PWM cycle. The maximum execution time of the code is 15 mc (machine cycle). If the system has a 50-mc's interval between 2 timer interrupts, then the occupation of the part is 15/50 * 100 % = 33.3%. It means that only two third performance of CPU for main program. In some application, the performance is acceptable. The PWM cycle will be calculated by #PWMCycle x 50 mc. The control level is from 0 ~ PWMCycle. For example, if PWMCycle equals to 10, the PWM Cycle is 10 x 50 = 500 mc. If Fosc is 12 MHz, 1 mc = 1 us. In the case, 2 KHz is the frequency of the PWM output, and the control level can be defined from 0 to 10 only.
Why do we give up the 100% control for output? There are so many reasons. Some applications didn't need to control so in detail. Some applications need more than one PWM output channels. It may lose some for output levels but win for output channels under the strategy. But, how can we increase the PWM channel? Let's go for further step. The following pseudo code works for 2 channels PWM outputs. In practice, PWMCycle1 and PWMCycle2 can be two constants with different values. To analyze the PWM outputs, about 30 mcs execution time in the code section. If the interval between timer interrupt is 100 mcs, the performance of the ISR is 30/100 = 30%. That just remains 70% for main program. Be aware of the PWM cycle equals to #PWMCycleN times Time Interval between timer interrupts. That senses we have to abandon getting high frequency of PWM outputs in order to get more channels.

```
PWMPROC1:   Djnz    PWMCount1,PWM_011
            Mov     PWMCount1,#PWMCycle1
            Mov     DutyCount1,DutyCycle1
PWM_011:    Cjne    DutyCount1,#00,PWM_012
            Clr     PWMOUT1
            Jmp     PWM_013
PWM_012:    Setb    PWMOUT1
PWM_013:    Dec     DutyCount1
;
PWMPROC2:   Djnz    PWMCount2,PWM_021
            Mov     PWMCount2,#PWMCycle2
            Mov     DutyCount2,DutyCycle2
PWM_021:    Cjne    DutyCount2,#00,PWM_022
            Clr     PWMOUT2
            Jmp     PWM_023
PWM_022:    Setb    PWMOUT2
PWM_023:    Dec     DutyCount2
            Ret
```

The above pseudo code gives an example for 2 PWM outputs. Use the same way to cascade the code (please look inside of PWMPROCn) to output additional PWM channel. **However, survey for the above code just gives a hint for designer to generate "Low Frequency Multi-PWM outputs.** All the concepts are described prior to the article.

The follow example gives an implementation for outputting 2 PWM's wave using DIP switch. The speed control is from 0 to 15: 0 for stop, 15 for full speed. Using the same design can be afforded using RS-232, AD control…, etc. The advantage of this code is to improve the way during setting speed control inside of the program. Anytime when you want to change the output, just update the value of registers DutyCycle1 or DutyCycle2, the ISR will automatically execute the core process of PWM outputs. It let the designer need not waste time to synchronize the output using complex way such as FSM.

Example Schematic:



Example Source Code:

```
;Project: Sample code for Dual PWM outputs
;                       Written By Charles C. L.
;
;Assembler : 2500 A.D. 8051 Macro Assembler
;Chip: 89C2051
;
; P3.0 for PWM1 output
; P3.1 for PWM2 output
;
; P1.0~P1.3 : Speed Control for PWM1
; P1.4~P1.7 : Speed Control for PWM2
;
; Definition of I/Os
PWMOUT          REG         P3
DIPSW           REG         P1
PWM1Out         REG         PWMOUT.0
PWM2Out         REG         PWMOUT.1
;
; Registers for PWM1
PWMCount1       REG         R2
DutyCount1      REG         R3
DutyCycle1      EQU         31h

; Registers for PWM2
PWMCount2       REG         R4
DutyCount2      REG         R5
DutyCycle2      EQU         32h

; Common Flags for PWM (unused in the sample)
PWMFLAG         EQU         40h
;
;Constant for timer0 setting
TMR0VAL         EQU         -100
;
;Constantans for PWMs' Cycle setting
PWMCycle1       EQU         15
PWMCycle2       EQU         15
;
; Program start here
```

```
;
; Reset vector
                ORG     000h
                JMP     Main
;
; Timer 0 interrupt vector
                ORG     00Bh
                JMP     TIMER0
;
; Setup Stack Pointer and then
; Initialize System and PWM output
;
Main:           Mov     SP,#70h
                Call    SysInit
                Call    PWMInit
;
; The real main program is in a program loop
; Here is the beginning of the loop
MainLoop:
                Mov     A,DIPSW     ;Read DIP switch
                Anl     A,#0Fh  ;   Get the LSNibble
                Mov     DutyCycle1,A    ;   Setting for PWM1
                Mov     A,DIPSW     ;Read DIP switch again
                Swap    A
                Anl     A,#0Fh  ;   Get the MSNibble
                Mov     DutyCycle2,A    ;   Setting for PWM2
                Call    Delay   ;Little Delay for stablization
                Jmp     MainLoop    ;Forever Loop for normal operation
;
; System Initialization
;
SysInit:
                MOV     TMOD,#00010010B     ; TMR0:MODE 2 TMR1:MODE 1
                MOV     TH0,#TMR0VAL  ; Initial Timer0 Value
                MOV     TL0,#TMR0VAL        ;       with TMR0VAL
                MOV     IE,#10000010B   ; Enable Timer0 interrupt
                SETB    TR0                 ; Start Timer0
                Ret             ; Return

PWMInit:
; Set up parameters for full stop of PWM1
```

```
                Mov     PWMCount1,#PWMCycle1
                Mov     DutyCount1,#0
                Mov     DutyCycle1,#0
;
; Set up parameters for full stop of PWM2
                Mov     PWMCount2,#PWMCycle2
                Mov     DutyCount2,#0
                Mov     DutyCycle2,#0
                Ret
;
; The Timer0 Intrrupt Subroutine
;       gives a 100us period per interrupt
;
TIMER0:

PWMPROC1:
                Djnz    PWMCount1,PWM_011
                Mov     PWMCount1,#PWMCycle1
                Mov     DutyCount1,DutyCycle1
PWM_011:
                Cjne    DutyCount1,#00,PWM_012
                Clr     PWM1Out
                Jmp     PWM_013
PWM_012:
                Setb    PWM1Out
PWM_013:
                Dec     DutyCount1
;
PWMPROC2:
                Djnz    PWMCount2,PWM_021
                Mov     PWMCount2,#PWMCycle2
                Mov     DutyCount2,DutyCycle2
PWM_021:
                Cjne    DutyCount2,#00,PWM_022
                Clr     PWM2Out
                Jmp     PWM_023
PWM_022:
                Setb    PWM2Out
PWM_023:
                Dec     DutyCount2
                Reti
```

```
;
Delay:      Mov     R7,#100
Dly0:       Mov     R6,#100
Dly1:       Nop
            Nop
            Djnz    R6,Dly1
            Djnz    R7,Dly0
            Ret
```