Lab 6 – Programmable timer and PWM (Pulse Width Modulation)

Objectives

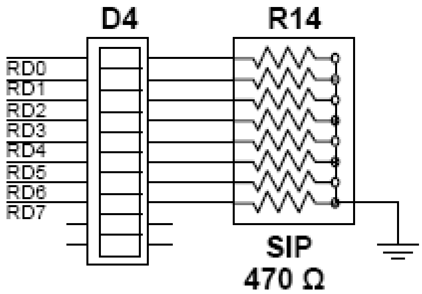
- ☐ To learn to introduce a time delay using Timer0 in the PIC18F4550 microcontroller.
- ☐ To learn to use PWM for the speed control of a DC motor.

Introduction / Briefing

PIC18F4550's Timer0 for delay

- ☐ The PIC18F4550 microcontroller has four internal timers. We will first use Timer0 to introduce a time delay and then use Timer2 to control the speed of a DC motor using Pulse Width Modulation (PWM).

- ☐ In the first part of the experiment, the LED bar at Port D will be blinked (turned ON and OFF repeatedly) at 1 second interval. Let's figure out how the 1 second interval or delay can be created with the help of Timer0.



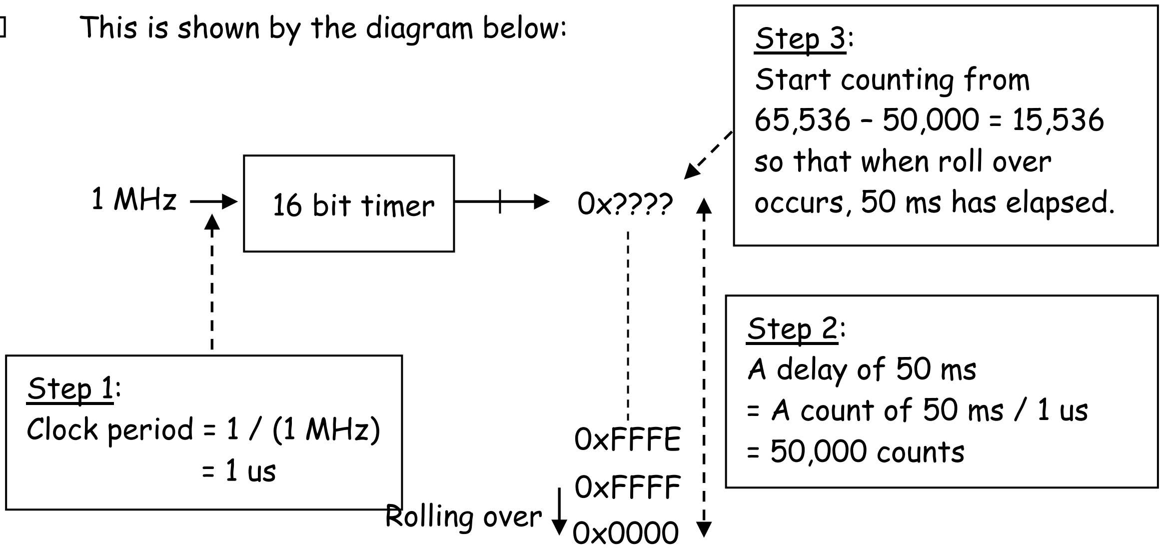
- ☐ Example 1: Assume that a 16-bit counter/timer is clocked by a 1 MHz clock signal. How long does it take to count from 0000 to FFFF and then roll over? [Roll over means changing from the maximum count of FFFF to 0000.]

- ☐ Counting from 0000 to FFFF and then rolling over is equivalent to 65,536 counts. Since each count takes 1us, this is equal to 65,536 us = 65.536 ms.

- ☐ Example 2: Now try this: what count should the counter starts with, so that exactly 50 ms has elapsed when roll over occurs?

- ☐ Since each count takes 1us, 50 ms is equal to 50 ms / 1us = 50,000 counts. So the count should start from 65,536 – 50,000 = 15,536.

☐ This is shown by the diagram below:



☐ In the case of PIC18F4550's Timer 0:

- The timer/counter clock frequency = Fosc / 4 = 48 MHz / 4 = 12 MHz.
- An interrupt flag TMR0IF will be set to 1 whenever the timer overflows from FFFF to 0000.
- A pre-scaler can be used to slow down the clock. For instance, a pre-scale value of 256 slows down the clock by 256 times, effectively, the timer/counter is clocked by a 12MHz / 256 clock signal.

☐ Example 3: Assume that the PIC18F4550's 16-bit Timer0 is clocked by a 48 MHz / 4 clock signal and a pre-scale value of 256 is used. What count should the timer starts with, so that exactly 1 sec has elapsed when roll over occurs?

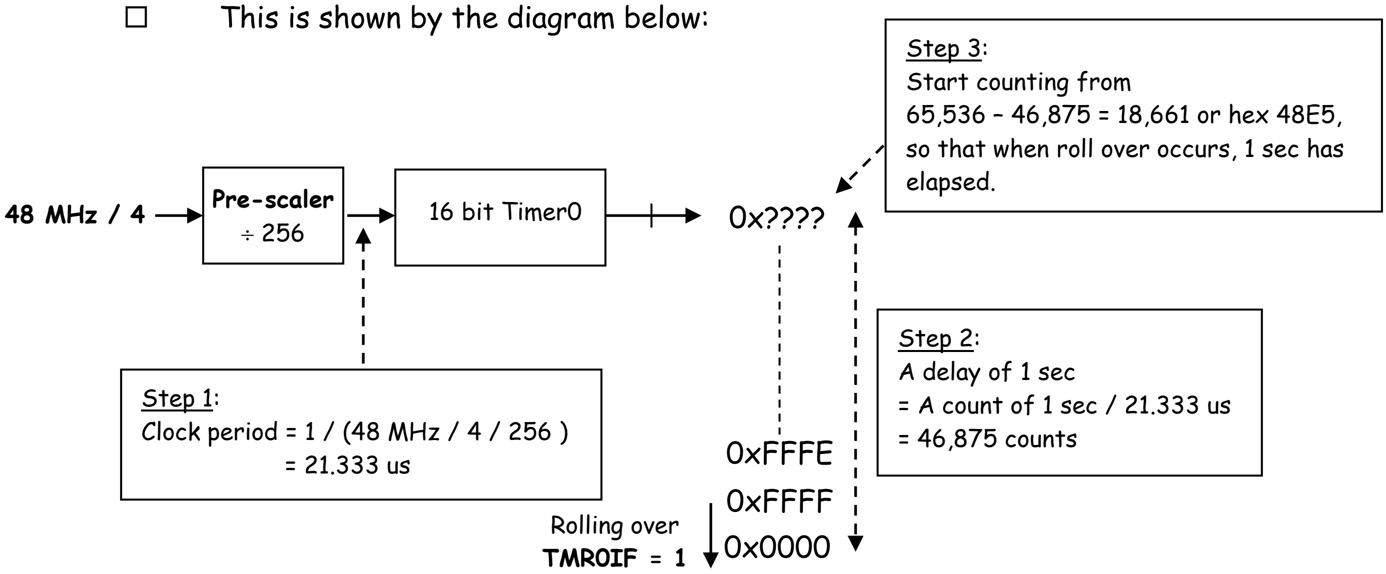
☐ Effective clock frequency for Timer0 = 48 MHz / 4 / 256 = 46875 Hz

☐ Each count = 1 / (46875 Hz) = 21.333 us.

☐ A delay of 1 sec = a count of 1 sec / 21.333 us = 46,875 counts.

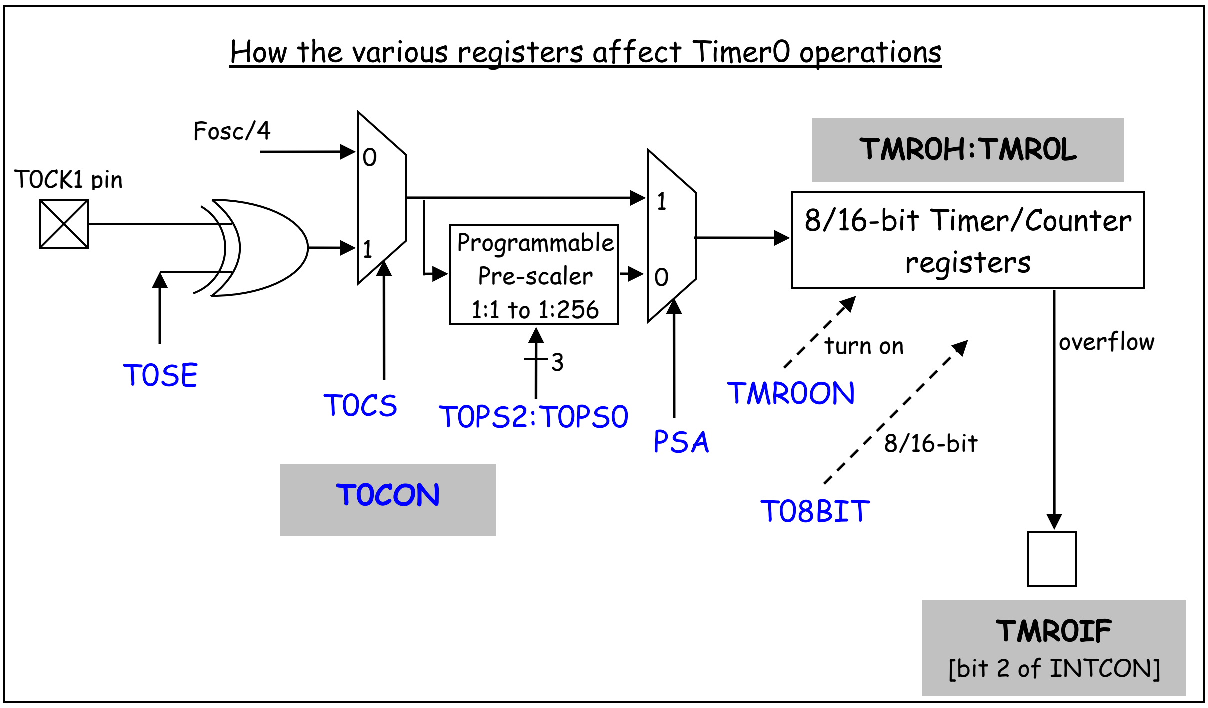
☐ So the Timer0 should start from 65536 – 46875 = 18661 or hex 48E5 – the conversion from decimal to hex can be done using the PC's calculator (Programs -> Accessories -> Calculator).

- This is shown by the diagram below:



- How do you set a pre-scaler of 256 & a starting count value of hex 48E5? How do you know that Timer0 overflow has occurred i.e. TMR0IF has become 1? To answer these questions, we must take a look at Timer0 registers.

- The following diagram shows how the various Timer0 registers affect its operations. You can come back to examine this diagram later, after the individual registers have been described.



TMR0H & TMR0L (Timer0 High & Low Registers) – These two 8-bit registers together form a 16-bit timer/counter.

TMR0H								TMR0L							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

☐ To set a starting count value of hex 48E5, hex 48 should be written to TMR0H first, and then hex E5 written to TMR0L.

Q1: *Give the C code to set a starting count value of hex 48E5:*

Your answer: TMR0H = _______;
_______________ ;

INTCON (Interrupt Control Register) bit 2 =TMR0IF (Timer0 Interrupt "overflow" Flag) – This bit is set to 1 whenever the Timer0 overflows i.e. count from FFFF to 0000.

INTCON

					TMR0IF		

Q2: *Give the C code to wait for Timer0 overflow to occur:*

Your answer: while (_____bits._____ == 0);

T0CON (Timer0 Control Register) – This register controls the Timer0 operation (as described below). It is used to turn the timer ON/OFF and to set the pre-scaler value.

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

TMR0ON D7 Timer0 **ON** and **OFF control** bit
1 = Enable (start) Timer0
0 = Stop Timer0

T08BIT D6 Timer0 **8-bit** / **16-bit selector** bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

T0CS D5 Timer0 **clock source select** bit
1 = External clock from RA4/T0CK1 pin
0 = Internal clock (Fosc/4 from XTAL oscillator)

T0SE	D4	Timer0 **source edge select** bit 1 = Increment on H-to-L transition on T0CK1 pin 0 = Increment on L-to-H transition on T0CK1 pin
PSA	D3	Timer0 **pre-scaler assignment** bit 1 = Timer0 clock input bypasses pre-scaler 0 = Timer0 clock input comes from pre-scaler output

T0PS2:T0PS0

D2 D1 D0	Timer0 **pre-scaler selector**
0 0 0 = 1:2	Pre-scale value (Fosc/4/2)
0 0 1 = 1:4	Pre-scale value (Fosc/4/4)
0 1 0 = 1:8	Pre-scale value (Fosc/4/8)
0 1 1 = 1:16	Pre-scale value (Fosc/4/16)
1 0 0 = 1:32	Pre-scale value (Fosc/4/32)
1 0 1 = 1:64	Pre-scale value (Fosc/4/64)
1 1 0 = 1:128	Pre-scale value (Fosc/4/128)
1 1 1 = 1:256	Pre-scale value (Fosc/4/256)

Q3: What binary pattern must be written to T0CON to use Timer0 as a 16-bit timer using internal clock (Fosc/4) and a pre-scale value of 256? The timer is NOT to be turned on at this point.

Your answer:

T0CON (Timer0 Control Register)

TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

☐ The C code required is T0CON = 0b 0 0 0 0 0 111;

☐ Putting the pieces together, the C code to introduce a time delay of 1 second can be written as follows:

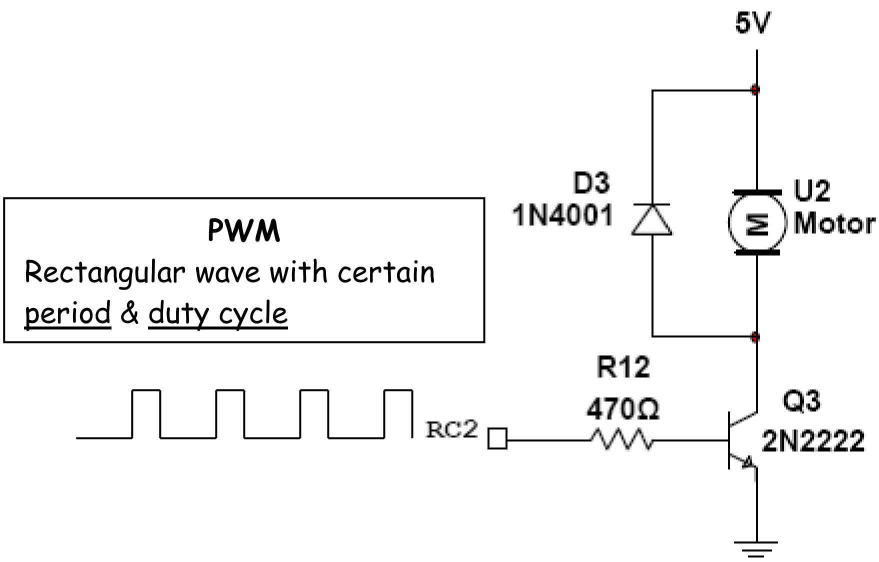
```
T0CON=0b00000111;              // Off Timer0, 16-bits mode, Fosc/4, prescaler of 256

TMR0H=0X48;                    // Starting count value
TMR0L=0XE5;

INTCONbits.TMR0IF=0;           // Clear flag first
T0CONbits.TMR0ON=1;            // Turn on Timer 0

while(INTCONbits.TMR0IF==0);   // Wait for time's up i.e. TMR0IF==1
T0CONbits.TMR0ON=0;            // Turn off Timer 0 to stop counting
```

☐ Description: 1. First, Timer0 is configured but turned OFF. 2. Then, the starting count value is written toTMR0H, followed by TMR0L. 3. Next, the flag is cleared and Timer0 turned ON. 4. After that, the while loop is used to wait for 1 second to elapse i.e. for the TMR0IF interrupt flag to be set. 5. Finally, Timer0 is turned OFF.

☐ With this, you should be able to figure out the TimerDelay.c used in the first part of the experiment to blink the LED bar at Port D at 1 second interval.

PIC18F4550's Timer2 for PWM

☐ In the second part of the experiment, PWM (Pulse Width Modulation) will be used to control the speed of the DC motor connected to RC2. We will now describe what PWM is and how it can be created with the help of Timer2.

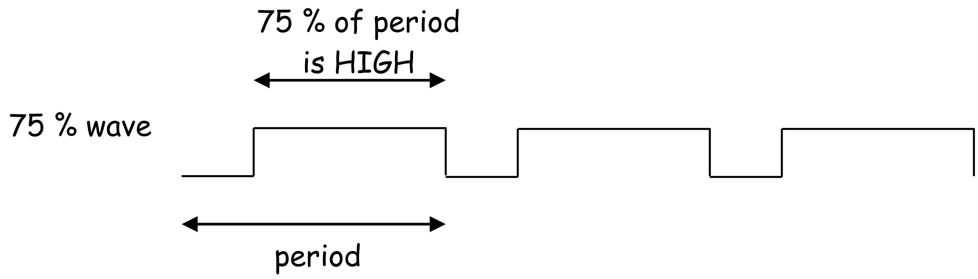


PWM

- PWM (Pulse Width Modulation) is a method used to control the speed of a DC motor.

- When 5V is applied to a small DC motor, it turns at a certain speed.

- A 75% duty cycle rectangular wave is high for 75% of the time (and low for 25% of the time). When this is applied, the motor slows down – effectively, it is getting 5V x 75% or 3.75V d.c.



- When a 50% duty cycle wave is applied, the motor slows down further, as it is effectively getting 2.5V d.c.

- Pulse Width Modulation = varying the duty cycle of the rectangular wave (i.e. varying the pulse width) to control the motor speed.

- In creating a rectangular wave or pulse train, we must know both 1. the <u>period</u> and 2. the <u>duty cycle</u>.

- How do we create a rectangular wave of a certain period and duty cycle in PIC18F4550? Let's try a 5 kHz, 25% duty cycle wave.

- PIC18F4550 has a **CCP** (Capture Compare) module which comes with PWM capability. The PWM output comes out at RC2.

- For PWM, the CCP module uses two **Timer2** registers to specify the **period**:

PWM period = (PR2 + 1) x 4 x N x Tosc

where PR2 is Timer2's 8-bit "Period register"
N = Timer2's pre-scale value of 1, 4 or 16, as set in T2CON (Timer2 Control) register (* see box on next page)
Tosc = 1 / Fosc, where Fosc = 48 MHz

Some Timer2 registers

PR2 (Period Register)

D7	D6	D5	D4	D3	D2	D1	D0

T2CON (Timer2 Control Register)

D7	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

D7 – not used

TOUTPS3:TOUTPS0	D6 D5 D4 D3	Timer2 output **post-scaler selector** 0 0 0 0 = 1:1 Post-scale value 0 0 0 1 = 1:2 Post-scale value 0 0 1 0 = 1:3 Post-scale value … 1 1 1 1 = 1:16 Post-scale value
TMR2ON	D2	Timer2 **ON** and **OFF control** bit 1 = Enable (start) Timer2 0 = Stop Timer2
T2CKPS1:T2CKPS0	D1 D0	Timer2 clock **pre-scaler selector** 0 0 = 1:1 Pre-scale value 0 1 = 1:4 Pre-scale value 1 X = 1:16 Pre-scale value

Q4. What is the PR2 value to generate a 5 kHz wave, assuming a pre-scaler of 16?

Your answer: ________________

☐ Solution

Frequency = 5 k ⇨ Period = 1 / 5k = 0.2 m

Tosc = 1 / 48M Pre-scaler, N = 16

Substituting into the formula,

PWM period = (PR2 + 1) x 4 x N x Tosc

1 / 5k = (PR2 + 1) x 4 x 16 x (1 / 48M)

=> PR2 = 149

☐ The "**High Time**" (or "On Time") is specified using another register called **CCPR1L**, as follows:

High Time = 25% of Period

25% x 149 = 37.25 = 37 (ignoring the decimal portion)

=> <u>CCPR1L = 37</u>

☐ The bottom 4 bits of the CCP1 Control Register (**CCP1CON**) should be set to 1100 for PWM operation.

CCP1CON (CCP1 Control Register)

D7	D6	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

1 1 0 0 for PWM mode

Other combinations for Capture / Compare modes not studied in this module

☐ Let's put everything together to program the PIC18F4550 to generate a 5 kHz, 25% wave.

☐ The complete program (in outline form) is given below:

```
TRISC=0x00;   // RC2 is connect to motor and should be made an output
T2CON=0b0 0000 1 11;  // Timer2 is On, Prescaler is 16
CCP1CON=0b00 00 1100;    // Turn on PWM
PR2 = 149;               // Load PWM period of 0.2 ms or 5 kHz
CCPR1L = 37        ;     // Load PWM on time ( i.e. 25% x 149 )
```

☐ With this, you should be able to figure out the TimerPWM.c used in the second part of the experiment to control DC motor speed using PWM.

Activites:

Before you begin, ensure that the Micro-controller Board is connected to the General IO Board.

Blinking an LED bar at 1 second interval, using a delay created using Timer0

1. Launch the *MPLABX IDE* and create a new project called *Lab6*.

2. Add the file *TimerDelay.c* to the *Lab6* project *Source File* folder. Make sure *Copy* is ticked before you click *Select*. If you have forgotten the steps, you will need to refer to the previous lab sheet.

3. Study the code and describe what this program will do:

   Blink LED bar at 1 sec intervals.

   ____

4. Note that the main function configures the *Timer0* but does not turn it ON. The *Timer0* is only turned on in the *Delay1sec* function.

5. Build, download and execute the program. Observe the result and see if it is as expected.

Change start count value to change delay.

6. Modify the code so that the delay is 0.5 second (instead of 1 second). Keep pre-scaler of 256.

Hint:

- Since Fosc remains at 48 MHz and pre-scaler remains at 256, the effective clock frequency of Timer0 remains unchanged at 48 MHz / 4 / 256 = 46875 Hz.
- Each count = 1 / (46875 Hz) = 21.333 us, the same as before.
- A delay of 0.5 sec = a count of 0.5 sec / 21.333 us = 23438 counts.
- So the Timer0 should start from 65536–23438 = 42098 or hex A472.

7. Build, download and execute the program to verify your coding. The LED bar now blinks at a faster rate.

Change pre-scaler to change delay.

8. Without changing the start count value in 6 above, modify the code to use a pre-scaler of 64 (instead of 256). [Hint: T0CON = 0b00000___;] What effect do you think this will have on the rate of blinking?

   Your answer: The LED bar will blink at a ______ (faster/slower) rate.

9. Build, download and execute the program to verify your answer above.

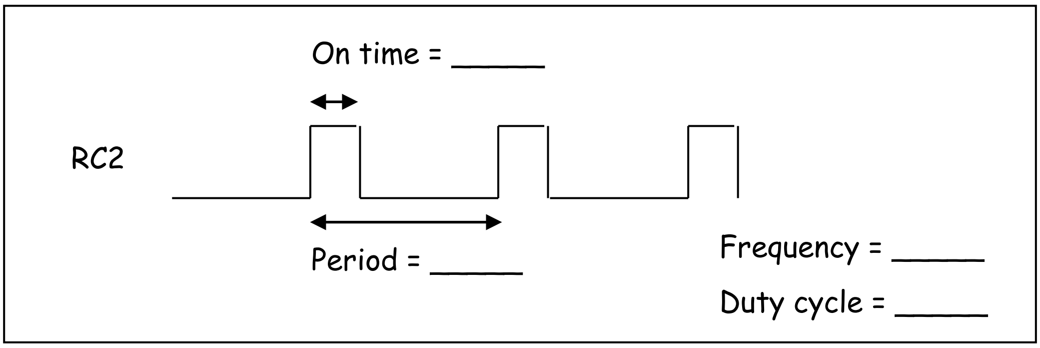
Controlling DC motor speed, using PWM created using Timer2

10. Replace *TimerDelay.c* with*TimerPWM.c*.

PWM 5kHz 25% duty cycle

11. The TimerPWM.c code is to produce a 5 kHz, 25% duty cycle wave at *RC2* using PWM.

12. Build, download and execute the program. Use the oscilloscope connected to *RC2* (/ DC motor) to see if the period & duty cycle are correct. Record your observations below:



13. Note that *PR2* = 149, *CCPR1L* = 37 in this case.

14. Note also the speed of the DC motor at 25%.

PWM 5kHz 25%, 50% or 75% duty cycle depending on dip switches

15. The value of *CCPR1L* to get a 5 kHz, 50% duty cycle wave is simply

    50% x 149 = 74

    Likewise, to get a 5 kHz, 75% duty cycle wave, *CCPR1L* = 75% x 149 = 112

16. Modify the code so that the duty cycle produced depends on the settings of the (active low) dip switches (on the General IO Board) connected to *RA4* and *RA3*, as follows:

RA4	RA3	Duty Cycle
Closed i.e. == 0	don't care	75 % (high speed)
Open i.e. == 1	Closed i.e. == 0	50 % (medium speed)
Open i.e. == 1	Open i.e. == 1	25 % (low speed)

17. Build, download and execute the program to verify your coding. Debug until the program can work.

// TimerDelay.c

```
/* TimerDelay.c Program containing a 1 sec delay function
*  Use Timer0
*  Frequency of OSC = 48 MHz, Prescaler = 256
*  TMR0H:TMR0L contain the starting count value
*  Monitor TMR0IF flag. When TMR0IF = 1, one sec is over
*/
#include <xc.h>

void Delay1sec(void);                       // Function to provide 1 sec delay using Timer0
void Delay1sec(void)
{
   TMR0H=0X48;                              // Starting count value
   TMR0L=0XE5;

   INTCONbits.TMR0IF=0;                     // Clear flag first
   T0CONbits.TMR0ON=1;                      // Turn on Timer 0

   while(INTCONbits.TMR0IF==0);             // Wait for time is up when TMR0IF=1
   T0CONbits.TMR0ON=0;                      // Turn off Timer 0 to stop counting
}

void main(void)
{
   TRISD=0x00;                              // PortD connected to 8 LEDs
   T0CON=0b00000111;                        // Off Timer0, 16-bits mode, prescaler to 256

   while(1)                                 // Repeatedly
   {
      PORTD=0x00;                           // Off all LEDs
      Delay1sec();

      PORTD=0xFF;                           // On all LEDs
      Delay1sec();

   }
}
```

// TimerPWM.c

```
/* TimerPWM.c Program to generate PWM at RC2
*  Use Timer2
*  Frequency of OSC = 48 MHz, Prescaler = 16
*  PR2 register set the frequency of waveform
*  CCPR1L with CP1CONbits.DC1B0, CCP1CONbits.DC1B1 set the On-Time
*  Use Timer0 for the one second delay function
*/
#include <xc.h>

void Delay1sec(void);               // Function to provide 1 sec delay using Timer0
void Delay1sec(void)
{
   TMR0H=0X48;                      // Starting count value
   TMR0L=0XE5;

   INTCONbits.TMR0IF=0;             // Clear flag first
   T0CONbits.TMR0ON=1;              // Turn on Timer 0

   while(INTCONbits.TMR0IF==0);     // Wait for time is up when TMR0IF=1
   T0CONbits.TMR0ON=0;              // Turn off Timer 0 to stop counting
}

void main(void)
{
// Do not remove these as well============
   ADCON1 = 0x0F;
   CMCON = 0x07;
// ======================================
// Your MAIN program Starts here: =========

   TRISC=0x00;                      // PortC RC2 connects to motor
   TRISD=0x00;                      // PortD connected to 8 LEDs
   T0CON=0b00000111;                // Off Timer0, 16-bits mode, prescaler to 256

   T2CON=0b00000111;                // Timer2 is On, Prescaler is 16

   CCP1CON=0b00001100;              // Turn on PWM on CCP1, output at RC2
   PR2 = 149;                       // Load period of PWM 0.2msec for 5KHz

   while(1)                         // Repeatedly
   {
      CCPR1L = 37;                  // Duty cycle 25%, 149 x 25% = 37
   }
}
```