Setting and using the timers and interrupts.

There are 4 timers which can be selected to use in PIC18F4550.Most of time, these timers work with interrupts. The interrupts can be set as high or low priority.

Before we start to set the timer, there is some basis that we need to know.
The internal speed of PIC18F4550 is 48MHz. One instruction takes 4 clock cycles.
The timer can be chosen to increment at each internal instruction cycle (Fosc/4) or at the rising (or falling) edge of external clock source. When using the internal clock, this means:
Timer frequency = Fosc/ 4 = 48MHz/ 4 = 12MHz.

Secondly, the timer has programmable prescaler. This can be used for decreasing the timer frequency again. E.g. set prescale value = 1:16. This will define timer frequency = (Fosc/4) *(1/16) = 0.75MHz.

The last thing is the relation between the timer and interrupt. When using timer 0, timer1, or timer3, they can set interrupt flag when overflow occurs. (Timer 2: use TMR2 register makes comparison with value of PR2. When match occurs, the interrupt flag will be set.) Then this interrupt flag will lead to the corresponding priority interrupt. The interrupt service routine will be executed.

Now we can start to config the timer and interrupt.

1. Library.
If you are going to use the commands like OpenTimerx, WriteTimerx, ReadTimerx, and CloseTimerx, (x stands for 0~3, which concerns with the timer you choose) make sure the header file <timers.h> is included.

#include <timers.h>
Add this library at the top of the source file.

2. Interrupt service routine prototype.
When you want to use the interrupt service routine, the prototype must be declared first.

void timer1_isr(void); //interrupt service routine Prototype for timer1
Add this prototype after the library declaration.

3. Config Timer0, Timer1 and Timer3
3.1 initialize the timer

Setting Timer0, Timer1 and Timer 3 are almost the same, because they have almost the same structures. Timer0 can be selected as 8-bit mode or 16-bit mode. And timer1 and Timer3 work in 16-bit mode. The timer will use the registers as shown below.

TMRxH	TMRxL

	Higher bits	Lower bits
• Timer0 (16 bit):	TMR0H,	TMR0L
• Timer1 (16 bit):	TMR1H,	TMR1L
• Timer3 (16 bit):	TMR3H,	TMR3L

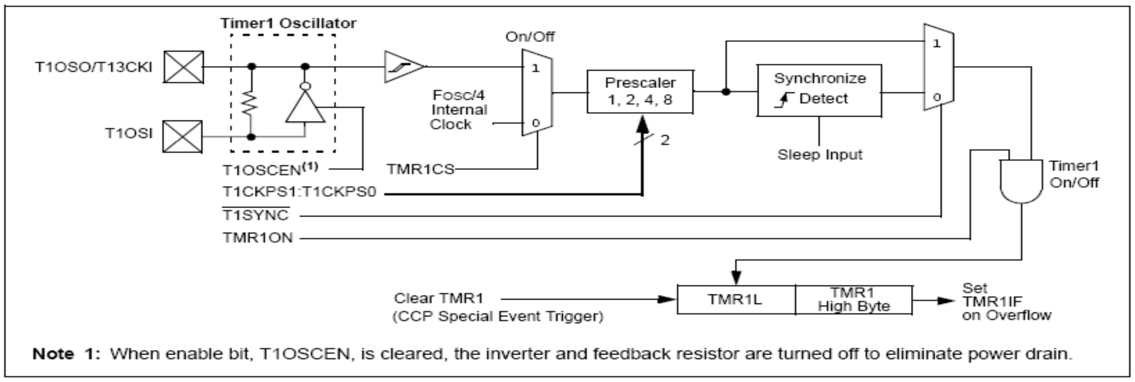
When the timer0 is set as 8 bit mode, lower bits will be used only.

In the following parts, we take the timer1 as an example to see how it can be set.

```
OpenTimer1(  TIMER_INT_ON &           // Interrupt enabled
             T1_8BIT_RW &             //set timer1 as two 8-bit registers
             T1_SOURCE_INT &          //choose internal clock source (TOSC)
             T1_PS_1_8 &              // Prescale Value: 1:8
             T1_OSC1EN_OFF &          //Disable Timer1 oscillator
             T1_SYNC_EXT_OFF          //Don't sync external clock input
           );
```

The interrupt is enabled; select the timer as two 8-bit-register mode. And the timer uses an internal clock source, the Prescale Value is chosen as 1:8. See the schematic of timer1 below.

Timer Freq = (48MHz / 4)*(1/8) = 1.5MHz
The internal structure of the timer1:



The next thing is setting the time for generating the interrupt.

3.2 Period setting:
Suppose you want a timer interrupt at every 20 millisecond, the starting value for timer register will be carried out.

As we known, the timer can set flag at the overflow happens. Then, because timer1 is 16bit, so overflow occurs at timer value reaches hexadecimal value 0xFFFF. (Decimal: 65535)
As we have set before, the timer1 frequency is 1.5MHz, this means that the timer value (TMR1H:TMR1L) will increment in this speed. Then we need to know the amount of the timer value for 20 ms:

Wanted interrupt time*timer frequency = 20 *(e^-3) * 1.5*(e^6) = 30000

Last step will be the calculation for the starting value, which means that the timer will increment from this value till 0xFFFF, and then overflow occurs.

Timer start value = 65535 – 30000 = 35535= 0x8ACF.
Then we write this value in the command below.

WriteTimer1(0x8ACF); // set start value of timer, set interrupt at every 20 ms.

When overflow happens, the interrupt flag will be set; the interrupt can be called, then the service routine can be executed.

3.3 Enable the interrupts.
Before you enable the interrupts, you should set the priority of the interrupts you used.

RCONbits.IPEN = 1;   //Enable priority levels on interrupts
RCONbits.SBOREN = 0; //Disable BOR

Then, enable the high or low priority interrupts from the register INTCON,
INTCON = 0b10000000; //enable the high priority interrupts, bit7 and 6 are concerned to high and low priority.

Last step will be that the corresponding timer interrupt priority should be set and to enable the interrupt.
These settings for the different timers are located in different registers; you can find them in datasheet. (TMRxIP: interrupt priority bit, TMRxIE: interrupt enable bit)
http://ww1.microchip.com/downloads/en/DeviceDoc/39632D.pdf

IPR1bits.TMR1IP = 1; //Timer1 interrupt priority high
PIE1bits.TMR1IE = 1; //Timer1 interrupt enable

Now timer1 can count from 0x8ACF, till 0xFFFF. After that overflow will happen, then this overflow will set the interrupt flag. If the system finds there is any interrupt flag set, then it will jump into defined interrupt service routine.

3.4 Give the direction for the ISR from bootloader

```
void high_ISR (void)
{
_asm goto timer1_isr _endasm

}
```

When you set timer1 interrupt as high priority, then you should add
_asm goto timer1_isr _endasm under the high_ISR function.

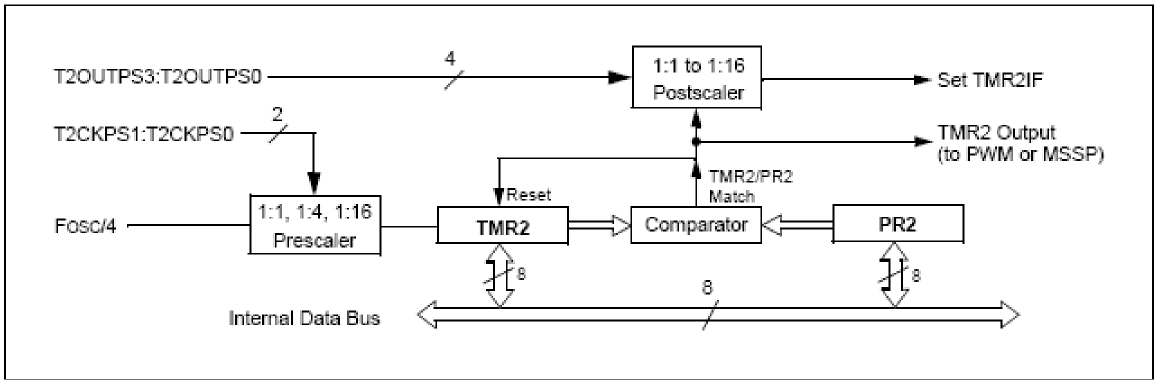
3.5 Implement the timer1 interrupt service routine

```
#pragma interrupt timer1_isr
void timer1_isr(void)
{
PIR1bits.TMR1IF = 0; //Reset Timer1 interrupt flag
WriteTimer1(0x8ACF); // Give new start value to the timer1


//write your codes …………………


}
```

Now you can add this function before the main function. And add your codes in. Remember every time you need to reset the interrupt flag, and give the start value to timer (otherwise the timer will roll over to 0x0000, then start to increase from this 0x0000). When timer value reaches 0 x FFFF, the overflow flag will lead interrupt generated again.

4. Config timer 2.

Timer 2 is different from the other timers; usually it is used as the time base for the PWM mode. But only 8-bit can be used for timer register (TMR2). The timer2 will do the comparison between TMR2 and PR2. If match occurs, then the interrupt flag is set. (At Postscale: 1:1) See the schematic below to see how it works.



As we can see that TMR2 and PR2 both have 8 bit. And like the other timers, you should choose the internal clock source and set the prescale value. But the interrupt flag will be set when TMR2 and PR2 value match, instead of the overflow. The postscaler is used for changing the frequency of interrupt. If postscaler is 1:1, which means if, match happened, the flag will be set. Then interrupt will be called. If postscaler is 1:16, this means that when match happened 16 times, then flag will be set once. The frequency is 16 times smaller than the frequency at postscale value 1:1.

Example:

```
OpenTimer2( TIMER_INT_ON &  //Interrupt enabled
            T2_PS_1_4 &          //Set the Prescale Value: 1:4
            T2_POST_1_1          //Set Postscale Value: 1:1.
            );
```

Here set Timer2 frequency = Fosc/4 * Prescaler * Posescaler = (48MHz/4)*(1/4) *(1/1) = 3MHz.

Set the PR2 register, TMR2 will compare with this register every cycle. And when match occurs, then interrupt flag is set, TMR2 will be reset to 0x00 automatically.

PR2 = 150; // interrupt at every (150/ 3MHz) = 50us

Beware the value of PR2 is 8 bit, which means it is from 0 to 255.

For the interrupt service routine and priority settings, you can see the explanation before. For detailed using and examples, you can find them in the example Timer0, Timer1 and Timer2.

5. Table of the timers and interrupt times.

TIMERS	Register bits	Prescaler	Postscaler	Timer Frequency	Maximum interrupt time	Minimum interrupt time
Timer0	8 / 16	1:2	-	6MHz	1.398 s (16bit, Pre*:1:256)	167ns (16bit, Pre: 1:2)
		1:4	-	3MHz		
		1:8	-	1.5MHz		
		1:16	-	0.75MHz		
		1:32	-	375kHz	5.44ms (8bit, Pre: 1:256)	167ns (8bit, Pre: 1:2)
		1:64	-	187.5kHz		
		1:128	-	93.75kHz		
		1:256	-	46.875kHz		
Timer1	16	1:1	-	12 MHz	43.69ms (Pre: 1:8)	83.3ns (Pre: 1:1)
		1:2	-	6 MHz		
		1:4	-	3 MHz		
		1:8	-	1.5 MHz		
Timer2	8	1:1	1:1~1:16	12MHz***	5.44ms (Pre: 1:16 Post**:1:16)	83.3ns (Pre: 1:1, Post: 1:1)
		1:4		3MHz***		
		1:16		0.75MHz***		
Timer3	16	1:1	-	12MHz	43.69ms (Pre:1:8)	83.3ns (Pre: 1:1)
		1:2	-	6 MHz		
		1:4	-	3 MHz		
		1:8	-	1.5 MHz		

*Pre: prescale value
**Post: postscale value
***this value is calculated by choosing Postscaler value 1:1. When choosing other postscale value, multiply postscale to this value.

For detailed structure of these timers, please check the datasheet, and see the chapter about Timer 0, Timer 1, Timer 2 and Timer 3.
http://ww1.microchip.com/downloads/en/DeviceDoc/39632D.pdf

For explanations about the commands for setting the timers, see C18 compiler library timer function chapter.
http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Libraries_51297f.pdf