Informe 10
PIC Temporizador

Laboratorio de Arquitectura del Computador

Elaborado por:
Tomás Guzmán, 21615008

1 MARCO TEÓRICO

El PIC cuenta también con un conjunto de capacidades para implementar un contador, o un temporizador.

1.1 TMR0

Este registro permite al programador llevar contadores o temporizadores, dependiendo de como se configura el mismo haciendo uso de los registros INTCON y OPTION_REG.

TMR0 no tiene bits que representen funcionalidades especiales, simplemente es un registro de 8 bits dedicado a simular el transcurrir del tiempo. Como es de 8 bits, puede contar desde 0 hasta 255.

Este registro va aumentando por cada ciclo de reloj de PIC. Estos aumentos se definen en una magnitud llamada prescalar. Aquí es donde el registro OPTION_REG entra al escenario

1.2 OPTION_REG

Como se había estudiado anteriormente, OPTION_REG es un registro que ofrece amplias funciones en el PIC

OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

En esta experiencia nos van a interesar nuevos bits. Ya habíamos usado el bit 6, INTEDG, para manejar interrupciones de flanco de subida y de bajada en RB0/INT.

Ahora nos interesa:

- T0CS: TMR0 Clock Source Select bit. Cuando este es activado, dependerá de la transición en el RA4/T0CKI pin, correspondiente a PORTA
- PSA: Prescaler Assignment bit. Determina si el prescalar asignado en los próximos bits será para el temporizador perro guardián, o para el Timer0. Si esta desactivo será para el último y así se trabajará en esta oportunidad.
- PS<2:0>: Prescaler Rate Selec. Estos tres bits trabajan juntos, y en ellos se guarda el valor del prescalar.

Bit Value	TMR0 Rate
000	1 : 2
001	1 : 4
010	1 : 8
011	1 : 16
100	1 : 32
101	1 : 64
110	1 : 128
111	1 : 256

En esta experiencia utilizaremos el 111, es decir 1:256.

1.3 EL PRESCALAR

Es una magnitud utilizada para determinar el incremento por ciclo de reloj de TMR0 o del perro guardián (WDT, no utilizado en esta experiencia). En este caso nos interesa el TMR0.

Como se utiliza un cristal de cuarzo como oscilador, este PIC funcionará en una frecuencia base de 4 MHz.

Para cualquier experiencia, la ecuación general a usar es:

$$t = 4 \times T(256 - \text{TMR0}) \times \text{prescalar}$$

En donde t es el tiempo del temporizador, T es el período (el inverso de la frecuencia) y prescalar es la magnitud elegida en PS<2:0>

Depende del tiempo t deseado, se pueden obtener distintos valores para TMR0. Note que, estos valores pueden ser números decimales, pero no hay problema con trabajar con sus aproximaciones a enteros, los errores serán muy bajos.

1.4 INTCON

INTCON en este caso será útil para guardar si ocurre o no una interrupción de TMR0.

INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

El bit que nos interesa en esta experiencia es T0IF, TMR0 Overflow Interrupt Flag bit. Cuando este es 1 significa que ocurrió un overflow en TMR0 (pasó de 255 a 0). Si es 0 es que esto no ha ocurrido.

Es responsabilidad del programador apagar esta flag para poder escuchar a próximas interrupciones.

2 IMPLEMENTACIÓN

```
    LIST P=16F84

OPT     EQU 01H
TMR0    EQU 01H
STATUS  EQU 03H
PORTA   EQU 05H
PORTB   EQU 06H
TRISA   EQU 05H
TRISB   EQU 06H
INTCON  EQU 0BH

COUNT   EQU 10H

#DEFINE RA0     PORTA, 0

#DEFINE T0IF    INTCON, 2

#DEFINE BANK0   BCF STATUS, 5
#DEFINE BANK1   BSF STATUS, 5

    ORG 0
    GOTO INICIO

INICIO ORG 10
```

```
    BANK1
    CLRF    TRISA       ; Declarado PORTA como salida
    MOVLW   B'00000111'
    MOVWF   OPT         ; Se activan PS<2:0>, asignando el prescalar 1:256
 al timer
    BANK0

    MOVLW   B'10100000' ; Se activan interrupciones globales y la interrup
ción de timer
    MOVWF   INTCON

    MOVLW   D'217'
    MOVWF   TMR0        ; Se carga el TMR0 en 217
    MOVLW   D'100'
    MOVWF   COUNT       ; Se carga 100 en la variable de conteo

MAIN
    NOP
    NOP                 ; No hace nada, pero gasta un ciclo de reloj
    GOTO    MAIN        ; Nos mantenemos en este ciclo "idle"

    ORG     4
    GOTO    INTER       ; Saltar a INTER si hubo interrupción de TMR0

INTER ORG 50
    DECFSZ  COUNT, 1    ; Disminuye a COUNT en 1. Si COUNT = 0, salta la p
róxima línea
    GOTO    RESET_TMR0  ; Si COUNT != 0, entonces vamos a RESET_TMR0
    GOTO    SEG         ; Si COUNT = 0, entonces pasó un segundi

RESET_TMR0
    MOVLW   D'217'
    MOVWF   TMR0        ; Se reinicia TMR0 a 217
    BCF     T0IF        ; Se apaga el flag de overflow de TMR0
    RETFIE

SEG
    BTFSC   RA0         ; Se hace una prueba sobre RA0 (PORTA<0>)
    GOTO    OFF         ; Si RA0 = 1, saltamos a OFF
    BSF     RA0         ; Como RA0 = 0, entonces lo encendemos
    GOTO    RESET_CNT

OFF
```

```
    BCF     RA0         ; Apagamos RA0

RESET_CNT
    MOVLW   D'100'
    MOVWF   COUNT       ; Se reinicia el valor del contador
    GOTO    RESET_TMR0

END
```

En esta experiencia conectaremos un LED a RA0 (PORTA<0>) para determinar si el mismo está encendido. La idea es que esto ocurra cada segundo. Es decir, el LED pasará un segundo apagado y otro prendido.

Como se puede observar hay ciclo MAIN el cual básicamente es no realizar nada. Este ciclo solo espera que ocurra una interrupción de TMR0 para poder manejarla.

Recordemos que COUNT es 100 en el primer momento. Apenas entra a la primera interrupción (cada 10 ms) esta ya será 99, luego 98, luego 97 y así.

Note que hay un número muy particular en estas líneas:

```
    MOVLW   D'217'
    MOVWF   TMR0        ; Se carga el TMR0 en 217
```

Este 217, dada la configuración del prescalar (256), hará que TMR0 haga ciclos de 10 ms, esto se puede ver en la ecuación de la que se habló antes.

$$t = 4 \times 0.25\mu s\ (256 - 217) \times 256$$
$$t = 0.009984\ s \simeq 10\ ms$$

Ahora hay otro número que parece mágico, pero no lo es, que es el de COUNT

```
    MOVLW   D'100'
    MOVWF   COUNT       ; Se carga 100 en la variable de conteo
```

Este 100 se carga porque cada vez que TMR0 hace overflow (cada 10 ms, casi), entonces se le resta uno. Esta operación se hace 100 veces, cada 10 ms, por lo cual toma 1000 ms realizar el ciclo completo, es decir 1 segundo.

En INTER

```
INTER ORG 50
    DECFSZ  COUNT, 1    ; Disminuye a COUNT en 1. Si COUNT = 0, salta la p
róxima línea
    GOTO    RESET_TMR0  ; Si COUNT != 0, entonces vamos a RESET_TMR0
    GOTO    SEG         ; Si COUNT = 0, entonces pasó un segundi
```

Se maneja dos GOTO, el primero simplemente reconfigura a TMR0 para contar 10 ms, nuevamente. El segundo (GOTO SEG) ocurre cuando COUNT = 0 y esto significa que ocurrió un segundo.

En el ciclo SEG entonces se prende o se apaga RA0 según sea el caso. Esta operación se puede visualizar mediante un LED (o un Probe como será usado en Multisim).

2.1 Compilación

Este programa en extensión .asm fue compilado haciendo uso de MPASMWIN.

Este programa se incluyó en el PATH de Windows, por lo cual se puede ejecutar desde cualquier terminal.

Su sintaxis es sencilla, en este caso:

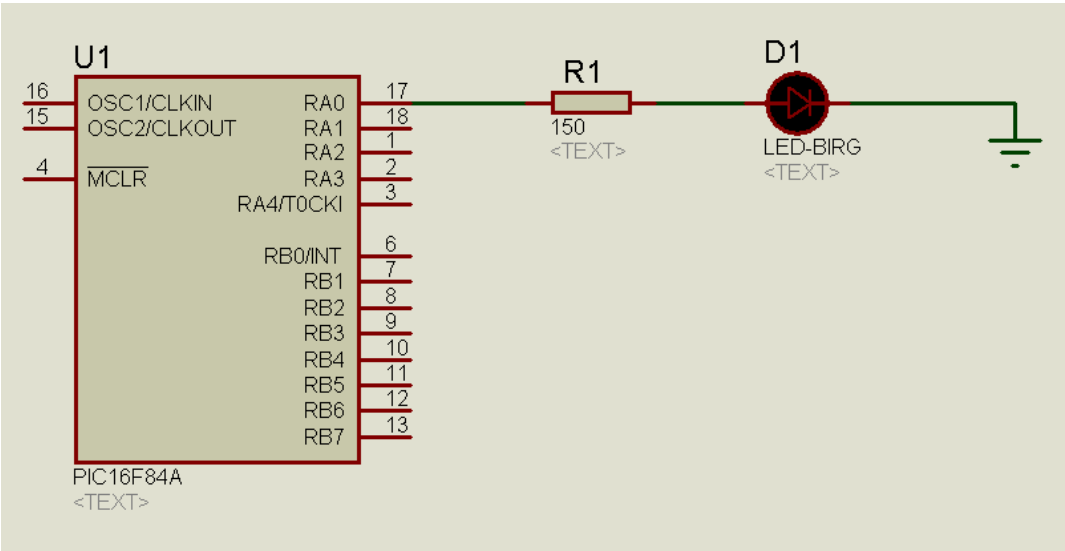
$ mpasmwin practica10.asm

Produce varios archivos, entre ellos el practica10.hex. Este último se usará en el PIC implementado en Multisim para visualizar la funcionalidad.

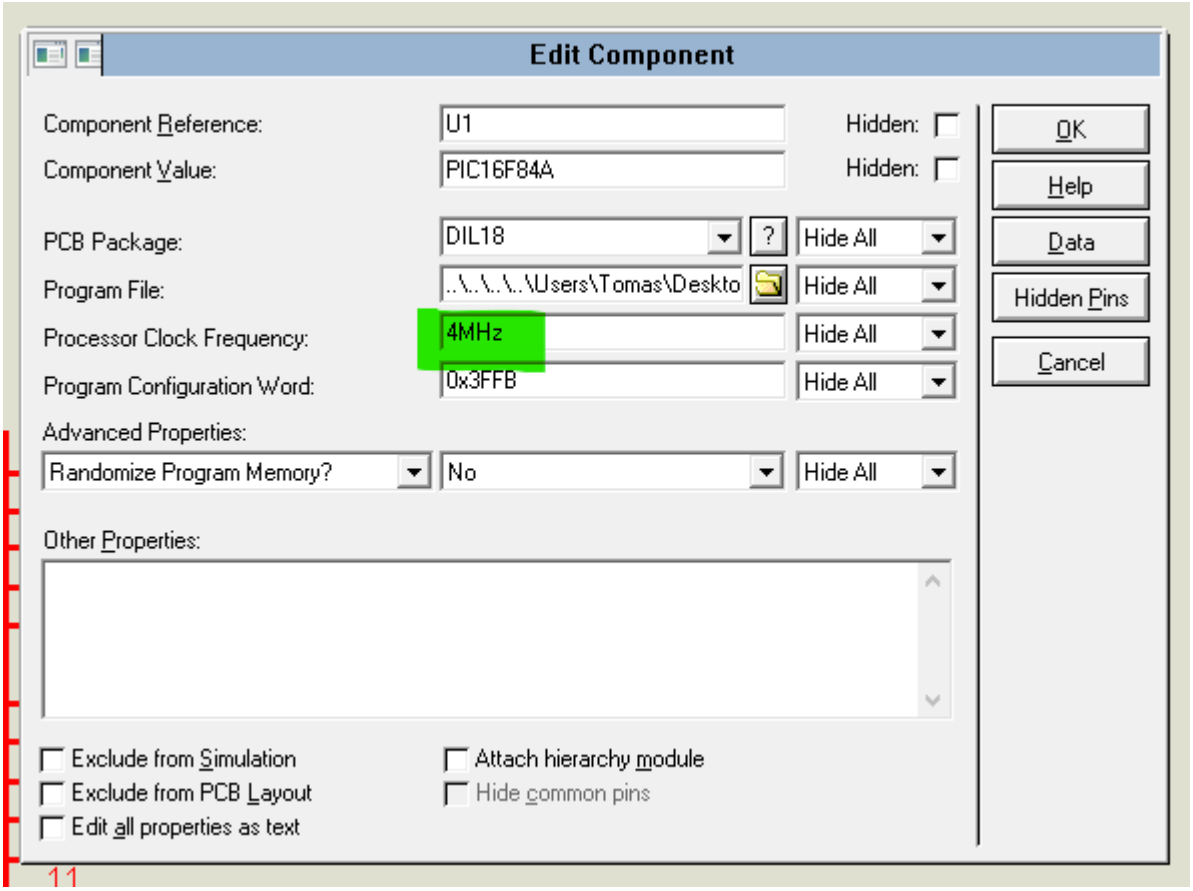
3 Simulación

Esta simulación resulta difícil de fotografiar, dado el fenómeno de Multisim de encender por intstantes los LEDs.

De esta forma se acudió a Proteus para poder realizar la misma. En este caso los resultados son sencillos de observar:

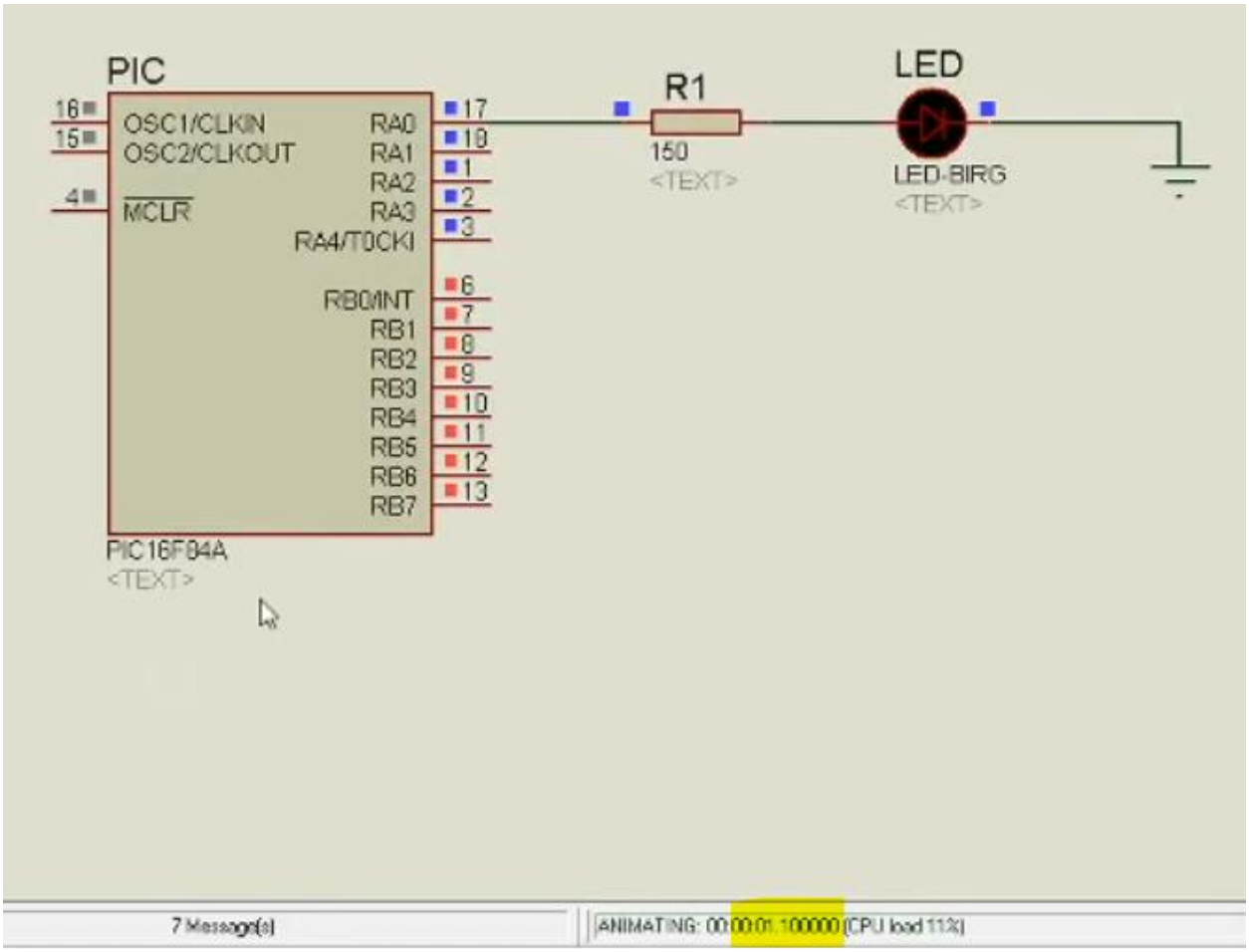


El circuito parece que estuviese muy sencillo, pero es exactamente el mismo. En Multisim se usaba un cristal de cuarzo para establecer los 4 MHz. En este caso, Proteus permite al programador utilizar configuración para poder lograr esta velocidad de ciclo de reloj:

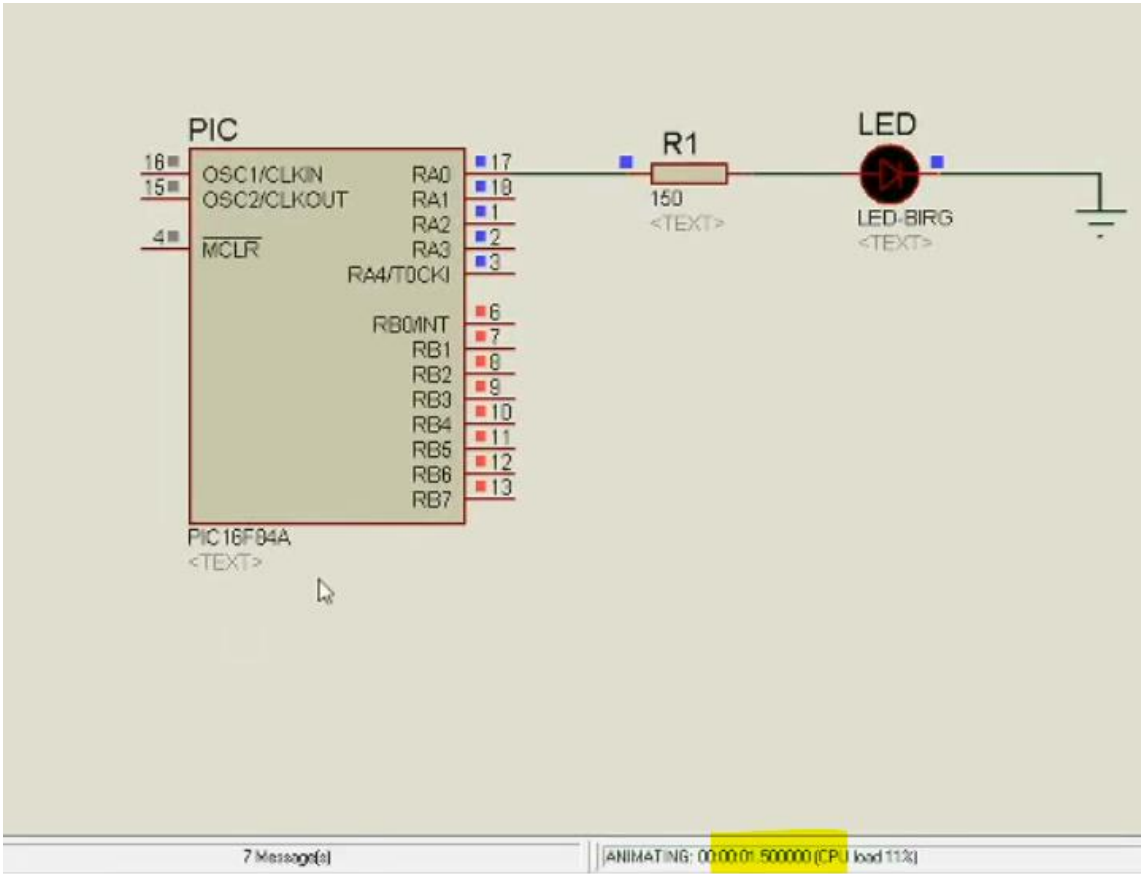


Por otro lado, los LEDs de Proteus son más fieles que aquellos de Multisim.

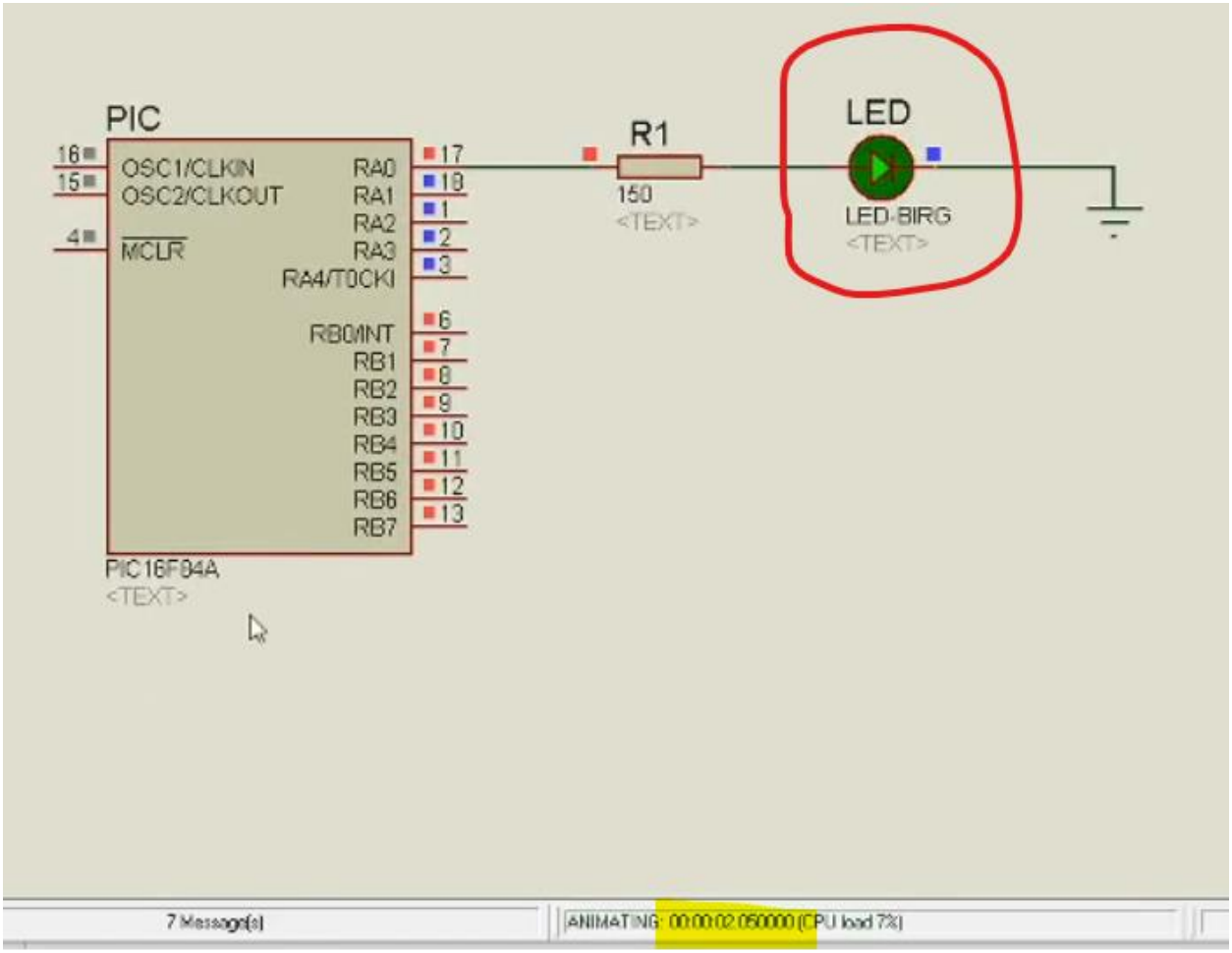
Se tomó un video de la pantalla de Windows para poder obtener las marcas de tiempo lo más fiel posible. Comenzando desde el segundo 1, podemos ver que el LED está apagado:



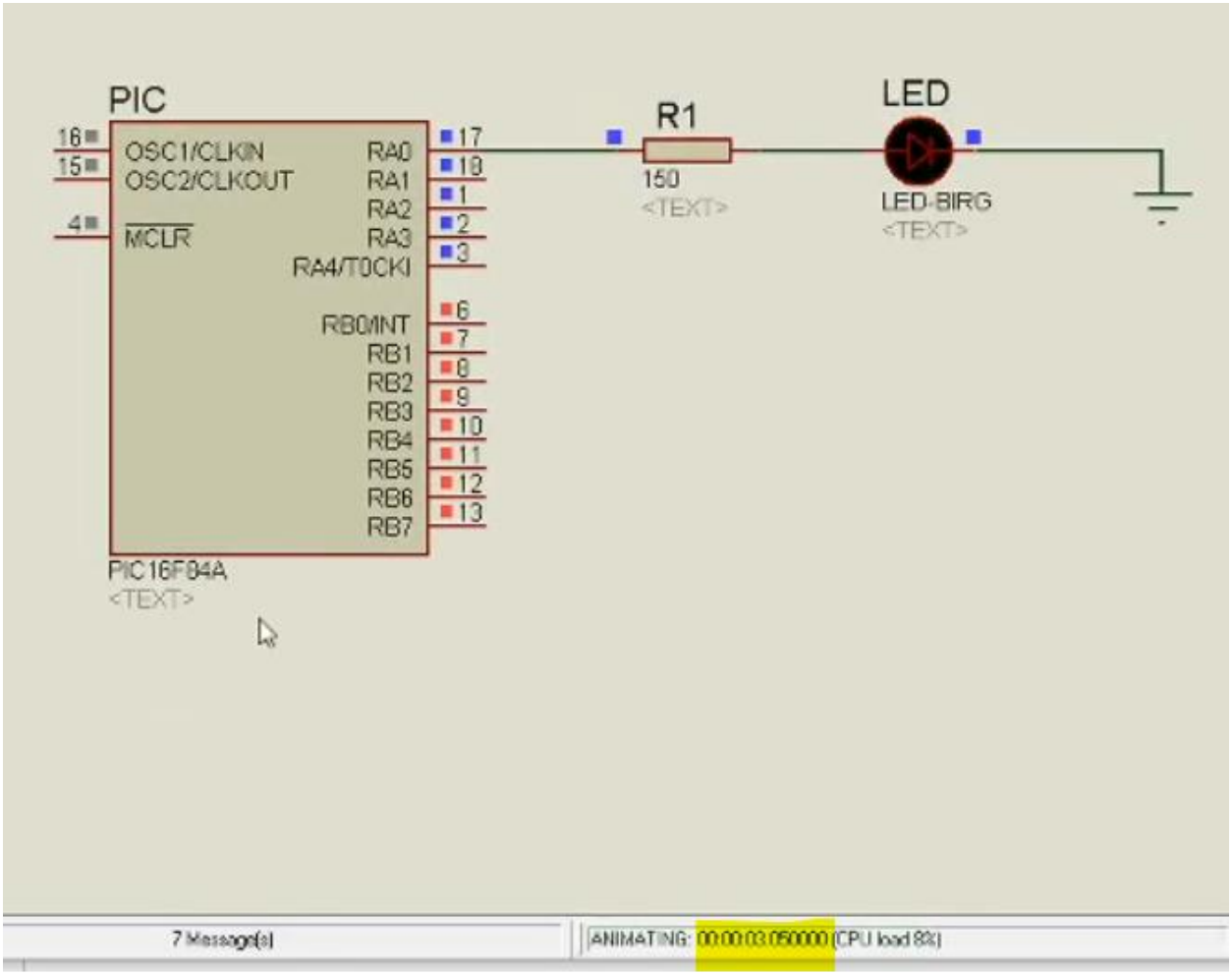
Y se aprecia como al segundo 1.5 sigue apagado (de no ser así habría un error en el código)



Ahora veamos al segundo 2. El LED está encendido



Y ya en el segundo tres está nuevamente apagado



Como se puede apreciar, es muy fiel el contador que se hizo.

4 CONCLUSIONES

- El PIC posee un módulo sofisticado para manejar temporizadores y contadores.
- El registro TMR0 es de 8 bits y puede contar desde 0 hasta 255, luego de esto produce un overflow.
- Este overflow es una interrupción y se maneja como tal.
- Con ayuda de variables adicionales se puede hacer que el PIC cuente distintos intervalos de tiempo. Por ejemplo, si se deja la configuración prescalar que tenemos actualmente y ubicamos un D'200' en COUNT, entonces contar dos segundos. Básicamente sería TMR0 * COUNT es el tiempo (en ms) que podríamos contar.