INFORME DE LABORATORIO 4: TEMPORIZADORES

Luis Miguel Rincon Pinilla
e-mail: lmrinconp@itc.edu.co
Cristian Camilo Pianda Rodríguez
e-mail: ccpiandar@itc.edu.co

RESUMEN: *En este informe se detalla los procedimientos seguidos y los resultados que se obtuvieron como parte del cuarto laboratorio con el entorno de programación MPLAB IDE. En el cual se adquirieron las destrezas necesarias para el uso de interrupciones externas usando el microcontrolador PIC18F4550 por medio de la modificación de sus registros de configuración.*

PALABRAS CLAVE: C++, Bandera, Compilador, Interrupción, Microcontrolador, Oscilador, Programación, Registro, Temporizador.

1 INTRODUCCIÓN

El medio de conexión para monitorear o controlar el hardware externo a los microcontroladores son las entradas y salidas analógicas o digitales con las que cuenta. Por lo cual es necesario comprender la manera en que se configuran cada uno de los puertos y registros con los que cuenta el PIC, especialmente los registros de configuración para las interrupciones.

Los módulos de Timer X llevan conteos generalmente de tiempo sin hacer uso de las funciones de delay lo cual ayuda a evadir ciclos muertos en los cuales el microcontrolador no reaccionara a ningún cambio ni ejecutara ninguna nueva instrucción.

Las interrupciones externas permiten atender eventos suspendiendo la ejecución del programa principal logrando dar prioridad a interacciones externas. Por otra parte, la conversión analógica-digital nos permite procesar en el microcontrolador las distintas señales analógicas provenientes del entorno a través de sensores.

2 OBJETIVO

Utilizar el módulo timer del PIC 18F4550 para realizar conteo de flancos y tiempo para la solución de diferentes problemas.

3 ELEMENTOS UTILIZADOS

Se Utilizaron los siguientes elementos durante el laboratorio:

- Displays de 7 segmentos de cátodo común. (Simulador)
- Fuente de voltaje D.C. (Simulador)
- Integrados 74LS48. (Simulador)
- LEDs. (Simulador)
- Microcontrolador PIC18F4550. (Simulador)
- MPLAB X IDE 5.40.
- Proteus 8.9.
- Pulsadores. (Simulador)
- Resistencias varias. (Simulador)

4 DESARROLLO

A continuación, se describe el proceso de diseño y programación del sistema que da solución a cada uno de los problemas planteados.

4.1 CRONÓMETRO

Para el primer sistema se requería desarrollar cronómetro que cuente minutos, segundos y décimas de segundo, utilizando los módulos de timer 1 para los minutos y el timer 0 para los segundos y décimas de segundo. El conteo de tiempo se debe visualizar en display 7 segmentos.

4.1.1 MONTAJE DEL CIRCUITO

Se realizó el montaje del circuito con un microcontrolador PIC18F4550 (*Fig. 1*) con las siguientes características:

- Se conectaron los dos pulsadores de Inicio/Pausa (**RB0**) y Reinicio (**RB1**) a las interrupciones externas **INT0** y **INT1**.
- Se conecto **RC2** a **RC0**.
- Para la conversión BCD a 7 segmentos se usaron cinco integrados 74LS48 conectados a los puertos A, B, C y D.
- Se conectaron cinco displays 7 segmentos uno a cada uno de los 74LS48.

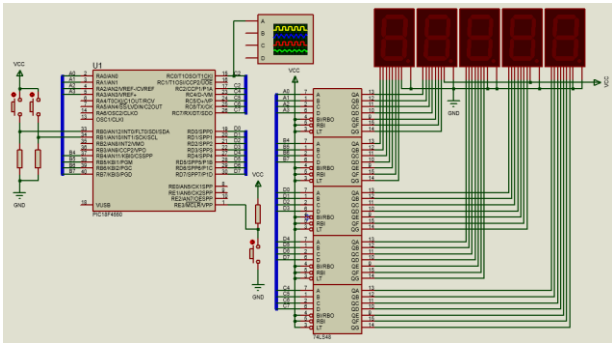

Figura 1. Circuito Cronómetro.

4.1.2 REGISTROS DE CONFIGURACIÓN

4.1.2.1 TIMER0

Para habilitar el Timer0 como temporizador fue necesario modificar el registro **T0CON**:

- T0CS=0. Elige el reloj de instrucciones como la fuente de reloj para el Timer0
- PSA=0. Habilita el uso de pre-escaler.

- T0PS=0b001. Usa preescaler 1:4
- T08BIT=0. Configura Timer0 como temporizador de 16 bits.
- TMR0ON=0. Inicializaremos el TMR0 apagado.

Para hallar el valor que se precargara en los registros de TMR0H y TMR0L se hicieron los siguientes cálculos:

$$F_{Ins} = \frac{8\ MHz}{4} = 2\ MHz$$

$$T_{Ins} = \frac{1}{2\ MHz} = 0{,}5\ \mu s$$

$$T_{Con} = 0{,}5\ \mu s * 4 * 65536 = 131{,}072\ ms$$

La unidad mínima que medirá el cronómetro es de 1 decima de segundo que es equivalente a 100ms lo que la convierte en el tiempo que se medirá con el Timer0.

$$TMRX = 65536 - \frac{100\ ms}{0{,}5\ \mu s * 4} = 15536$$

4.1.2.2 TIMER1

Para habilitar el Timer1 como contador fue necesario modificar el registro **T1CON**:

- RD16=0. Configura Timer1 como contador de 8 bits.
- T1RUN=0. El oscilador de TMR1 no se usará como reloj del dispositivo.
- T1CKPS=0. No se habilita pre-escaler para TMR1.
- T1OSCEN=0. Deshabilita el uso del oscilador del TMR1.
- T1SYNC=0. Sincroniza la entrada del reloj externo.
- TMR1CS=1. Configura RC0 como la fuente externa del reloj para TMR1.
- TMR1ON=1. Enciende el contador TMR1.

4.1.2.3 INTERRUPCIONES

Para habilitar el uso de interrupciones se modificaron los siguientes registros:

INTCON

- GIE. Habilita o deshabilita las interrupciones globales.
- INT0IF=0. Inicia la bandera de INT0 en 0.
- INT0IE=1. Habilita la interrupción INT0.
- PEIE=1. Habilita las interrupciones periféricas
- TMR0IF=0. Inicia la bandera de TMR0 en 0.
- TMR0IE=1. Habilita la interrupción de desbordamiento de TMR0.

INTCON2

- INTEDG0=1. INT0 como detector de flanco de subida.
- INTEDG1=1. INT1 como detector de flanco de subida.

INTCON3

- INT1IF=0. Inicia la bandera de INT1 en 0.
- INT1IE=1. Habilita la interrupción INT1.

Para las interrupciones periféricas se hicieron las siguientes modificaciones;

- PIR1bits.TMR1IF=0. Inicia la bandera de TMR1 en 0.
- PIE1bits.TMR1IE=1. Habilita la interrupción de desbordamiento de TMR1.

4.1.3 PROGRAMACIÓN

El código de programación en C que creamos para el microcontrolador es el siguiente:

```
#include <xc.h>
#pragma config FOSC = INTOSCIO_EC
int dec=0;
int sec=0;
int min=0;

void ExtInt_Init(void){
   INTCON2bits.INTEDG0=1;
   INTCON2bits.INTEDG1=1;
   INTCONbits.INT0IF=0;
   INTCONbits.INT0IE=1;
   INTCON3bits.INT1IF=0;
   INTCON3bits.INT1IE=1;
}

void Timer0_Init(void){
   INTCONbits.GIE=0;
   T0CONbits.T0CS=0;
   T0CONbits.PSA=0;
   T0CONbits.T0PS=0b001;
   T0CONbits.T08BIT=0;
   TMR0H=(15536)>>8;
   TMR0L=15536;
   T0CONbits.TMR0ON=0;
   INTCONbits.TMR0IF=0;
   INTCONbits.TMR0IE=1;
   INTCONbits.GIE=1;
}

void Timer1_Init(void){
   INTCONbits.PEIE=1;
   PIR1bits.TMR1IF=0;
   PIE1bits.TMR1IE=1;
   T1CONbits.RD16=0;
   T1CONbits.T1RUN=0;
   T1CONbits.T1CKPS=0;
   T1CONbits.T1OSCEN=0;
   T1CONbits.T1SYNC=0;
   T1CONbits.TMR1CS=1;
   T1CONbits.TMR1ON=1;
   TMR1L=0;
}

void Port_Init(void){
   OSCCONbits.IRCF=0b111;
   ADCON1=0b00001111;
   TRISA=0;
   TRISB=0b00000011;
   TRISC=0b00000001;
   TRISD=0;
   PORTA=0;
```

```
    PORTB=0;
    PORTC=0;
    PORTCbits.RC2=1;
    PORTD=0;
}

void main(void) {
    Timer0_Init();
    Timer1_Init();
    ExtInt_Init();
    Port_Init();
    while(1){
        PORTA=(TMR1L%10);
        PORTB=((TMR1L/10)*16);
        PORTC=(dec)<<4;
        PORTD=((sec/10)*16)+(sec%10);
    }
    return;
}

void __interrupt()ISR(void){
    if(INTCONbits.TMR0IF==1){
        dec++;
        INTCONbits.TMR0IF=0;
        TMR0H=(15536)>>8;
        TMR0L=15536;
        if(dec==10){
            PORTCbits.RC2=0;
            dec=0;
            sec++;
        }
        if(sec==60){
            PORTCbits.RC2=1;
            sec=0;
        }
    }
    if(TMR1L==100){
        TMR1L=0;
    }
    if(INTCONbits.INT0IF==1){
        T0CONbits.TMR0ON=!T0CONbits.TMR0ON;
        INTCONbits.INT0IF=0;
    }
    if(INTCON3bits.INT1IF==1){
        T0CONbits.TMR0ON=0;
        sec=0;
        dec=0;
        TMR1L=0;
        INTCON3bits.INT1IF=0;
    }
}
```

4.2 CONTADOR DE AFORO C.C.

Para el segundo sistema se requería desarrollar un contador ascendente/descendente para medir el aforo de un centro comercial con el fin de cumplir con los protocolos de cuarentena. En el centro comercial se han dispuesto sensores (2) en las entradas y salidas del centro comercial, los cuales envían un flanco de subida cuando una persona entra o sale del establecimiento. Utilice el módulo timer para realizar el conteo de personas de tal manera que siempre se tenga conocimiento del aforo y que esta información se pueda visualizar en un display 7 segmentos. Si el aforo de 30 personas se supera un led rojo se debe encender como señal de alarma.

4.2.1 MONTAJE DEL CIRCUITO

Se realizó el montaje del circuito con un microcontrolador PIC18F4550 (*Fig. 2*) con las siguientes características:

- Se conectaron los dos pulsadores para simular la entrada (**RB0**) y salida (**RB1**) de personas a las interrupciones externas **INT0** y **INT1**.
- Se conecto **RC2** a **RC0**.
- Para la conversión BCD a 7 segmentos se usaron dos integrados 74LS48 conectados al puerto D.
- Se conectaron dos displays 7 segmentos uno a cada uno de los 74LS48.

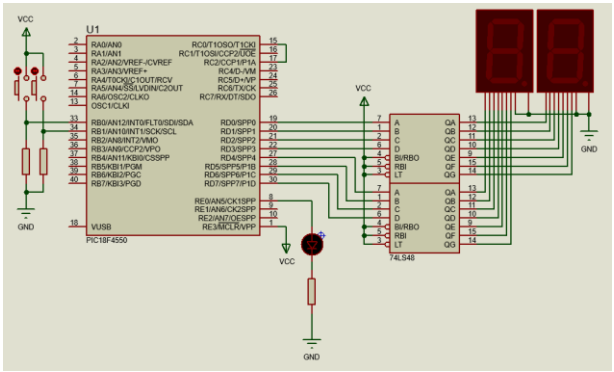

Figura 2. Circuito contador de aforo.

4.2.2 REGISTROS DE CONFIGURACIÓN

Se usaron las mismas configuraciones que en el ejercicio anterior con la excepción de que no se habilitó TMR0 ni sus interrupciones.

4.2.3 PROGRAMACIÓN

El código de programación en C que creamos para el microcontrolador es el siguiente:

```
#include <xc.h>
#pragma config FOSC = INTOSCIO_EC

void ExtInt_Init(void){
    INTCON2bits.INTEDG0=1;
    INTCON2bits.INTEDG1=1;
    INTCONbits.INT0IF=0;
    INTCONbits.INT0IE=1;
    INTCON3bits.INT1IF=0;
    INTCON3bits.INT1IE=1;
}

void Timer1_Init(void){
    INTCONbits.GIE=0;
    INTCONbits.PEIE=1;
    PIR1bits.TMR1IF=0;
    PIE1bits.TMR1IE=1;
```

```
    T1CONbits.RD16=0;
    T1CONbits.T1RUN=0;
    T1CONbits.T1CKPS=0;
    T1CONbits.T1OSCEN=0;
    T1CONbits.T1SYNC=0;
    T1CONbits.TMR1CS=1;
    T1CONbits.TMR1ON=1;
    TMR1L=0;
    INTCONbits.GIE=1;
}

void Port_Init(void){
    OSCCONbits.IRCF=0b111;
    ADCON1=0b00001111;
    TRISB=0b00000001;
    TRISC=0b00000001;
    TRISD=0;
    TRISEbits.RE0=0;
    PORTCbits.RC2=1;
    PORTB=0;
    PORTC=0;
    PORTD=0;
    PORTEbits.RE0=0;
}

void main(void) {
    Timer1_Init();
    ExtInt_Init();
    Port_Init();

    while(1){
        PORTCbits.RC2=0;
        PORTD=((TMR1L/10)*16)+(TMR1L%10);
        if(TMR1L>30){
        PORTEbits.RE0=1;
        }
        else{
        PORTEbits.RE0=0;
        }
    }
    return;
}

void __interrupt()ISR(void){
    if(INTCONbits.INT0IF==1){
        PORTCbits.RC2=1;
        INTCONbits.INT0IF=0;
    }
    if(INTCON3bits.INT1IF==1){
        if(TMR1L>0){
            TMR1L--;
        }
        INTCON3bits.INT1IF=0;
    }
}
```

4.3 GENERADOR DE SEÑAL CUADRADA

Para el tercer sistema se requería desarrollar un generador de señal cuadrada de 100ms de período con ciclo útil de:

- 50%.
- 80%.

4.3.1 MONTAJE DEL CIRCUITO

Se realizó el montaje del circuito con un microcontrolador PIC18F4550 (*Fig. 3*) con las siguientes características:

- Se conectaron los dos LEDs y las puntas del osciloscopio a los pines **RD0** (Señal 50%) y **RD1** (Señal 80%).

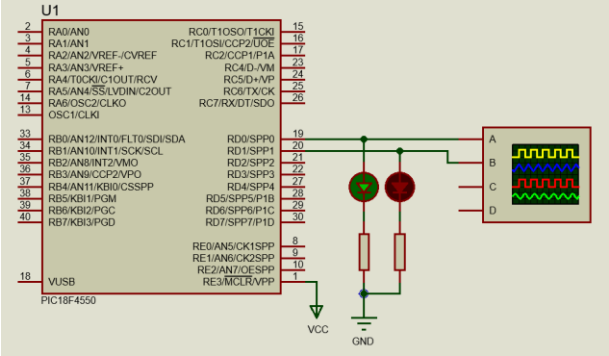

Figura 3. Circuito generador de señal cuadrada.

4.3.2 REGISTROS DE CONFIGURACIÓN

Se usaron las mismas configuraciones que en el ejercicio uno con la excepción de que no se habilitaron interrupciones externas ni el TMR1, además de hicieron las siguientes modificaciones a TMR0.

- T0CONbits.PSA=1. Se deshabilita el uso de pre-escaler.

Para hallar el valor que se precargara en los registros de TMR0H y TMR0L se hicieron los siguientes cálculos:

$$F_{Ins} = \frac{8\ MHz}{4} = 2\ MHz$$

$$T_{Ins} = \frac{1}{2\ MHz} = 0{,}5\ \mu s$$

$$T_{Con} = 0{,}5\ \mu s * 1 * 65536 = 32{,}768\ ms$$

La unidad mínima que se medirá es de 10 ms lo que la convierte en el tiempo que se medirá con el Timer0.

$$TMRX = 65536 - \frac{10\ ms}{0{,}5\ \mu s * 1} = 45536$$

4.3.3 PROGRAMACIÓN

El código de programación en C que creamos para el microcontrolador es el siguiente:

```
#include <xc.h>
#pragma config FOSC = INTOSCIO_EC
int time=0;

void Timer0_Init(void){
    INTCONbits.GIE=0;
    T0CONbits.T0CS=0;
    T0CONbits.PSA=1;
    T0CONbits.T08BIT=0;
    TMR0H=(45536)>>8;
```

```
    TMR0L=45536;
    T0CONbits.TMR0ON=1;
    INTCONbits.TMR0IF=0;
    INTCONbits.TMR0IE=1;
    INTCONbits.GIE=1;
}

void main(void) {
    OSCCONbits.IRCF=0b111;
    TRISDbits.RD0=0;
    TRISDbits.RD1=0;
    PORTDbits.RD0=0;
    PORTDbits.RD1=0;
    Timer0_Init();

    while(1){
        if(time==0){
            PORTDbits.RD0=1;
            PORTDbits.RD1=1;
        }
        if(time==5){
            PORTDbits.RD1=0;
        }
        if(time==8){
            PORTDbits.RD0=0;
        }
        if(time>=10){
            time=0;
        }
    }
    return;
}

void __interrupt()ISR(void){
    if(INTCONbits.TMR0IF==1){
        time++;
        INTCONbits.TMR0IF=0;
        TMR0H=(45536)>>8;
        TMR0L=45536;
    }
}
```

4.4 SEMÁFORO CON PRIORIDAD DE PEATÓN

Para el cuarto sistema se requería desarrollar un programa que ejecute la rutina del semáforo de un cruce peatonal

- Al iniciarse el sistema los semáforos SA y SB, se encuentran en modo de operación normal: la luz roja se enciende durante un segundo con el amarillo apagado, luego el amarillo se enciende durante 1 segundo con el rojo apagado y se repite esta secuencia.
- SA y SB funcionan al mismo tiempo.
- Cuando un peatón vaya a cruzar la calle debe presionar el botón de paso (por interrupciones) y de forma inmediata los semáforos deben activar la luz roja durante 15 segundos y seguidamente volver al modo de operación normal.
- Utilice los módulos de timer que considere necesarios. Todo conteo de tiempo debe realizarse por medio de estos módulos

4.4.1 MONTAJE DEL CIRCUITO

Se realizó el montaje del circuito con un microcontrolador PIC18F4550 (*Fig. 4*) con las siguientes características:

- Se conectaron los dos pulsadores de peatón **RB0** y **RB1** a las interrupciones externas **INT0** y **INT1**.
- Se conectaron los LEDs rojos de los semáforos a **RB2** y **RB4**. Los LEDs amarillos a **RB3** y **RB5**.
- Para la conversión BCD a 7 segmentos se usaron dos integrados 74LS48 conectados al puerto D.
- Se conectaron dos displays 7 segmentos uno a cada uno de los 74LS48.

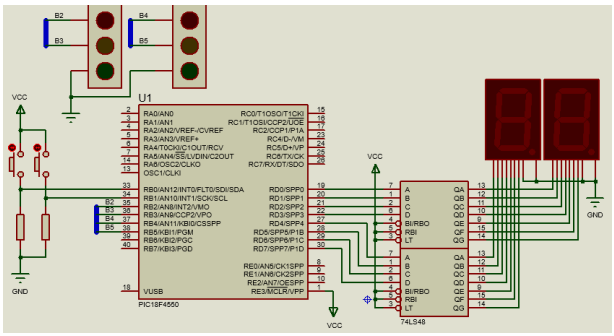

Figura 4. Circuito semáforo con prioridad de peatón.

4.4.2 REGISTROS DE CONFIGURACIÓN

Se usaron las mismas configuraciones que en el ejercicio uno con la excepción de que no se habilito el TMR1, además de hicieron las siguientes modificaciones a TMR0.

- T0PS=0b001. Usa pre-escaler 1:32.

Para hallar el valor que se precargara en los registros de TMR0H y TMR0L se hicieron los siguientes cálculos:

$$F_{Ins} = \frac{8\ MHz}{4} = 2\ MHz$$

$$T_{Ins} = \frac{1}{2\ MHz} = 0{,}5\ \mu s$$

$$T_{Con} = 0{,}5\ \mu s * 32 * 65536 = 1{,}048576\ s$$

La unidad mínima que se medirá es de 1 s lo que la convierte en el tiempo que se medirá con el Timer0.

$$TMRX = 65536 - \frac{1\ s}{0{,}5\ \mu s * 32} = 3036$$

4.4.3 PROGRAMACIÓN

El código de programación en C que creamos para el microcontrolador es el siguiente:

```
#include <xc.h>
#pragma config FOSC = INTOSCIO_EC
int sec=0;
int flag=0;

void ExtInt_Init(void){
    INTCON2bits.INTEDG0=1;
    INTCON2bits.INTEDG1=1;
    INTCONbits.INT0IF=0;
    INTCONbits.INT0IE=1;
    INTCON3bits.INT1IF=0;
    INTCON3bits.INT1IE=1;
}

void Timer0_Init(void){
    INTCONbits.GIE=0;
    T0CONbits.T0CS=0;
    T0CONbits.PSA=0;
    T0CONbits.T0PS=0b100;
    T0CONbits.T08BIT=0;
    TMR0H=(3036)>>8;
    TMR0L=3036;
    T0CONbits.TMR0ON=1;
    INTCONbits.TMR0IF=0;
    INTCONbits.TMR0IE=1;
    INTCONbits.GIE=1;
}

void Port_Init(void){
    OSCCONbits.IRCF=0b111;
    ADCON1=0b00001111;
    TRISB=0b00000011;
    TRISD=0;
    PORTB=0;
    PORTD=0;
}

void main(void) {
    Timer0_Init();
    ExtInt_Init();
    Port_Init();
    while(1){
        PORTD=((sec/10)*16)+(sec%10);
        if(flag==0){
            if(sec==0){
                PORTBbits.RB3=0;
                PORTBbits.RB5=0;
                PORTBbits.RB2=1;
                PORTBbits.RB4=1;
            }
            if(sec==1){
                PORTBbits.RB2=0;
                PORTBbits.RB4=0;
                PORTBbits.RB3=1;
                PORTBbits.RB5=1;
            }
            if(sec>=2){
                sec=0;
            }
        }
        else{
            if(sec==0){
                PORTBbits.RB3=0;
                PORTBbits.RB5=0;
                PORTBbits.RB2=1;
                PORTBbits.RB4=1;
            }
            if(sec==15){
                sec=0;
                flag=0;
            }
        }
    }
    return;
}

void __interrupt()ISR(void){
    if(INTCONbits.TMR0IF==1){
        sec++;
        INTCONbits.TMR0IF=0;
        TMR0H=(3036)>>8;
        TMR0L=3036;
    }
    if(INTCONbits.INT0IF==1||INTCON3bits.INT1IF==1){
        flag=1;
        INTCONbits.INT0IF=0;
        INTCON3bits.INT1IF=0;
        sec=0;
    }
}
```

5 Enlace del Video

El video de la simulación se encuentra en: https://youtu.be/OKATstV5Xo8

6 CONCLUSIONES

Lo expuesto en este informe de laboratorio permite arribar a la siguiente conclusión:

- Durante la ejecución del laboratorio se puso en práctica la programación modular por medio de funciones lo cual facilito reutilizar secciones de uno de los programas en otros por medio de pequeñas adaptaciones.
- Junto con los módulos de Timer se pueden utilizar otros registros como los de interrupción externa (**INTCON:INTCON3**), permitiendo adquirir información obtenida del ambiente y dar respuesta en el programa de acuerdo a las condiciones dadas.
- En el proceso de simulación se logró observar que los Módulos de Timer son herramientas de alta precisión para la medición de tiempo, esta precisión se puede variar dependiendo del uso del pre-escaler según los requerimientos del sistema.