PROGRAMACIÓN DE MICROCONTROLADORES PIC CON LENGUAJE C

Tomo 2

PROGRAMACIÓN DE MÓDULOS: TEMPORIZADORES, ADC, CCP Y DE COMUNICACIONES

Sixto Reinoso V.
Marco Pilatasig
Luis Mena
Jorge Sánchez

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Programación de microcontroladores PIC con Lenguaje C Tomo II
Programación de módulos: temporizadores, ADC, CCP Y DE comunicaciones
Sixto Reinoso V; Marco Pilatasig; Luis Mena y Jorge Sánchez

Primera edición electrónica. Octubre de 2018
ISBN: 978-9942-765-37-6
Revisión científica: Juan Pablo Pallo Noroña y Julio Enrique Cuji Rodriguez

Universidad de las Fuerzas Armadas ESPE
Crnl. Ing. Ramiro Pazmiño O.
Rector

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas ESPE
Cpnv. Hugo Pérez
Presidente

Edición y producción
David Andrade Aguirre
daa06@yahoo.es

Diseño
Pablo Zavala A.

Universidad de las Fuerzas Armadas ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador.
htpp//www.espe.edu.ec

PROGRAMACIÓN DE MICROCONTROLADORES PIC CON LENGUAJE C

Tomo II
PROGRAMACIÓN DE MÓDULOS: TEMPORIZADORES, ADC, CCP Y DE COMUNICACIONES

Sixto Reinoso V.
Marco Pilatasig
Luis Mena
Jorge Sánchez

INTRODUCCIÓN

En la segunda parte de la serie Programación de Microcontroladores PIC con lenguaje C, los temas más destacados son: el Manejo de teclados matriciales mediante el módulo ADC, manejo y aplicaciones con temporizadores e interrupciones externas, el uso del módulo CCP para comparador, captura y PWM. El uso del ADC para la lectura y procesamiento de señales analógicas es otro tema destacado del texto. Además se incluye el manejo del módulo CCP mejorado (ECCP) para aplicaciones en control de potencia con circuitos inversores. Se abarcan las comunicaciones más populares como por ejemplo USART, SPI, I2C. Se realiza una aplicación especial con el módulo USB para adquisición de datos con PIC y LabVIEW. En este caso se da una amplia explicación para utilizar el Virtual USB que es otra de las utilidades que presenta ISIS en sus últimas versiones.

Los microcontroladores PIC se utilizan en aplicaciones, en las que la variable tiempo es un parámetro muy importante, como por ejemplo, para generar señales a una frecuencia predeterminada, medir la duración de una señal, entre otras. En el capítulo 1, se analiza la estructura, funcionamiento y programación de cada uno de los módulos temporizadores que dispone el PIC18F4550.

El microcontrolador PIC18F4550 dispone de 13 canales ADC, los mismos que tiene una resolución de 10 bits, la función de los ADC es simplemente convertir los niveles de tensión analógicos (0v a 5V), que ingresan a un pin del PIC provenientes de distintos sensores y convertirlos en códigos binarios para que puedan ser procesados internamente. El proceso de conversión de una señal analógica a digital, se lo realiza mediante un módulo muestreador, cuantificador y codificador, de tal forma que para un cierto valor analógico le corresponde un respectivo código digital. El capítulo 2, describe el uso y aplicaciones del módulo CCP y ECCP.

Cuando se utilizan los módulos CCP, se puede realizar tres funciones básicas que se basan en la utilización de los temporizadores:

- Comparador: Permite realizar la comparación entre el valor que tiene en temporizador y el valor que tiene un registro, para ejecutar alguna acción en el microcontrolador.

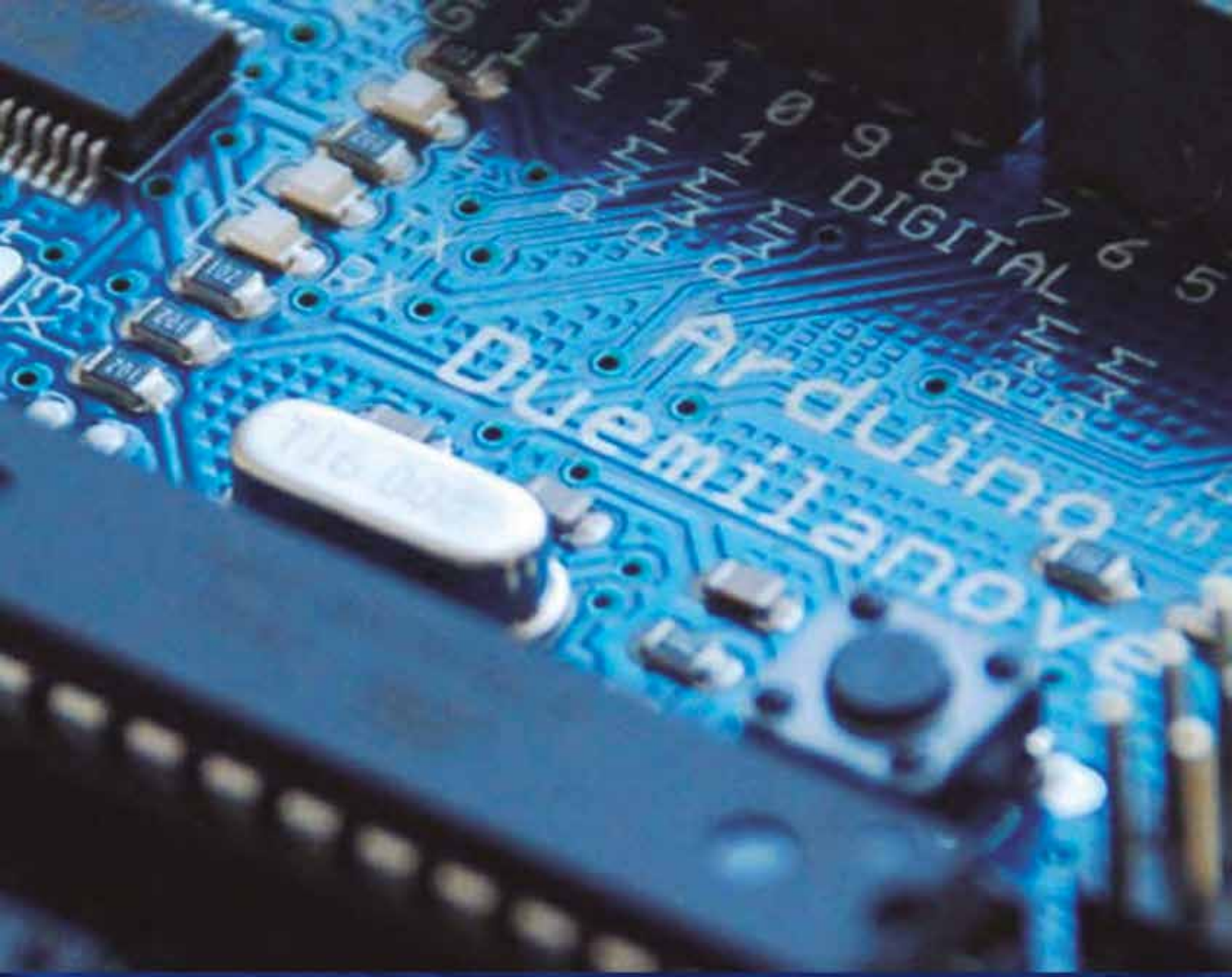
- Captura: Obtiene el valor del temporizador en un tiempo específico, determinado por la acción de un terminal del PIC.
- PWM: Se utiliza para generar señales moduladas por ancho de pulso, las mismas que son utilizadas para controlar la velocidad de motores DC.

El microcontrolador PIC18F4550 posee diferentes canales de comunicaciones que soportan varios protocolos. En el capítulo 3, se estudia el uso y la programación de los canales de comunicaciones más usados y su interfaz con otros dispositivos.

Los motores DC, servomotores y paso a paso, son muy utilizados en aplicaciones como la robótica, máquinas industriales, debido a su precisión en el movimiento ya que trabajan con señales digitales. En el capítulo 4 se realizan aplicaciones con este tipo de motores.

En cada tema tratado se incluyen los programas, diagramas de conexiones y los resultados obtenidos. Para completar el proceso de aprendizaje al final de cada capítulo se insertan más ejercicios de aplicación resueltos y se proponen otros para reforzar los conocimientos adquiridos en el capítulo.

Se debe mencionar que toda la información incluida en el texto en referencia a las instrucciones del compilador ha sido tomada del manual CCS y de las hojas técnicas de los microcontroladores utilizados de Microchip Technology Inc., los mismos que se pueden descargar gratuitamente de los sitios webs oficiales de los fabricantes citados.



CAPÍTULO 1

TEMPORIZADORES

El PIC18F4550, posee cuatro módulos temporizadores de 8 y/o 16 bits, los mismos que pueden trabajar en modo temporizador o en modo contador. El Temporizador trabaja en modo "*temporizador*" cuando cuenta los pulsos del reloj del sistema y en modo contador cuando cuenta los pulsos de una fuente externa.

Los temporizadores se designan:

- Temporizador TMR0.
- Temporizador TMR1.
- Temporizador TMR2.
- Temporizador TMR3.

TEMPORIZADOR 0

El módulo TMR0 temporizador/contador incorpora las siguientes características:

- Configurable como temporizador o contador de 8/16 bits.
- Escritura y lectura en los registros.
- Pre – escalar de 8 bits.
- Seleccionable la fuente de reloj interna o externa.
- Seleccionable el flanco de la señal de reloj externa.
- Interrupción por desbordamiento.

El registro T0CON controla todos los aspectos de la operación del módulo TMR0.

T0CON: TIMER0 CONTROL REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Tabla 1.1. Registro T0CON. Cortesía Microchip Technology INC.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

TMR0ON: Puesta en marcha del temporizador 0.

- 0= Temporizador apagado.
- 1= Temporizador activado.

T08BIT: Bit de control: Timer0 8-Bit/16-Bit.

- 1 = Timer0 se configure como un temporizador o contador de 8bits.
- 0 = Timer0 se configure como un temporizador o contador de 16 bits.

T0CS: Timer0 Clock Source Select bit. Bit de selección de la fuente de reloj.

- 1 = Transición en el pin T0CKI.
- 0 = Transición interna en los ciclos de reloj (CLKO).

T0SE: Timer0 Source Edge Select bit. Bit de selección del flanco.

- 1 = Incrementa en la transición de alto a bajo (H_T_L) en el pin T0CKI.
- 0 = Incrementa en la transición de bajo a alto (L_T_H) en el pin T0CKI.

PSA: Timer0 Prescaler Assignment bit. Bit de asignación del pre-escalar.

- 1 = Pre-escalar no se asigna al Timer0.
- 0 = Pre-escalar es asignado al Timer0.

T0PS2:T0PS0: Timer0 Prescaler Select bits. Bits de selección del pre-escalar.

El divisor de frecuencia trabaja según la tabla 1.2.

T0PS2	T0PS1	T0PS0	VALOR DEL PRE-ESCALAR
1	1	1	1:256
1	1	0	1:128
1	0	1	1:64
1	0	0	1:32
0	1	1	1:16
0	1	0	1:8
0	0	1	1:4
0	0	0	1:2

Tabla 1.2. Pre escalar del TMR0.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

La figura 1.1, muestra el diagrama de bloques del Temporizador 0 para operación con 8 bits.

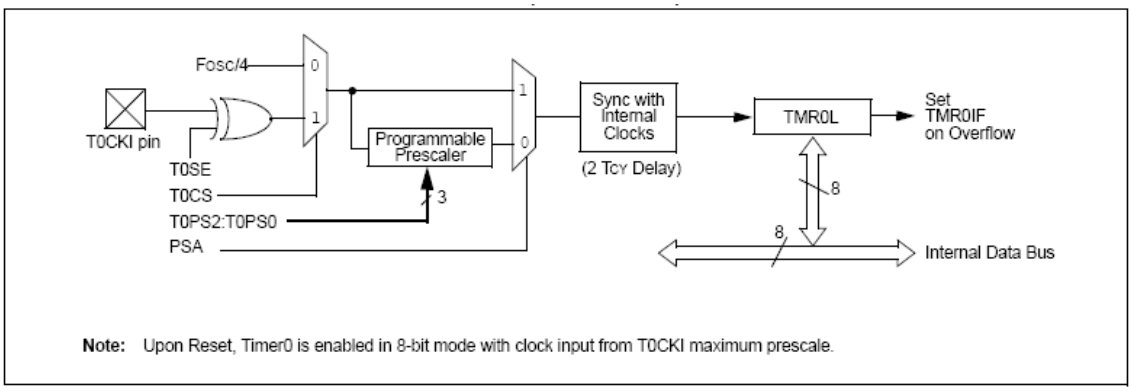


Figura 1.1. Diagrama de bloques TMR0 para 8 bits.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

La figura 1.2, muestra el diagrama de bloques del Temporizador 0 para operación con 16 bits.

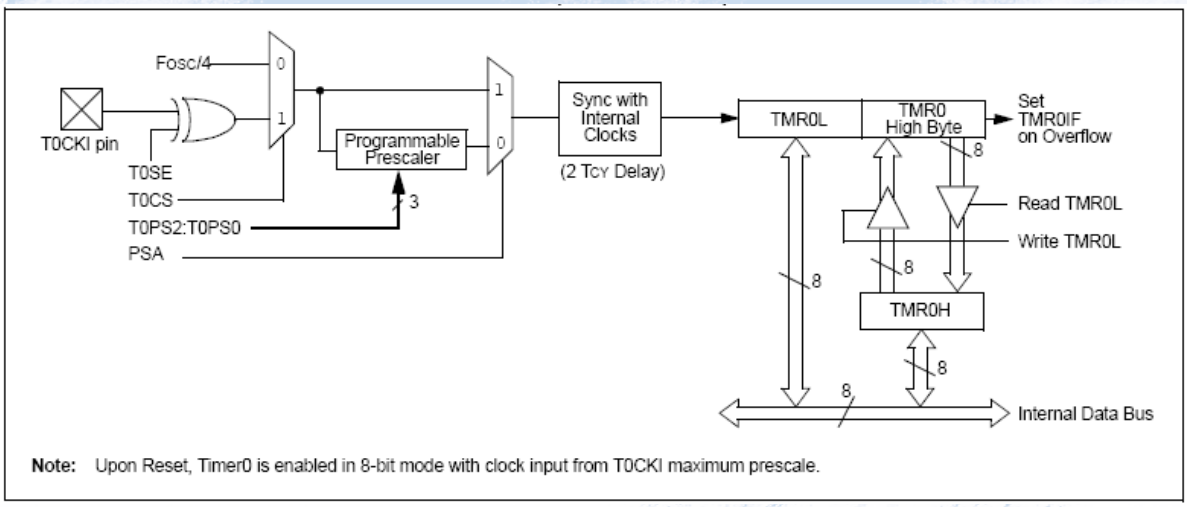


Figura 1.2. Diagrama de bloques TMR0 para 16 bits.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Funciones y directivas del compilador CCS para el módulo TMR0

El compilador CCS proporciona una gran variedad de funciones e instrucciones que permiten manejar las interrupciones a través de los temporizadores. Las más importantes son:

setup_timer_0(mode). Establece el modo de trabajo del TMR0 (temporizador/contador), el pre-escalar, el tamaño del registro (8 o 16 bits).

Dónde:

mode puede ser una o dos constantes definidas en el archivo .h:

- RTCC_INTERNAL, RTCC_EXT_L_TO_H o RTCC_EXT_H_TO_L
- RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256
  Sólo en el PIC18XXX: RTCC_OFF, RTCC_8_BIT

Ejemplo:

setup_timer_0(rtcc_div_256|rtcc_internal|rtcc_8_bit). Esta función sirve para fijar el pre escalar a 256, TMR0 CLK Interno con 8 bits.

setup_counters (rtcc_state, ps_state). Con esta función también se puede fijar los valores del TMR0. Ejemplo:

setup_counters(rtcc_internal|rtcc_8_bit,rtcc_div_256).

Otra forma de fijar los valores en el timer0.

Dónde:

rtcc_state puede ser uno de las constantes del dispositivo definidos en el archive .h. Por ejemplo:

RTCC_INTERNAL, RTCC_EXT_L_TO_H or RTCC_EXT_H_TO_L.

ps_state puede ser uno de las constantes del dispositivo definidos en el archive .h. Por ejemplo:

RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256, WDT_18MS, WDT_36MS, WDT_72MS, WDT_144MS, WDT_288MS, WDT_576MS, WDT_1152MS, WDT_2304MS.

set_timer0(valor). Fija el valor inicial del TMR0 puede ser de 8 o 16 bits.

set_rtcc(valor). Fija el valor inicial del TMR0 puede ser de 8 o 16 bits.

value = get_timer0(). Devuelve el valor del timer0, CLK/contador.

enable_interrupts(int_timer0). Habilita la interrupción del temporizador 0.

Uso del timer0 en modo temporizador

El temporizador TMR0 al usar el registro 8 bits, puede contar 256 pulsos o con 16 bits hasta 65536. Cada pulso es un ciclo de máquina del oscilador

de reloj, es decir, si el divisor de frecuencia (pre-escalar) está deshabilitado, el registro se incrementa cada 4 periodos de la señal de reloj.

El tiempo del temporizador se calcula:

Tiempo = 4* TOSC * (256-TMR0L) * Rango Divisor de Frecuencia (para 8 bits).

Tiempo = 4* TOSC * (65536-TMR0) * Rango Divisor de Frecuencia (para 16 bits).

Ejercicio 1.1. TMR0 en modo temporizador y 8 bits. Generar un pulso de 1s.

En el siguiente ejercicio se va a utilizar el temporizador 0 en modo de 8 bits para generar un retardo de tiempo de 1 s. Como se requiere un tiempo de 1 s y trabajando con un oscilador de 4 MHz, no se puede temporizar un tiempo tan alto, es necesario reducirlo a un menor valor y para luego mediante programación utilizando un temporizador completar el tiempo.

Con un oscilador de 4MHz, el máximo tiempo que se puede generar sería de:

El Periodo del oscilador TOSC = 1/ 4 MHz = 0.25 us.

Tiempo = 4 * TOSC * (256-TMR0L) * Rango Divisor de Frecuencia.

Tiempo = 4 * 0.25 us (256-0) *256

Tiempo = 65536 us = 65.536 ms.

Por tanto, para generar 1s, se repite 1000/65.536 = 15,26 veces la interrupción del TMR0.

Ahora bien, 0,26 veces no se puede repetir, por lo que se puede aproximar al inmediato superior (16) y calcular el valor que se debe cargar al TMR0 que en este caso ya no sería 0.

El nuevo tiempo que debe generar el TMR0, es:

1000/Tiempo (ms) = 16

Tiempo = 1000/16= 62.5 ms

Ahora se calcula el valor a cargar en el TMR0L.

62.5 ms = 4 * 0,25 us * (256-TMR0L) * 256

TMR0L = 256 - (62.5 ms / 0.256 ms) = 256 – 244.14

TMR0L = 12 (aproximadamente).

Se carga TMR0 = 12 (a partir de este valor el temporizador deberá contar 244 ciclos)

Divisor de frecuencia (pre-escalar) 256 (PS2, PS1, PS0 = 1 1 1).

Con estos datos se tiene un tiempo de:

Tiempo = 4 * 0.25 * (256 – 12) * 256 = 62 464 us.

Para completar 1 segundo se debe repetir 16 veces (62 464 us. * 16 = 999 424 us = 1 segundo).

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT, NOPROTECT, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
int contador;                           //Variable para el contador.
#use fast_io(d)                         //Define el Puerto D para respuesta rápida.

#int_Timer0                             //Interrupción int_Timer0.
void DesbordeTimer0()                   //Función interrupción.
{ contador = contador +1;               //contador se incrementa en 1.
 if (contador == 16){output_bit(PIN_D0,!input(PIN_D0));     //Cada 16 interrupciones cambia de valor
    contador=0;
{set_timer0(12);                        //Fija el TMR0 en 12.
{main(void)                             //Función principal main.
{contador =0;                           //Contador= 0,
  output_d(0x00);
  set_tris_d(0x00);
  setup_timer_0(rtcc_div_256|rtcc_internal|rtcc_8_bit);        //pre-escalar a 256,
                                        //TMR0 interno con 8 bits.
  set_timer0(12);                       //Fija el TMR0  en 12.
  enable_interrupts( int_timer0 );      //Habilita la interrupción TMR0.
  enable_interrupts( global );          //Habilitación global de las interrupciones.

 while(true) {                          //Bucle infinito
          // Aquí incluir instrucciones para el programa…
  }                                     //Fin del bucle.
}                                       //Fin del main.
```

La figura 1.3 muestra el circuito utilizado para la interrupción con el TMR0.

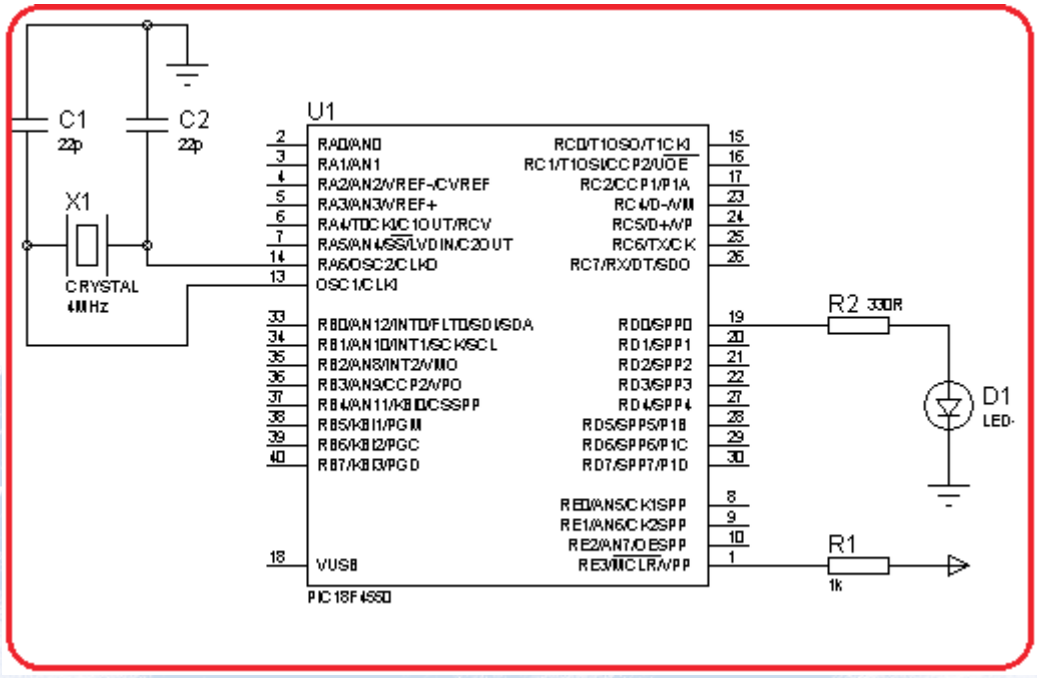


Figura 1.3. Circuito para demostración del funcionamiento del TMR0.
Elaborado por: Los Autores.

Uso del timer0 en modo de 16 bit como temporizador

El TMR0 trabajando en modo de 16 bits, puede contar hasta 65536 pulsos de la $F_{OSC}/4$. En este caso para trabajar con 16 bits la función CCS, que se utiliza es:

setup_timer_0(rtcc_div_256|rtcc_internal). Pres escalar a 256, TMR0 Interno con 16 bits, o también:

setup_counters(rtcc_internal,rtcc_div_256).

Ejercicio 1.2. TMR0 en modo temporizador y 16 bits.

Por el puerto RD0 se requiere sacar una señal cuadrada de 2 s (1s en alto y 1s en bajo). Utilizando la ecuación de tiempo, se determina el valor a cargar el temporizador TMR0.

65536 - TMR0 = Tiempo / 4 * TOSC * Rango Divisor de Frecuencia

Tiempo 1s = 1000000 us

Rango Divisor de frecuencia (pre-escalar) = 256
TOSC= 1/4MHz = 0.25us
65536 - TMR0= 1000000/(4*0.25us*256) = 3906
TMR0 = 65536 - 3906 = 61630 = 0xF0BE.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer0                                   //Interrupción del TMR0.
void DesbordeTimer0()                         //Función de la interrupción.
{
  output_bit(PIN_D0,!input(PIN_D0));          //Invierte el estado del PIN D0, cada Segundo.
  set_timer0 (0xF0BE);                        //Inicializa el TMR0 con 61630(F0BE).
}
void main(){                                  //Función principal main.
output_d(0x00);                               //Puerto D = 0.
          set_tris_d(0x00);                   //Fija como salida el Puerto D.
setup_timer_0(rtcc_div_256|rtcc_internal);            //Pres escalar a 256, TMR0.....
                                              //... Interno con 16.bits
set_timer0(0xF0BE);                           //Inicializa el TMR0 en 61630.
enable_interrupts ( INT_Timer0 );             //Habilitación de la interrupción del TMR0.
enable_interrupts ( GLOBAL);                  //Habilitación de las interrupciones globales.

  while(true)   {                             //Bucle infinito.
        // Aquí incluir instrucciones para el programa…
  }                                           //Fin del bucle infinito.
}                                             //Fin del main.
```

Uso del timer0 en modo contador con registro de 8 bits

Para utilizar el TMR0 como contador recuerde que se debe seleccionar los bits TOSE y TOCS del registro T0CON.

El TMR0 trabaja en modo de contador cuando cuenta los pulsos provenientes de una señal externa que se ingresa por el puerto RA4/TOCI.

Las funciones en CCS que permite utilizar como contador al temporizador TIMER0 son:

- ***setup_timer_0(rtcc_div_1 | rtcc_ext_L_TO_H | rtcc_8_bit):*** Fija el módulo TMR0 como contador usando el registro de 8 bits.

- ***rtcc_8_bit:*** TMR0 contador de 8 bits.
- ***rtcc_ext_L_TO_H:*** Detecta el cambio de bajo a alto. Para detectar de alto a bajo colocar ***rtcc_ext_H_TO_L:***
- ***rtcc_div_1:*** divisor de frecuencia 1.

Ejercicio 1.3. TMR0 en modo contador con registro de 8 bits.

En el siguiente ejercicio un LED conectado en el puerto RB0, se prende y se apaga cada 500 ms. Un LED conectado en RD0 cambia de estado cada 5 pulsos que ingresan por RA4. Como el registro del temporizador 0, en 8 bits puede contar hasta 256, para que se produzca el desborde, el TIMER0 se carga con 251.

PROGRAMA:

//USO DEL TIMER0 EN MODO DE 8 BIT COMO CONTADOR.

```
#include <18f4550.h>                        //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, PUT, NOWDT, NOLVP, NOCPD //Configuración de fusibles.
#use delay (clock=4000000)                  //F_OSC = 4 MHz.
#BYTE port_b= 0xF81                         //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                         //Identificador para el puerto b en  la localidad 0xF83.
#use fast_io(d)                             //Define el Puerto D para respuesta rápida.

#INT_Timer0                                 //Interrupción TMR0.
void DesbordeTimer0()                       //Función Desborde.
{output_bit(PIN_D0,!input(PIN_D0));         //Cada 5 impulsos cambia de valor.
SET_TIMER0(251);                            //Inicializa el TMR0 en 251.
}main(void)                                 //Función principal main.
{output_d(0x00);                            //Puerto d = 0.
        //Pre-escalar 1, Flanco de subida, TMR0 contador de 8 bits.
setup_timer_0(rtcc_div_1|rtcc_ext_l_to_h | rtcc_8_bit);
set_timer0(251);                            //Inicializa el TMR0 en 251.
enable_interrupts ( INT_Timer0 );           //Habilitación de la interrupción del TMR0.
enable_interrupts ( GLOBAL );               //Habilitación de las interrupciones globales.

while(TRUE)                                 //Bucle infinito.
{
  output_low(PIN_B0);                       //LED off.
  delay_ms(500);                            //Retardo de 500 ms.
  output_high(PIN_B0);                      //LED on.
  delay_ms(500);                            //Retardo de 500 ms.
 }                                          //Fin del bucle infinito.
}                                           //Fin del main.
```

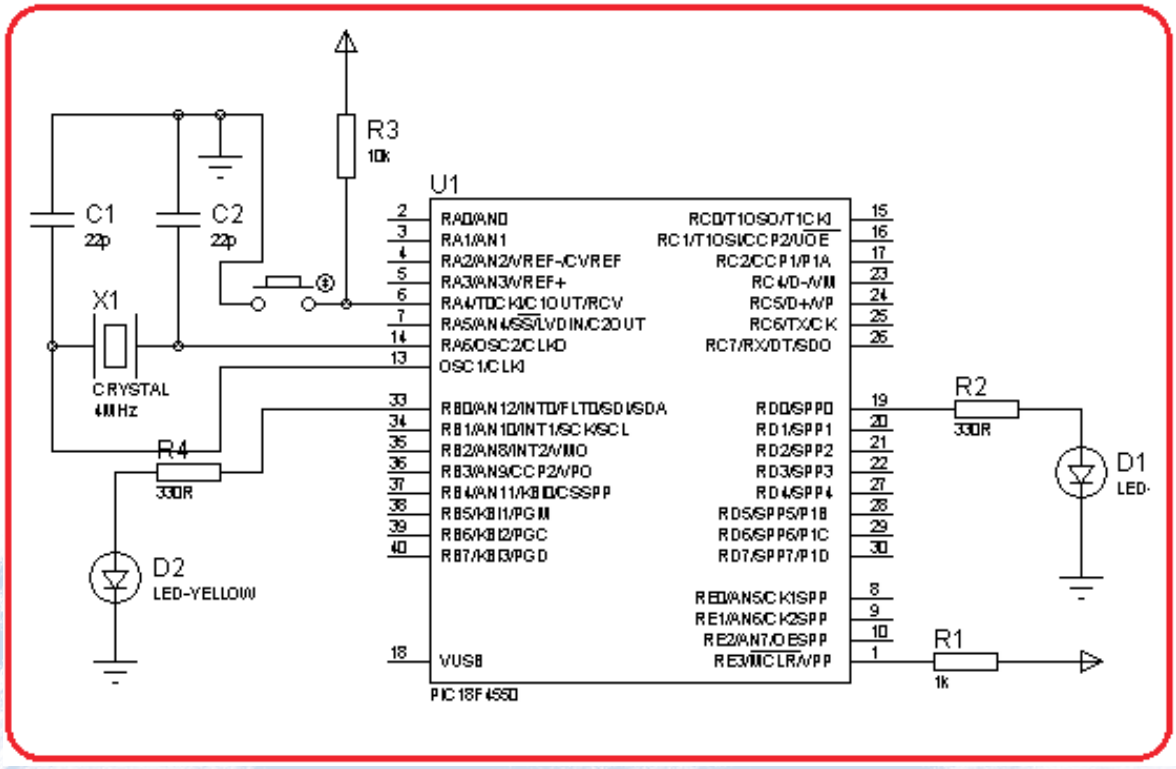


Figura 1.4. Circuito para demostrar al TMR0 como contador.
Elaborado por: Los Autores.

Uso del timer0 en modo contador con registro de 16 bits

Para modo de 16 bits, las funciones en CCS son:

SETUP_TIMER_0(RTCC_DIV_1|RTCC_EXT_L_TO_H):

RTCC_DIV_1: Divisor de frecuencia 1.

RTCC_EXT_L_TO_H: Detecta el cambio de bajo a alto.

Para detectar de alto a bajo colocar ***RTCC_EXT_H_TO_L***

Ejercicio 1.4. TMR0 en modo contador y registro de 16bits.

Realizando el mismo ejercicio anterior, se tiene el TMR0 cuenta hasta 65536 pulsos, por tanto, para que cuente 5 pulsos y se produzca el desborde es 65531 (FFFB), valor con el cual se fija el TIMER0.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOPROTECT, NOWRT, PUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer0                                   //Interrupción TMR0.
void DesbordeTimer0()                         //Función interrupción.
{
output_bit(PIN_D0,!input(PIN_D0));            //Cada 5 impulsos cambia de valor.
set_timer0(0xFFFB);                           //Inicializa el TMR0 en 65531.
}

main(void)                                    //Función principal main.
{
output_d(0x00);                               //Puerto d = 0.

setup_timer_0(rtcc_div_1|rtcc_ext_l_to_h);    //TMR0 con 16 bits.
set_timer0(0xFFFB);                           //Inicializa el TMR0 en 65531.
enable_interrupts ( INT_Timer0 );             //Habilitación de la interrupción del TMR0.
enable_interrupts ( GLOBAL );                 //Habilitación de las interrupciones globales.

while(TRUE)                                   //Bucle infinito.
 {
  output_low(PIN_B0);                         //LED off.
  delay_ms(500);                              //Retardo de 500 ms.
  output_high(PIN_B0);                        //LED on.
  delay_ms(500);                              //Retardo de 500 ms.
}                                             //Fin del bucle infinito.
}                                             //Fin del main.
```

TEMPORIZADOR 1

El módulo TMR1 temporizador/contador de 16 bits incorpora las siguientes características:

- Configurable como temporizador/contador sincrónico o asincrónico de 16 bits.
- Escritura y lectura en los registros TMR1H y TMR1L.
- Pre – escalar de 3 bits.
- Seleccionable la fuente de reloj interna o externa.
- Interrupción por desbordamiento (FFFF a 0000).
- Reinicio en evento especial del módulo CCP.

- Reloj del dispositivo indicador de estado (T1RUN).
- El registro T1CON controla todos los aspectos de la operación del módulo temporizador del TMR1.

T1CON: TIMER1 CONTROL REGISTER

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

Tabla 1.3. Distribución de bits del registro T1CON.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

RD16: 16-Bit Read/Write Mode Enable bit.

- 1 = Permite la lectura / escritura del registro Timer1 en una operación de 16-bits.
- 0 = Permite la lectura / escritura del registro Timer1 en dos operaciones de 8-bits.

T1RUN: Timer1 System Clock Status bit

- 1 = Reloj del dispositivo se deriva del oscilador Timer1.
- 0 = Reloj del dispositivo se deriva de otra fuente.

T1CKPS1:T1CKPS0: Timer1 Selección de los bits de pre escala de la entrada de reloj.

- 11 = 1:8 Valor del pre escalar.
- 10 = 1:4 Valor del pre escalar.
- 01 = 1:2 Valor del pre escalar.
- 00 = 1:1 Valor del pre escalar.

T1OSCEN: Timer1 Oscillator Enable bit.

- 1 = Timer1 oscilador está habilitado.
- 0 = Timer1 oscilador se desconecta.

El inversor oscilador y resistencia de realimentación se apagan para eliminar la fuga de energía.

T1SYNC: Timer1 External Clock Input Synchronization Select bit. Sincronización con la entrada de reloj externo.

CuandoTMR1CS = 1:

- 1 = No sincronizado con la entrada de reloj externo.
- 0 = Sincronizado con la entrada de reloj externo.

Cuando TMR1CS = 0:

Este bit se ignora. Timer1 utiliza el reloj interno.

TMR1CS: Timer1 Clock Source Select bit.

- 1 = Reloj .externo en los pines RC0/T1OSO/T13CKI (en el flanco ascendente)
- 0 = Reloj interno (F_{OSC} / 4).

TMR1ON: Timer1 On bit.

- 1 = Habilita Timer1.
- 0 = Detiene Timer1.

La figura 1.5, muestra el diagrama de bloques del módulo temporizador TMR1.

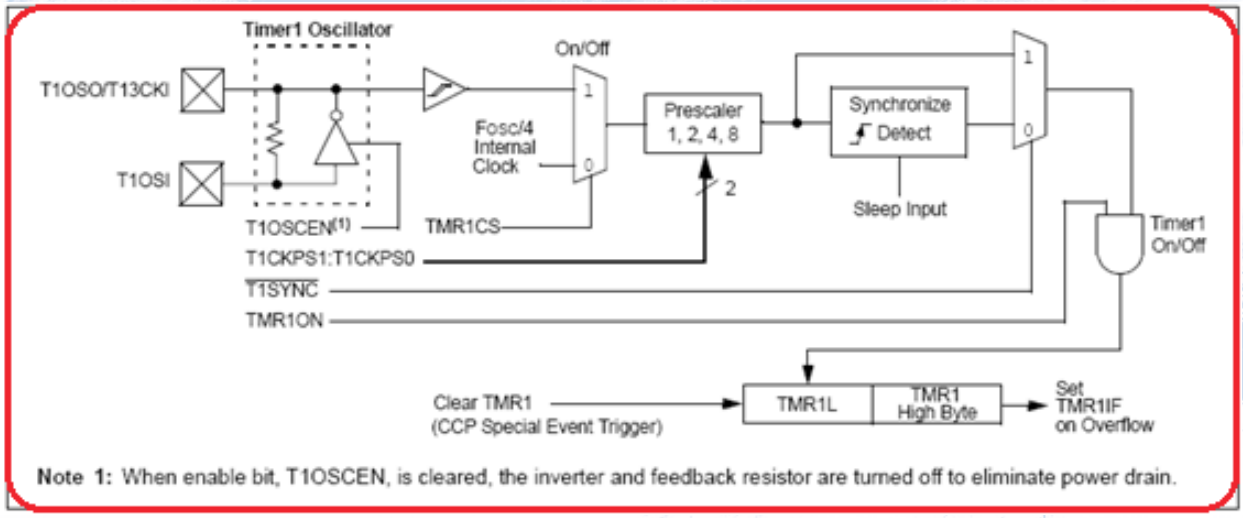


Figura 1.5. Diagrama de bloques del temporizador 1.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

El TIMER1 en modo de temporizador (TMR1CS= 0), incrementa en un ciclo de máquina y en modo contador (TMR1CS= 1), se incrementa por el flanco de subida de la señal externa.

El TIMER1 dispone de un oscilador interno que se puede habilitar con el bit T1OSCEN =1. En este caso los pines RC0/T1OSO/T1CKI y RC1/

T1OSI/CCP2/UOE. Además el TIMER1 tiene un reset interno que puede ser generado por el módulo CCP.

El tiempo de desbordamiento del TIMER1 se puede calcular con:

Tiempo = 4* TOSC * (65536-TMR1) * Rango Divisor de Frecuencia

El compilador CCS provee varias funciones y directivas para el manejo transparente del módulo TMR1, es decir, trata de que el programador no intervenga en la selección de los bits de control del registro T1CON.

Funciones para control del TIMER1 el compilador CCS

Las funciones más relevantes son:

setup_timer_1(modo). Deshabilita o fija la fuente del CLK y el pre-escalar para el TIMER1.

Donde, modo, puede ser:

T1_DISABLED, T1_INTERNAL, T1_EXTERNAL, T1_EXTERNAL_SYNC
T1_CLK_OUT
T1_DIV_BY_1, T1_DIV_BY_2, T1_DIV_BY_4, T1_DIV_BY_8

Cuando se requiere de varias constantes se puede añadir ordenando mediante el signo |. Ejemplo: T1_INTERNAL | T1_DIV_BY_1

set_timer1(value). Inicializa el valor del TIMER1, temporizador/contador.

Donde, ***value,*** es un número entero de 16 bits.

value=get_timer1. Devuelve el valor del TIMER1, temporizador/contador.

int_timer1. Interrupción del TIMER1 por desbordamiento.

Temporizador TMR1 como temporizador

Para usar el módulo temporizador TMR1 para que funcione como temporizador (cuente los pulsos del reloj del sistema), se debe definir los parámetros según se describen en las siguientes funciones:

- La fuente del reloj interno y el pre-escalar.
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
- Fijar el valor del TRM1
  set_timer0(value);

- Habilitar la interrupción del TMR1.
  enable_interrupts (INT_Timer1);
- Habilitar las interrupciones globales.
  enable_interrupts (GLOBAL);

Ejercicio 1.5. Uso del módulo TMR1 como temporizador.

El siguiente ejercicio se prende y se apaga un LED conectado al pin RD0 cada 500 ms. El tiempo es generado por el TMR1.

Utilizando la ecuación de tiempo, se determina el valor a cargar el temporizador TMR1.

65536 – TMR1 = Tiempo / 4 * TOSC * Rango Divisor de Frecuencia.
Tiempo 0.5s = 500 000 us
Rango Divisor de frecuencia (pre-escalar) = 8
TOSC= 1/4MHz = 0.25us
65536 – TMR1= 500 000/(4*0.25us*8) = 62 500
TMR1 = 65 536 – 62 500 = 3036 = $BDC.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP         //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer1                                   //Interrupción del TMR1.
void DesbordeTimer1(){                        //Función de la interrupción.
  output_bit(PIN_D0,!input(PIN_D0));          //Invierte el estado del PIN D0, cada 0.5 s.
  SET_TIMER1(0xBDC);                          //Inicializa el TMR1 con 3036 (BDC).
}

void main(){                                  //Función principal main.
output_d(0x00);                               //Puerto D = 0.
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);       //Pre escalar a 8 TMR1 Interno con 16 bits.
set_timer1(0xBDC);                            //Inicializa el TMR1 en 3036.
enable_interrupts ( INT_Timer1 );             //Habilitación de la interrupción del TMR1.
enable_interrupts ( GLOBAL );                 //Habilitación de las interrupciones globales.

  while(true) {                               //Bucle infinito
        restart_wdt();          //Reinicia el watchdog… Aquí incluir instrucciones para el programa…
  }                                           //Fin del bucle.
}                                             //Fin del main
```

Temporizador TMR1 como contador

Para usar el TMR1 como contador se debe definir:

- La fuente de pulsos externos y el pre-escalar.
- *setup_timer_1(T1_EXTERNAL | T1_DIV_BY_1);*
- Fijar el valor del TRM1
- *set_timer0(value);*
- Habilitar la interrupción del TMR1.
- *enable_interrupts (INT_Timer1);*
- Habilitar las interrupciones globales.
- *enable_interrupts (GLOBAL);*

Hay que recordar que los pulsos de la señal externa para que sean contados por el TMR1 deben ser ingresados por el terminal RC0/T1OSO/T1CKI.

Ejercicio 1.6. Uso del módulo TMR1 como contador.

A continuación en el ejercicio, un LED conectado en el pin RB0 titila cada 0.5 s. Cada vez que reciba el pin RC0/T1OSO/T1CKI 5 pulsos un LED conectado en el terminal RD0 cambia de estado. La figura 1.6, muestra las conexiones del circuito utilizado.

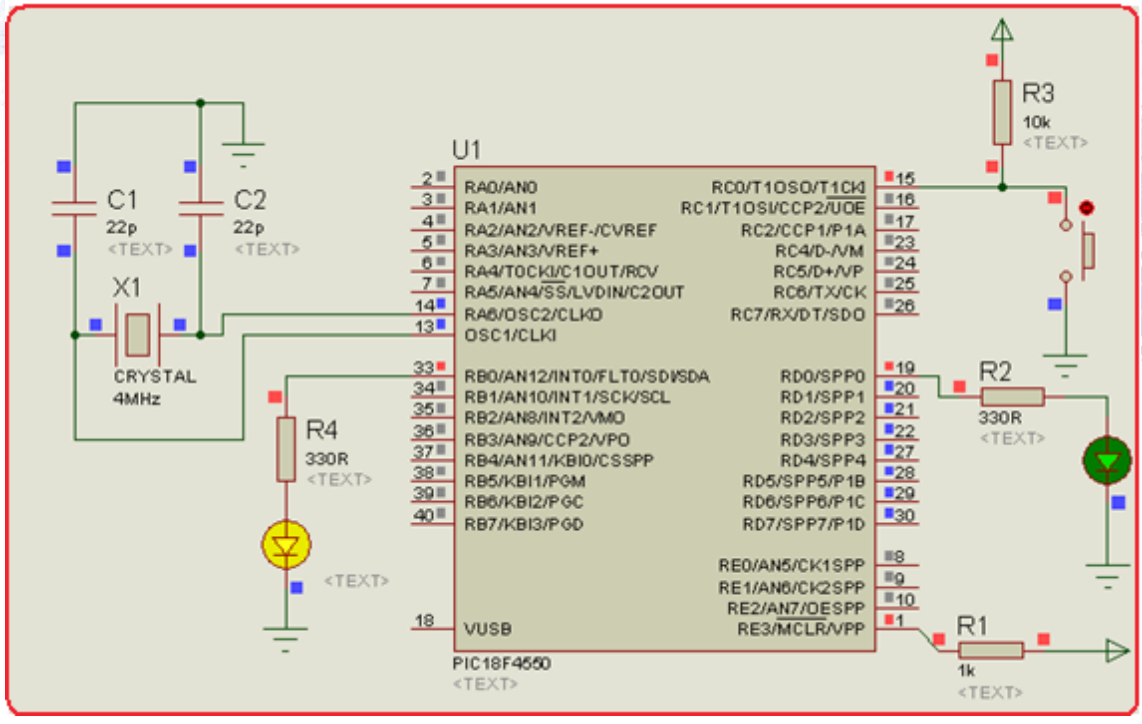


Figura 1.6. Uso del Timer1.
Elaborado por: Los Autores.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOPROTECT, NOWRT, NOPUT            //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer1{                                  //Solicita a la interrupción del TMR1.
void DesbordeTimer1()                         //Función de la interrupción.
  output_bit(PIN_D0,!input(PIN_D0));          //Invierte el estado del PIN D0, cada 0.5 segundo.
  SET_TIMER1(0xfffb);                         //Inicializa el TMR1 con 35531 (FFFB).
}

void main(){                                  //Función principal main.
output_d(0x00);                               //Puerto D= 0.
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);       //Pre-escalar a 8, TMR1 y 16 bits
set_timer_1(0xfffb);                          //Inicializa el TMR1 en 35531.
enable_interrupts ( INT_Timer1 );             //Habilitación de la interrupción del TMR1.
enable_interrupts ( GLOBAL );                 //Habilitación de las interrupciones globales.

  while(true)                                 //Bucle infinito
   {
  output_low(PIN_B0);                         //LED off.
  delay_ms(500);                              //Retardo de 500 ms.
  output_high(PIN_B0);                        //LED on.
  delay_ms(500);                              //Retardo de 500 ms.
   }                                          //Fin del bucle.
}                                             //Fin del main.
```

TEMPORIZADOR 2

El módulo TMR2 temporizador/contador incorpora las siguientes características:

- Temporizador y registro de periodo de 8 bits (registro TMR2 y PR2 respectivamente).
- Ambos registros se pueden leer o escribir.
- Pre-escalar programable por software (1:1, 1:4 y 1:16).
- Post-escalar programable por software (1:1 a 1:16).
- Interrupción controlada por PR2.
- El módulo MSSP (MASTER SYNCHRONOUS SERIAL PORT) utiliza opcionalmente el TMR2 para generar una señal de reloj.
- Se puede emplear como base de tiempos para la modulación del ancho de pulsos (PWM) utilizando el módulo CCP.
- El TIMER2 es controlado por el registro T2CON.

T2CON: T2CON: TIMER2 CONTROL REGISTER

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

Tabla 1.4. Distribución de bits del registro T2CON.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

T2OUTPS3:T2OUTPS0: Timer2 Output Post scale Select bits. Bits de selección del post-escalar.

0000 = 1:1
0001 = 1:2
……..
……..
……..
1111 = 1:16

TMR2ON: Timer2 On bit. Bit de habilitación del TMR2.
1 = Timer2 es activado.
0 = Timer2 es detenido.

T2CKPS1:T2CKPS0: Timer2 Clock Pre scale Select bits. Bits de selección del Pre-escalar.

00 = Prescaler es 1:1
01 = Prescaler es 1:4
1x = Prescaler es 1:16
Nota: x puede tomar cualquier valor (0 o1)
El diagrama de bloques del TMR2 se indica en la figura 1.7.

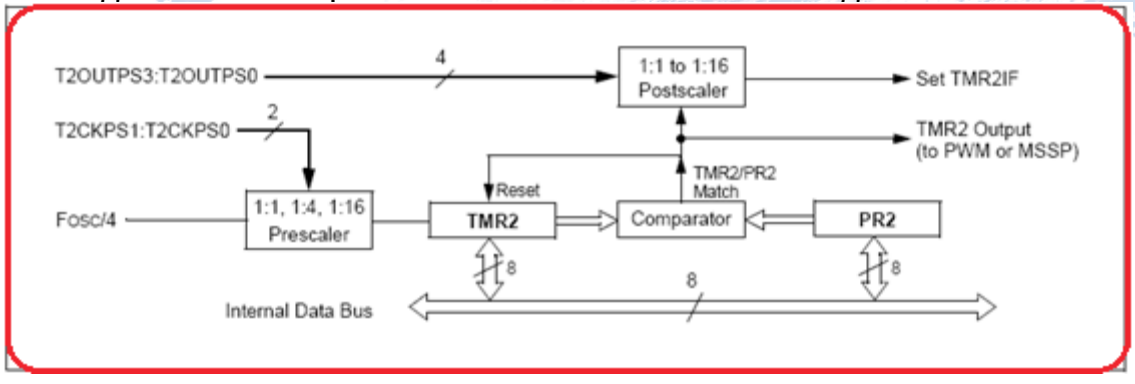


Figura 1.7. Diagrama de bloques del TIMER2.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Los contadores del pre-escalar y post-escalar son borrados cuando se escribe en el registro TMR2, en el registro T2CON o en cualquier *reset*. El TMR2 no se borra cuando se escribe en el T2CON.

En operación normal el TIMER2 incrementa de 00h en cada ciclo de máquina, según se haya fijado el valor del pre-escalar. Al mismo tiempo se comparan los valores de los dos registros TMR2 y PR2, cuando se han igualado se genera una señal de *reset,* provocando el aclarado del registro TMR2 lo que se puede llamar como desbordamiento del TIMER2 y es utilizado para producir la interrupción. El valor del PR2 para provocar la interrupción se calcula con la ecuación:

Tiempo = 4* TOSC * Pre-escalar*(PR2 + 1) * Post-escalar

Funciones de control para TIMER2 el compilador CCS

Con el compilador CCS el TIMER2 se configura:

setup_timer_2(modo,periodo,postscaler), donde:

Modo: Afecta a los bits TMR2ON y T2CKPS1:T2CKPS0 del registro T2CON, es decir, activa/desactiva el TMR2 y fija el pre-escalar. Puede ser: *T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16*

Periodo: valor entero de 8 bits (0 -255) para el registro PR2.

Postscaler: valor de post-escalar (1 a 16), según los bits T2OUTPS3:T2OUTPS0, del registro T2CON.

set_timer2(valor): Inicializa el ***valor*** del TIMER2. ***valor*** es un entero de 8 bits (0 a 255).

valor=get_timer2*:** Devuelve el ***valor del TMR2.

***int_timer2*:** Interrupción por desbordamiento del TIMER2.

Ejercicio 1.7. Utilizando el TIMER2, generar una onda cuadrada de 10 Hz, por el puerto RD0.

El periodo de señal T = 100 ms = o sea 50 ms en alto y 50 ms en bajo.

Con un pre-escalar de 16, un post-escalar de 16 y un oscilador de 4 MHz, aplicando la ecuación calculamos el valor del registro PR2.

Tiempo = 4* TOSC * Pre-escalar*(PR2 + 1) * Post-escalar

50 000us = 4*0.25us*16*(PR2+1)*16

PR2 = 194.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOPROTECT, NOWRT, NOPUT, NOWDT     //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer2{                                  //Interrupción del TMR1.
void DesbordeTimer2()                         //Función de la interrupción.
output_bit(PIN_D0,!input(PIN_D0));            //Invierte el estado del PIN D0, cada 0.5 s.
 }

void main(){                                  //Función principal main.
setup_timer_2(T2_DIV_BY_16,194,16);           //Pre-escalar a 16, PR2 = 194 y post-escalar a 16.
SET_TIMER2(0);                                //Inicializa el TMR2 en 0.
enable_interrupts ( INT_Timer2 );             //Habilitación de la interrupción del TM2.
enable_interrupts ( GLOBAL );                 //Habilitación de las interrupciones globales.

 while(true)                                  //Bucle infinito
   {restart_wdt();//Reinicia el watchdog… Aquí incluir instrucciones para el programa…
   }                                          //Fin del bucle infinito.
}                                             //Fin del main.
```

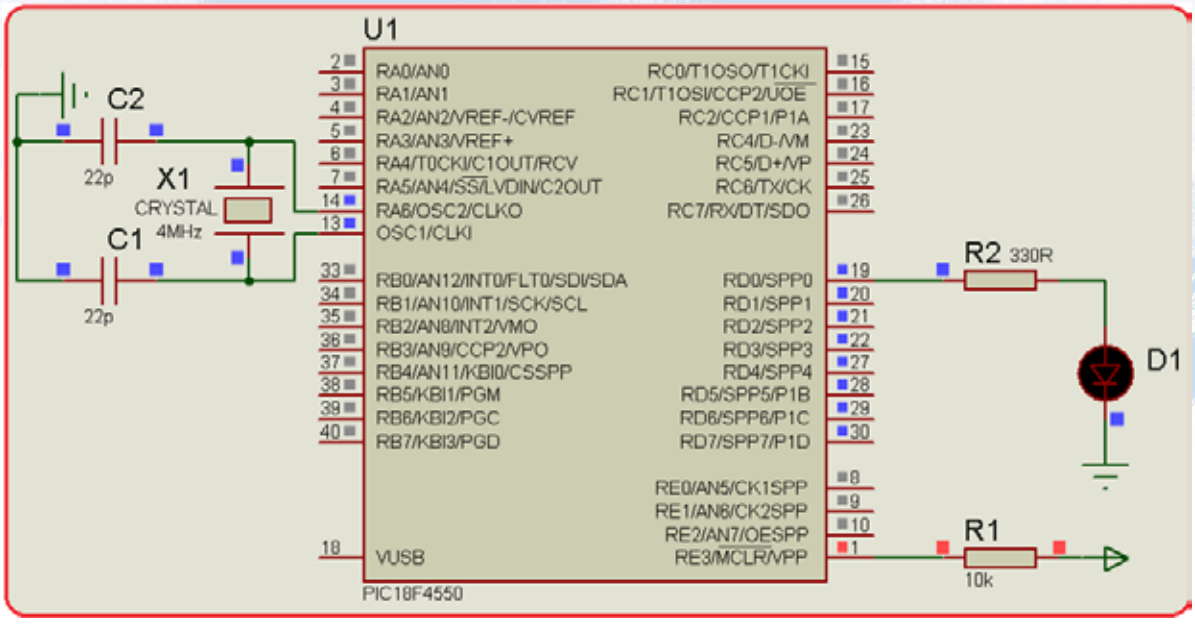


Figura 1.8. Circuito de aplicación para el TMR2.
Elaborado por: Los Autores.

El resultado de la simulación se muestra en el circuito de la figura 1.9.

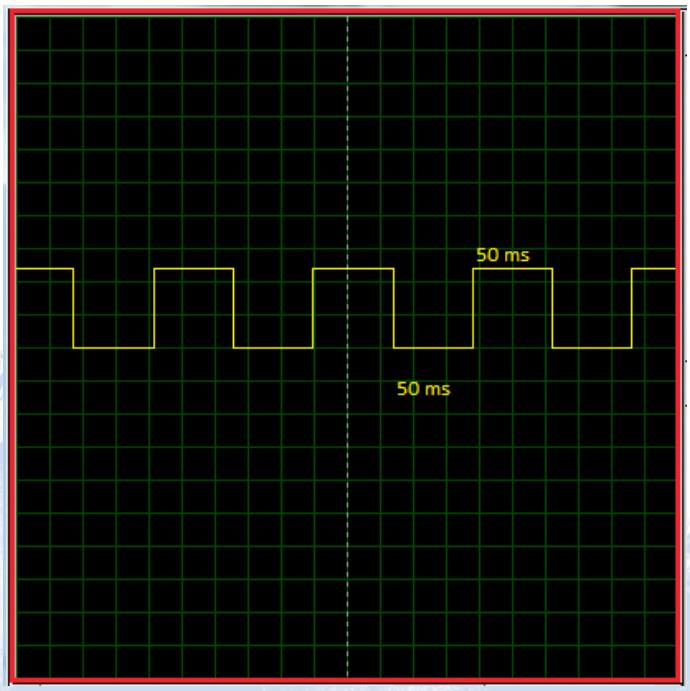


Figura 1.9. Generación de una onda cuadrada de 10 Hz.
Elaborado por: Los Autores.

TEMPORIZADOR 3

El módulo TMR3 de 16 bits es un temporizador/contador que incorpora las siguientes características:

- Seleccionable por software para operación de 16 bits como temporizador o contador.
- Se pueden leer o escribir en los registros de 8 bits (TMR3H y TMR3L).
- Seleccionable el dispositivo de la fuente de reloj (interna o externa) con opción de reloj o el oscilador interno del TIMER1.
- Interrupción por desbordamiento.
- Reset por disparo del módulo CCP.

El TIMER3 es controlado por el registro T3CON.

T3CON: TIMER3 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

Tabla 1.5. Distribución de bits del registro T3CON.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

RD16: Bit de habilitación de lectura/escritura 16 bits.

- 1 = Habilita los registros del TIMER3 para lectura/escritura en una operación de 16 bits.
- 0 = Habilita los registros del TIMER3 para lectura/escritura en dos operaciones de 8 bits.

T3CCP2:T3CCP1: Timer3 y Timer1 bits de habilitación para el módulo CCPx.

1x = Timer3 es la fuente de reloj para ambos módulos CCP (capture/compare).
01 = Timer3 es la fuente de reloj para el módulo CCP2 (capture/compare).
Timer1 es la fuente de reloj para el módulo CCP1 (capture/compare).
00 = Timer1 es la fuente de reloj para ambos módulos (capture/compare).

T3CKPS1:T3CKPS0: Timer3 Input Clock Prescale Select bits.

11 = 1:8
10 = 1:4
01 = 1:2
00 = 1:1

T3SYNC: Timer3 External Clock Input Synchronization Control bit. No utilizar si el reloj del dispositivo proviene de Timer1/Timer3.

Cuando TMR3CS = 1:

1 = No sincronizar entrada de reloj externo.
0 = Sincronizar entrada de reloj externo.

Cuando TMR3CS = 0: Este bit se ignora. El Timer3 usa el reloj interno.

TMR3CS: Timer3 Clock Source Select bit

- *1 = entrada de reloj externa del Timer1 oscilador o T13CKI (en el flanco de subida tras el primer flanco de bajada)*
- *0 = Reloj interno ($F_{OSC}/4$)*

TMR3ON: Timer3 On bit

- *1 = Habilita el Timer3*
- *0 = Para el Timer3*

Funciones para control del TIMER3 el compilador CCS

Dado que el TIMER3 trabaja de manera similar al TIMER1 las funciones más relevantes son similares para los dos temporizadores:

setup_timer_3(modo). Deshabilita o fija la fuente del CLK y el pre-escalar para el TIMER3.

Donde, ***modo***, puede ser:

T3_DISABLED, T3_INTERNAL, T3_EXTERNAL, T3_EXTERNAL_SYNC
T3_CLK_OUT
T3_DIV_BY_1, T3_DIV_BY_2, T3_DIV_BY_4, T3_DIV_BY_8

Cuando se requiere de varias contantes se puede añadir ordenando mediante el signo **|**. Ejemplo: T3_INTERNAL | T3_DIV_BY_1

set_timer3(valor). Inicializa el valor del TIMER3, temporizador/contador.

Donde, ***valor,*** es un número entero de 16 bits.

value=get_timer3. Devuelve el valor del TIMER3, temporizador/contador.

INT_TIMER3. Interrupción del TIMER3 por desbordamiento.

El tiempo de desbordamiento del TIMER1 se puede calcular con:

Tiempo = 4* TOSC * (65536-TMR3) * Rango Divisor de Frecuencia

Ejercicio 1.8. Prender/apagar un LED conectado en el pin RD0, cada 500 ms. Usar el TIMER3 para generar el tiempo.

Utilizando la ecuación de tiempo, se determina el valor a cargar el temporizador TMR3.

65536 – TMR3 = Tiempo / 4 * TOSC * Rango Divisor de Frecuencia.

Tiempo 0.5s = 500 000 us.

Rango Divisor de frecuencia (pre-escalar) = 8.

TOSC= 1/4MHz = 0.25us,

65536 – TMR3= 500 000/(4*0.25us*8) = 62 500.

TMR3 = 65 536 – 62 500 = 3036 = $BDC.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP         //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Time3                                    //Interrupción del TMR3.
void DesbordeTimer3(){                        //Función de la interrupción.
  output_bit(PIN_D0,!input(PIN_D0));          //Invierte el estado del PIN D0, cada 0.5 s.
  SET_TIMER3(0xbdc);                          //Inicializa el TMR3 con 3036 (BDC).
}

void main(){                                  //Función principal main.
output_d(0x00);                               //Puerto D = 0.
setup_timer_3(T3_INTERNAL|T3_DIV_BY_8);       //Pre-escalar a 8, TMR3 Interno con 16 bits
set_timer0(0xbdc);                            //Inicializa el TMR3 en 3036.
enable_interrupts ( INT_Timer3 );             //Habilitación de la interrupción del TMR3.
enable_interrupts ( GLOBAL );                 //Habilitación de las interrupciones globales.

 while(true)                                  //Bucle infinito
   {
restart_wdt();      //Reinicia el watchdog… Aquí incluir instrucciones para el programa…
  }                                           //Fin del bucle infinito.
}                                             //Fin del main.
```

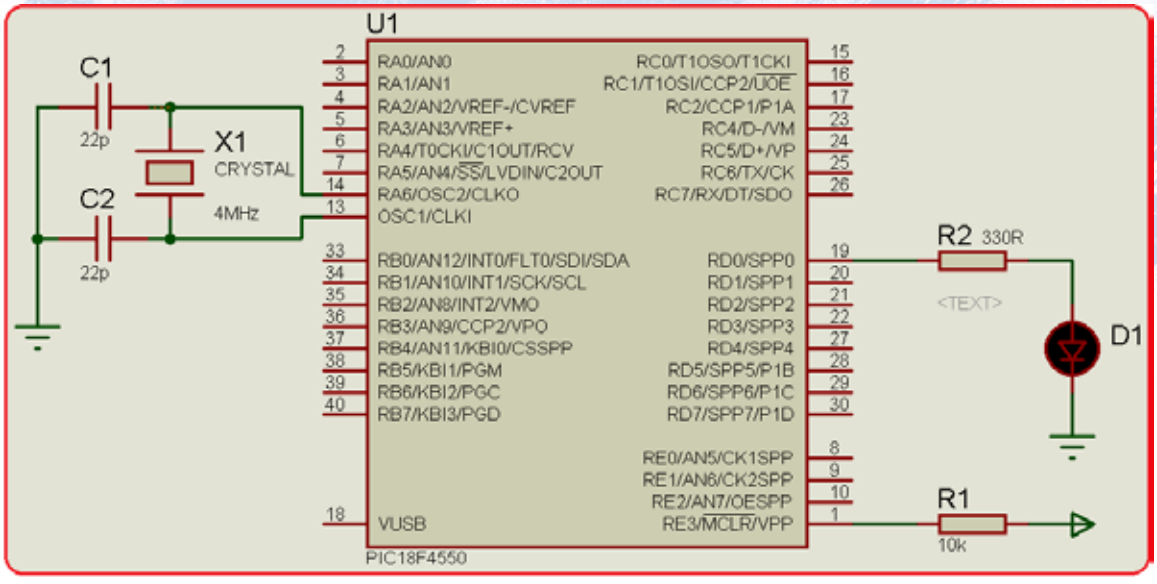


Figura 1.10. Aplicación del Timer3.
Elaborado por: Los Autores.

EJERCICIOS APLICACIÓN

Ejercicios resueltos

Generar una onda cuadrada de 1 KHz por el puerto RD0.

Por el puerto RD0 se requiere sacar una señal cuadrada de 1 KHz o 1 ms (0.5ms en alto y 0.5ms en bajo). Utilizando la ecuación de tiempo, determinamos el valor para cargar el temporizador TMR0.

65536 - TMR0 = Tiempo / 4 * TOSC * Rango Divisor de Frecuencia

Tiempo 0.5 ms = 500 us

Rango Divisor de frecuencia (pre-escalar) = 2 (para que el resultado sea entero)

T_{OSC}= 1/4MHZ = 0.25us

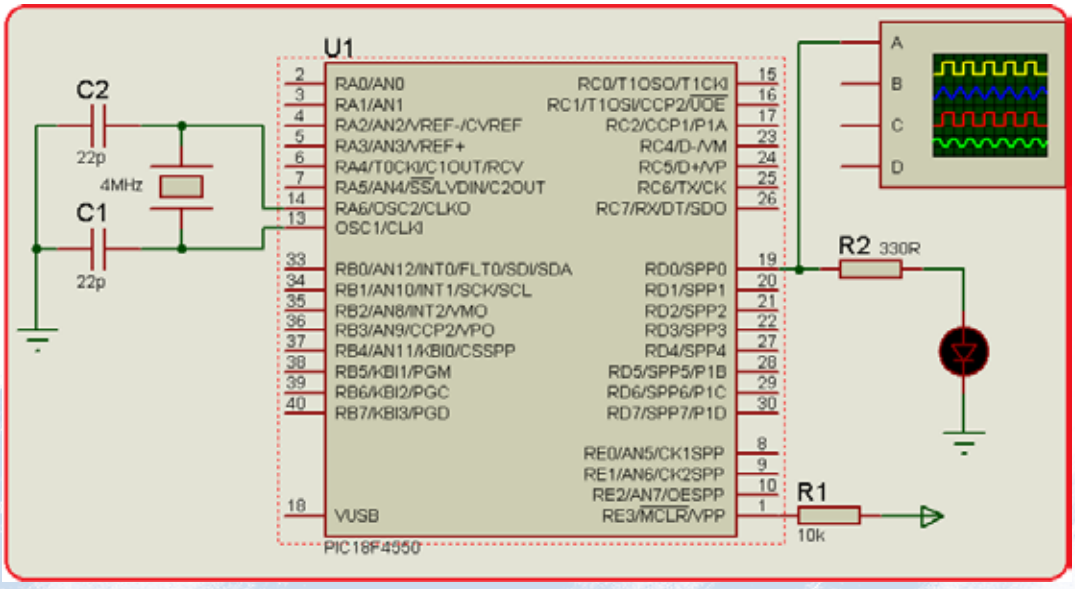
65536 - TMR0 = 500 us/(4*0.25us*2) = 250

TMR0 = 65536 - 255 = 65286

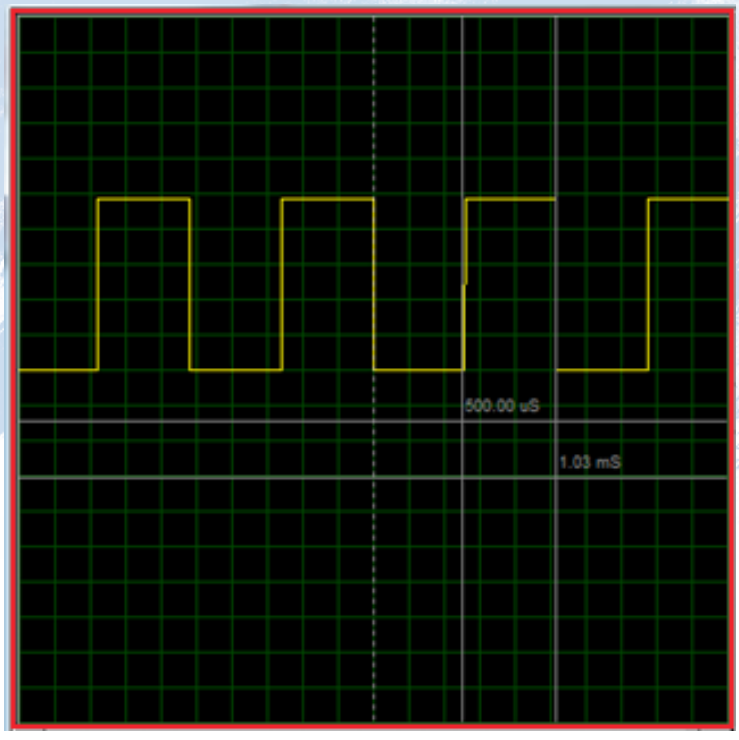
PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP         //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.

#INT_Timer0                                   //Interrupción del TMR0.
void DesbordeTimer0(){                        //Función de la interrupción.
  output_bit(PIN_D0,!input(PIN_D0));          //Invierte el estado del PIN D0, cada segundo.
  SET_TIMER0(65286);                          //Inicializa el TMR0 con 65286.

void main(){                                  //Función principal main.
output_d(0x00);                               //Puerto D = 0.
set_tris_d(0x00);                             //Fija como salida el Puerto D.
setup_timer_0(rtcc_div_2|rtcc_internal);      //Pre escalar a 256, TMR0.....
                                              //... Interno con 16 bits
set_timer0(65286)                             //Inicializa el TMR0 en 652861.
enable_interrupts ( INT_Timer0 );             //Habilitación de la interrupción del TMR0.
enable_interrupts ( GLOBAL);                  //Habilitación de las interrupciones globales.

   while(true) {                              //Bucle infinito
         // Aquí incluir instrucciones para el programa…
}                                             //Fin del bucle infinito.
}                                             //Fin del main.
```

La figura 1.11, indica el circuito utilizado y la señal obtenida. Nótese los tiempos en alto 500 ms y el periodo de 1.03 ms marcado en la onda.



a) Circuito



b) Onda.

Figura 1.11. Generador de onda cuadrada de 1 kHz.
Elaborado por: Los Autores.

Generar una onda cuadrada de 50 Hz con duty cycle (DC) de 5%, utilizando el Times 1.

Calcular el valor que debemos cargar al Timer1.

La frecuencia de salida es 50 Hz, por tanto, el periodo de la señal es:

Tsalida= 1/50Hz = 20 ms.

Con un DC del 5% el tiempo de alto es 1 ms.

Aplicando la ecuación de tiempo del TMR1

65536 – TMR1 = Tiempo / 4 * TOSC * Rango Divisor de Frecuencia

Tiempo 1 ms = 1000 us

Rango Divisor de frecuencia (pre-escalar) = 2 (para que el resultado sea entero)

TOSC= 1/4MHZ = 0.25us (Reloj del sistema del micro).

65536 – TMR1 = 1000 us/(4*0.25us*2) = 500

TMR1 = 65536 - 500= 65036.

La lógica del programa es:

- Al arrancar el microcontrolador una variable de control comienza en 1 y en éste valor el puerto RD0 se pone en alto.
- Se produce la primera interrupción del TMR1 al 1ms y desactiva a RD0.
- La variable de control registra el número de interrupciones del TMR1.
- Al llegar a 20 está variable se reinicia y nuevamente la salida RD0 se pone en alto y se repite el proceso.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP         //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC = 4 MHz.
#use fast_io(d)                               //Define el Puerto D para respuesta rápida.
Int contador = 1;                             //Variable de control.

#INT_Timer1                                   //Interrupción del TMR1.
void DesbordeTimer1(){                        //Función de la interrupción.
   output_bit(pin_d0,0);                      //RD0 en bajo.
   contador++;                                //Incrementa variable contador en 1.
   if (contador == 20){                       //Si contador es 20…
      contador=1;                             //… reinicia contador a 1.
  }
 SET_TIMER1(65036);                           //Inicializa el TMR1 con 65036.
}

void main(){                                  //Función principal main.
    output_d(0x00);                           //Puerto D = 0.
    set_tris_d(0x00);                         //Fija como salida el Puerto D.
                                              //TMR1 Interno con 16 bits y prescalar 2.
    setup_timer_1(T1_INTERNAL|T1_DIV_BY_2);
    set_timer1(65036);                        //Inicializa el TMR1 en 65036.
    enable_interrupts ( INT_Timer1 );         //Habilitación de la interrupción del TMR1.
    enable_interrupts ( GLOBAL);              //Habilitación de las interrupciones globales.
 while(true) {                                //Bucle infinito

  if (contador == 1){                         //Activa RD0 si contador es 1.
          output_bit(pin_d0,1);
       }
  }                                           //Fin del bucle infinito.
 }                                            //Fin del main
```

La figura 1.12, muestra la señal obtenida. Nótese que el pulso en alto es 1ms y el periodo total de 20 ms. Esta señal se podría aplicar para el funcionamiento de un servomotor.

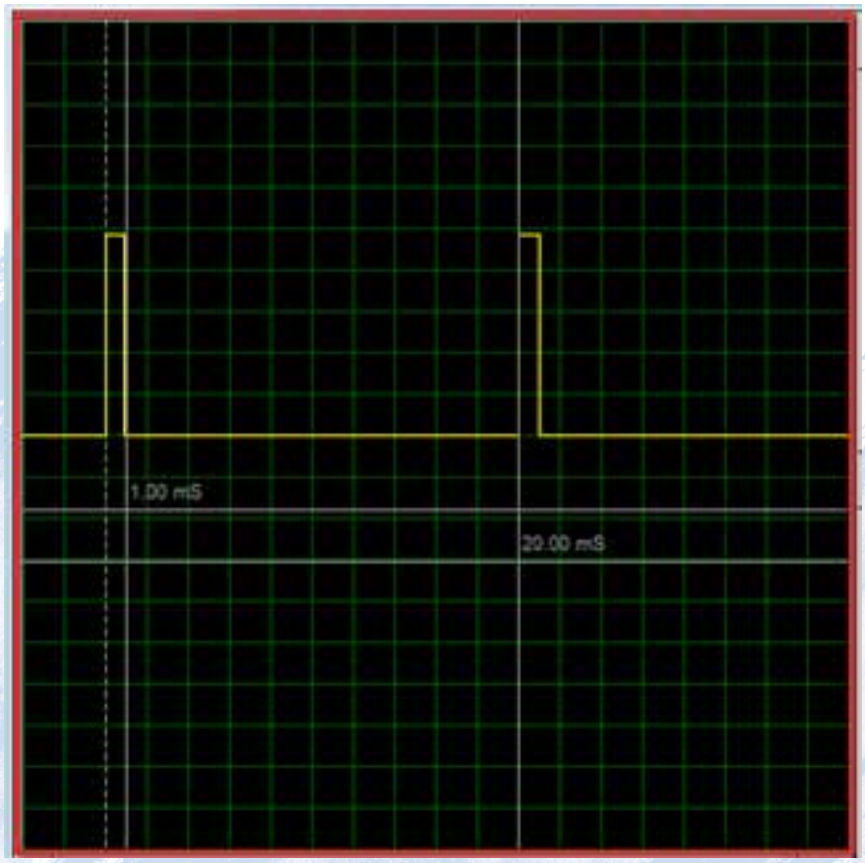


Figura 1.12. Señal de 50 Hz, con duty cycle de 5%.
Elaborado por: Los Autores.

Contador ascendente MOD-10 con temporizador.

La aplicación consiste en realizar un contador ascendente de 0-9 visualizado en display de 7 segmentos. El retardo será de 1 s. y generado por el TIMER0. Se dispone de un pulsador para reset del contador que será generado por la interrupción externa.

Nota: En la sección del temporizador TMR0 de 16 bits se analizó como calcular los valores del TMR0 para generar retardos de 1 s.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#use fast_io(d)                         //Define el Puerto D para respuesta rápida.
#define inicio bit_test(port_b,1)
#define estrella bit_test(port_d,2)
#BYTE port_b= 0xF81                     //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                     //Identificador para el puerto d en  la localidad 0xF83.
#BYTE tris_b = 0xF93                    //Identificador del registro tris b en  la localidad 0xF93.
#BYTE tris_d = 0xF95                    //Identificador del registro tris d en  la localidad 0xF95.

int contador = 1;                       //Variable de control.
int i=0;                                //Variable i condiciona el funcionamiento del pulsador de inicio.

#int_ext                                //Interrupción externa.
void PortB0_Interrupt(void) {           //Función interrupción externa.
 ext_int_edge(H_TO_L);                  //Habilita la interrupción externa  en flanco de bajada.
 i= 0;                                  //i= 0, apaga el circuito,  motor apagado.
 output_d(0x00);                        //Asigna 0 al Puerto D.
 DISABLE_INTERRUPTS( INT_Timer1 );      //Deshabilita el TMR1, cuando el circuito está apagado.
 }

#int_Timer1                             //Interrupción del TMR1.
void DesbordeTimer1(){                  //Función de la interrupción.
   i=0;                                 //Desactiva al pulsador de inicio. El motor está funcionando.
                                        //Retardo de 5s.
   contador++;                          //Incrementa variable contador en 1.
   if (contador == 10){                 //Si contador es 10…
      output_bit(pin_d1,0);             //Apaga el contactor estrella.
      output_bit(pin_d2,1);             //Activa el contactor triangulo.
      contador=1;                       //… reinicia contador a 1.
   }
 SET_TIMER1(3036);                      //Inicializa el TMR1 con 3036.
}

void main(){                            //Función principal main.
set_tris_b(0xFF);                       //Puerto B como entrada.
output_d(0x00);                         //Puerto D = 0.
set_tris_d(0x00);                       //Fija como salida el Puerto D.
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);      // TMR1 Interno con 16 bits,  pre escalar de 8,
set_timer1(3036);                       //Inicializa el TMR1 en 3036.
enable_interrupts ( INT_EXT);           //Habilitación de la interrupción externa.
enable_interrupts ( GLOBAL);            //Habilitación de las interrupciones globales.

  while(true) {                         //Bucle infinito
    if (inicio == 0 && estrella==0){    //Arranca el motor si cumple condiciones.
    ENABLE_INTERRUPTS( INT_Timer1 );    //Habilita interrupción del TMR1.
    i=1;                                //Condición para arranque del motor
    }
  if(i==1){
    output_bit(pin_d0,1);               //Salida activa para el contactor de línea.
    output_bit(pin_d1,1);               //Salida activa para el contactor de estrella.
  }
  }                                     //Fin del bucle infinito.
}                                       //Fin del main.
```

El circuito de la figura 1.13 muestra los componentes y las conexiones requeridas para el ejercicio.

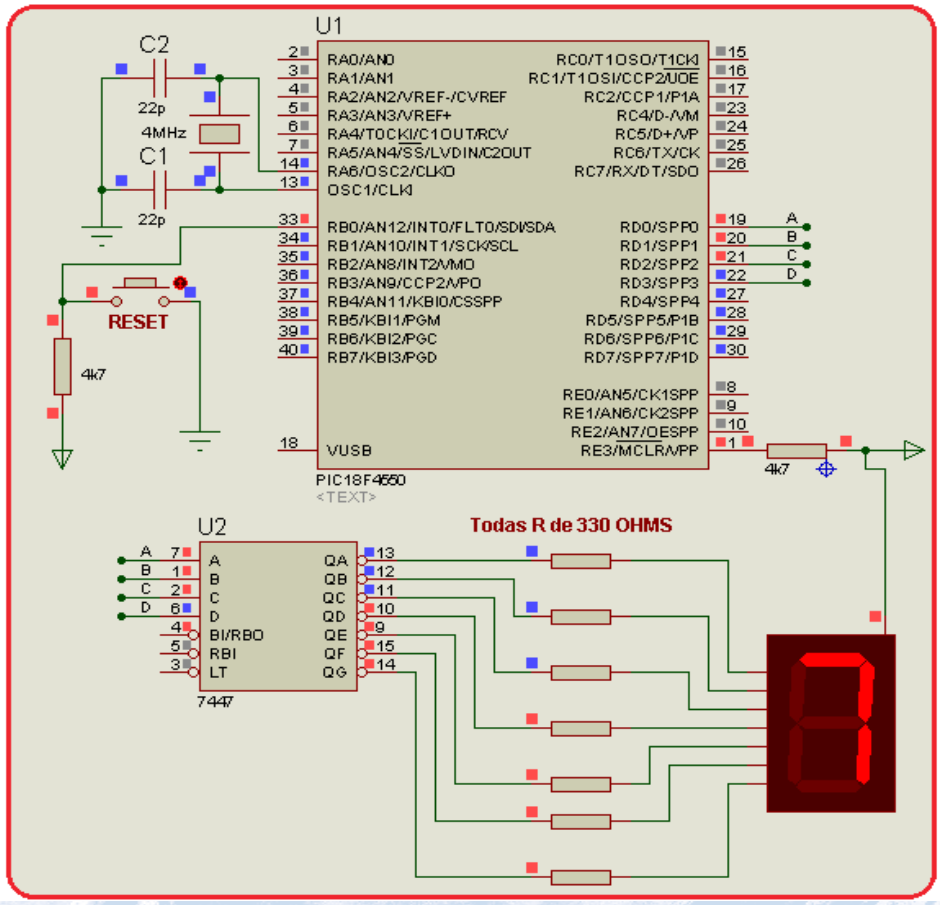


Figura 1.13. Contador ascendente módulo 10, con *reset*.
Elaborado por: Los Autores.

Arranque estrella – triangulo de un motor trifásico.

Uno de los circuitos más populares en control industrial es el "famoso" arranque estrella triángulo de un motor trifásico de inducción. Este tipo de arranque se ha realizado con componentes eléctricos como relés, contactores o dispositivos electrónicos como PLC. En nuestro caso vamos a utilizar un microcontrolador para realizar el circuito de control. El PIC entregará las señales digitales para mediante circuitos de interfaces adecuados manejar al motor eléctrico. Las interfaces con optoacopladores, transistores, relés u otros dispositivos eléctricos no objetivo de estudio de este libro. Se realizará la programación del Micro para obtener las señales digitales, que manejen al circuito de acoplamiento.

Recordar el funcionamiento. El motor arranca en estrella, un contactor conecta las bobinas del motor en esta condición y por lo general, también existe un contactor de línea que alimenta al motor. Transcurrido un cierto tiempo el motor cambia sus conexiones internas de los arrollamientos a la conexión que se conoce en triángulo, apagando el contactor de estrella.

De esto se deduce que necesitamos tres salidas: LÍNEA, ESTRELLA y TRIÁNGULO.

- RD0 – LÍNEA.
- RD1 – ESTRELLA.
- RD2 – TRIÁNGULO.

  Dos pulsadores INICIO y PARO.
- RB1 – INICIO.
- RB0 – PARO. Mediante una interrupción se detendrá el funcionamiento del motor en cualquier momento.

Un temporizador para generar el retardo para la conexión de estrella a triángulo. Se utilizará el TMR1. En este tipo de circuitos no se debe utilizar para generar el retardo la instrucción DELAY. Recuerde queda el micro inutilizado cuando se provoca un retardo de esta forma.

Calcular el valor que se debe cargar al Timer1.

Suponer que el cambio de estrella a triángulo se realiza en 5s. Como este tiempo es demasiado grande para generar directamente con el TIMER1, se generará un tiempo menor y se repite las veces que falta para completar el tiempo requerido.

Para el caso a tratarse se considera un tiempo de 500 ms, por tanto, se tiene que repetir 10 interrupciones para generar los 5s.

Aplicando la ecuación de tiempo del TMR1, se calcula de carga del TMR1 y se aplica un pre escalar de máximo de 8.

65 536 – TMR1 = Tiempo / 4 * T_{OSC}* Rango Divisor de Frecuencia

Tiempo= 500 000 us

Rango Divisor de frecuencia (pre-escalar) = 8

T_{OSC}= 1/4MHz = 0.25us (Reloj del sistema del micro).

65 536 – TMR1 = 500 000 us/(4*0.25us*8) = 62 500

TMR1 = 65 536 – 62 500= 3 036.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#use fast_io(d)                         //Define el Puerto D para respuesta rápida.
#define inicio bit_test(port_b,1)
#define estrella bit_test(port_d,2)
#BYTE port_b= 0xF81                     //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                     //Identificador para el puerto d en  la localidad 0xF83.
#BYTE tris_b = 0xF93                    //Identificador del registro tris b en  la localidad 0xF93.
#BYTE tris_d = 0xF95                    //Identificador del registro tris d en  la localidad 0xF95.

int contador = 1;                       //Variable de control.
int i=0;                                //Variable i condiciona el funcionamiento del pulsador de inicio.

#int_ext                                //Interrupción externa.
void PortB0_Interrupt(void) {           //Función interrupción externa.
  ext_int_edge(H_TO_L);                 //Habilita la interrupción externa  en flanco de bajada.
  i= 0;                                 //i= 0, apaga el circuito,  motor apagado.
  output_d(0x00);                       //Asigna 0 al Puerto D.
 DISABLE_INTERRUPTS( INT_Timer1 );      //Deshabilita el TMR1, cuando el circuito está apagado.
  }

#int_Timer1                             //Interrupción del TMR1.
void DesbordeTimer1(){                  //Función de la interrupción.
    i=0;                                //Desactiva al pulsador de inicio. El motor está funcionando.
                                        //Retardo de 5s.
    contador++;                         //Incrementa variable contador en 1.
    if (contador == 10){                //Si contador es 10…
      output_bit(pin_d1,0);             //Apaga el contactor estrella.
      output_bit(pin_d2,1);             //Activa el contactor triangulo.
      contador=1;                       //… reinicia contador a 1.
   }
  SET_TIMER1(3036);                     //Inicializa el TMR1 con 3036.
}
void main(){                            //Función principal main.
set_tris_b(0xFF);                       //Puerto B como entrada.
output_d(0x00);                         //Puerto D = 0.
set_tris_d(0x00);                       //Fija como salida el Puerto D.
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);      // TMR1 Interno con 16 bits,  pre escalar de 8,
set_timer1(3036);                            //Inicializa el TMR1 en 3036.
enable_interrupts ( INT_EXT);                //Habilitación de la interrupción externa.
enable_interrupts ( GLOBAL);                 //Habilitación de las interrupciones globales.

   while(true) {                             //Bucle infinito
     if (inicio == 0 && estrella==0){        //Arranca el motor si cumple condiciones.
     ENABLE_INTERRUPTS( INT_Timer1 );        //Habilita interrupción del TMR1.
     i=1;                                    //Condición para arranque del motor
     }
   if(i==1){
     output_bit(pin_d0,1);                   //Salida activa para el contactor de línea.
     output_bit(pin_d1,1);                   //Salida activa para el contactor de estrella.
    }
   }                                         //Fin del bucle infinito.
}                                            //Fin del main.
```

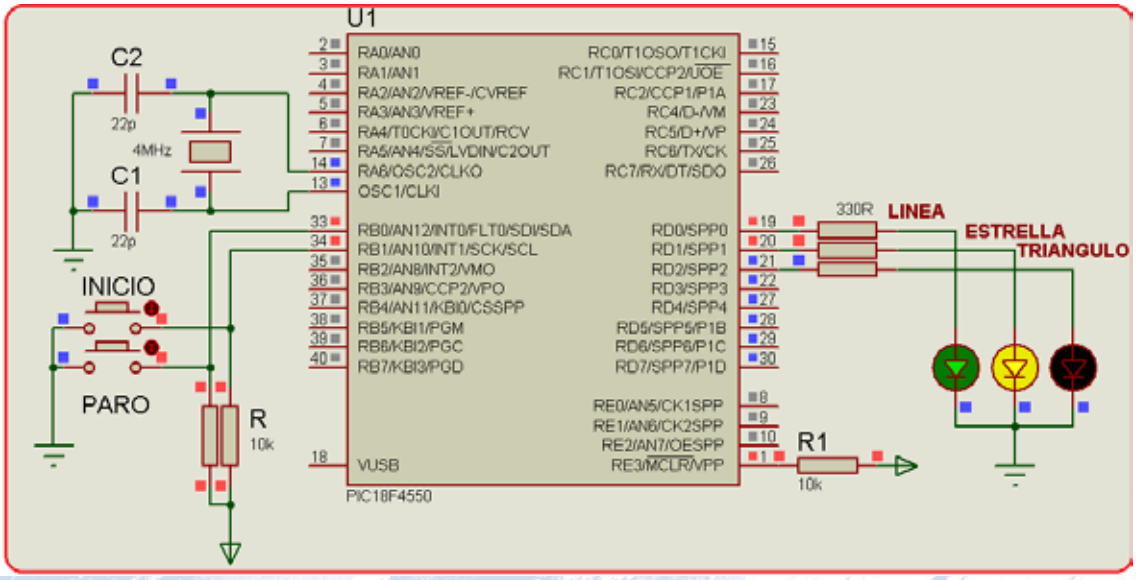


Figura 1.14. Circuito de control para arranque estrella – triángulo de motor trifásico de inducción.
Elaborado por: Los Autores.

Diseñar el programa y circuito para controlar un proceso que cuente 6 cajas y se detenga. Transcurridos 10 s, el proceso se inicia automáticamente. Un LED se activa indicando que le proceso está desarrollando normalmente y se apaga cuando está detenido. Existe un botón de paro de emergencia, en este caso el contador se pone en cero y el proceso se detiene. El proceso se inicia mediante un pulsador.

Con este ejercicio se demostrará la potencialidad del Microcontrolador, al realizar un circuito de automatización que por lo general se realiza con PLC.

Requerimientos:

- Un LED conectado en el puerto RB7 indicará el proceso (1) activo o apagado (0).
- Mediante la interrupción int_ext se utiliza para reiniciar el sistema. El pulsador de inicio conectado en el puerto RB1, servirá para iniciar el proceso.
- El temporizador TMR1 generará el retardo de 10s. En el ejercicio 1.4 se realizó el proceso para obtener 5s. Utilizando el mismo procedimiento para 10 s, bastaría que la variable contador cuente hasta 20.
- Para contar los pulsos generados por el SENSOR CAJAS, se utilizará el Timer0 operando en modo contador de 8 bits. Como en el ejercicio son 6 cajas, el temporizador para llegar a 255 que es el máximo valor tendríamos que cargar con 249.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#use fast_io(d)                         //Define el Puerto D para respuesta rápida.
#use fast_io(b)                         //Define el Puerto B para respuesta rápida.
#define inicio bit_test(port_b,1)       //Define al puerto RB1 como inicio.
#BYTE port_b= 0xF81                     //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_d= 0xF83                     //Identificador para el puerto d en  la localidad 0xF83.
#BYTE tris_b = 0xF93                    //Identificador del registro tris b en  la localidad 0xF93.
#BYTE tris_d = 0xF95                    //Identificador del registro tris d en  la localidad 0xF95.

int contador = 1;                       //Controla el número de repeticiones de la interrupción TRM1.
int i=0;                                //Variable i, controla el estado del proceso.
int valor;                              //Almacena el valor actual del TMR0.
int x;                                  //Indica la posición del array de datos.
const int m[]= {64,121,36,48,25,18,2};  //Datos decodificados en 7 segmentos del 0 al 6.

#int_ext                                //Interrupción externa.
void PortB0_Interrupt(void) {           //Función interrupción externa.
  ext_int_edge(H_TO_L);                 //Habilita la interrupción externa  en flanco de bajada.
  i= 0;                                 //i= 0, apaga el circuito,  motor apagado.
  output_d(64);                         //Asigna 0 al Puerto D.
  output_bit(pin_b7,0);
  DISABLE_INTERRUPTS( INT_Timer1 );   //Deshabilita el TMR1.
  DISABLE_INTERRUPTS( INT_Timer0 );   //Deshabilita el TMR0, cuando el circuito está apagado.
  }
#INT_Timer0                             //Interrupción TMR0.
void DesbordeTimer0(){                  //Función del TMR0.
   SET_TIMER0(249);                     //Inicializa el TMR0 con 249.
   SET_TIMER1(3036);                    //Inicializa el TMR1 con 3036.
}
#int_Timer1                             //Interrupción del TMR1.
void DesbordeTimer1(){                  //Función de la interrupción.
                                        //Genera los 10s.
   contador++;                          //Incrementa variable contador en 1.
   if (contador == 20){                 //Si contador es 10…
      output_bit(pin_b7,1);             //Desactiva el proceso.
      contador=1;                       //… reinicia contador a 1.
      i= 1;                             //Transcurridos los 10 s, reinicia el proceso automáticamente.
      SET_TIMER0(249);                  //Inicializa el TMR0 con 249.
      x=0;                              //Apunta al 64 de la tabla de datos del display.
   }
}
void main(){                            //Función principal main.
set_tris_b(0x0F);                       //Puerto B como entrada.
set_tris_d(0x00);                       //Fija como salida el Puerto D.
output_d(64);                           //Puerto D = 0.
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);                 //TMR1 Interno con 16 bits y pre escalar de 8.
enable_interrupts ( INT_EXT);           //Habilitación de la interrupción externa.
enable_interrupts ( GLOBAL);            //Habilitación de las interrupciones globales.

  while(true) {                         //Bucle infinito.
    if (inicio == 0){                   //Inicio manual del proceso.
       i=1;                             //Proceso activado.
       SET_TIMER0(249);                 //Fija el TMR0 en 249.
    }
   if(i==1){                            //Proceso activado…
```

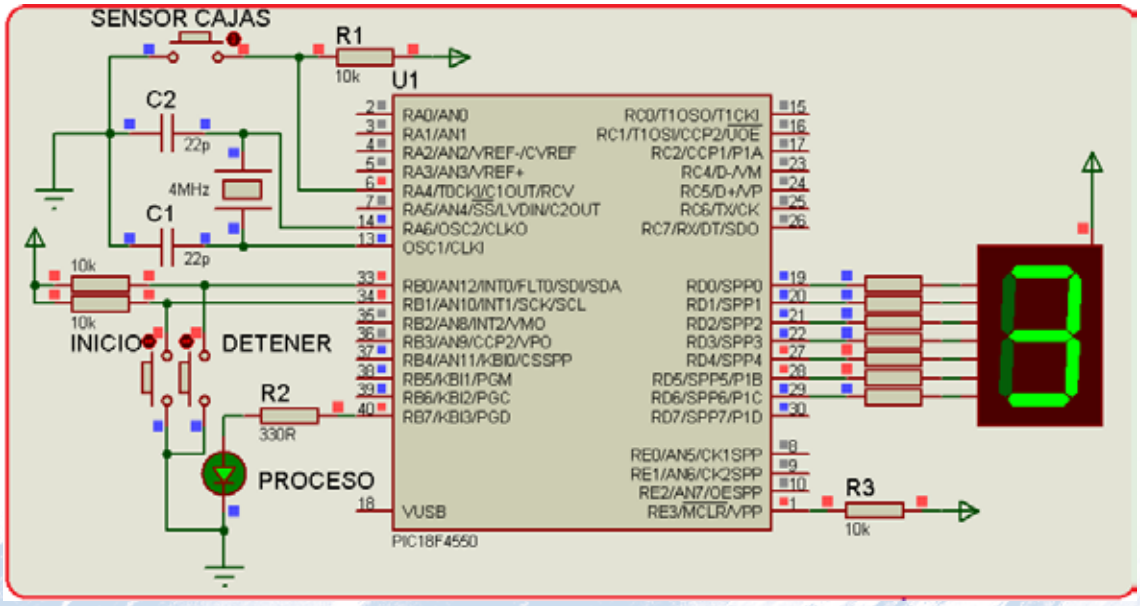


Figura 1.15. Gráfico para el ejercicio 1.5.
Elaborado por: Los Autores.

EJERCICIOS PROPUESTOS

Calcule el máximo valor de temporización que se puede obtener con el Timer2 si el oscilador del sistema es 12 MHz.

Realice un programa para un contador del 0 al 9 usando el Timer3. Visualizar la cuenta en un display de 7 segmentos y activar un LED cada vez que ocurre el desborde del TIMER.

Diseñe un programa y el circuito para contador de personas que ingresa por el acceso principal a un Cine. En un LCD, se debe indicar el número de asistentes. Cada diez personas se activará momentáneamente un LED.

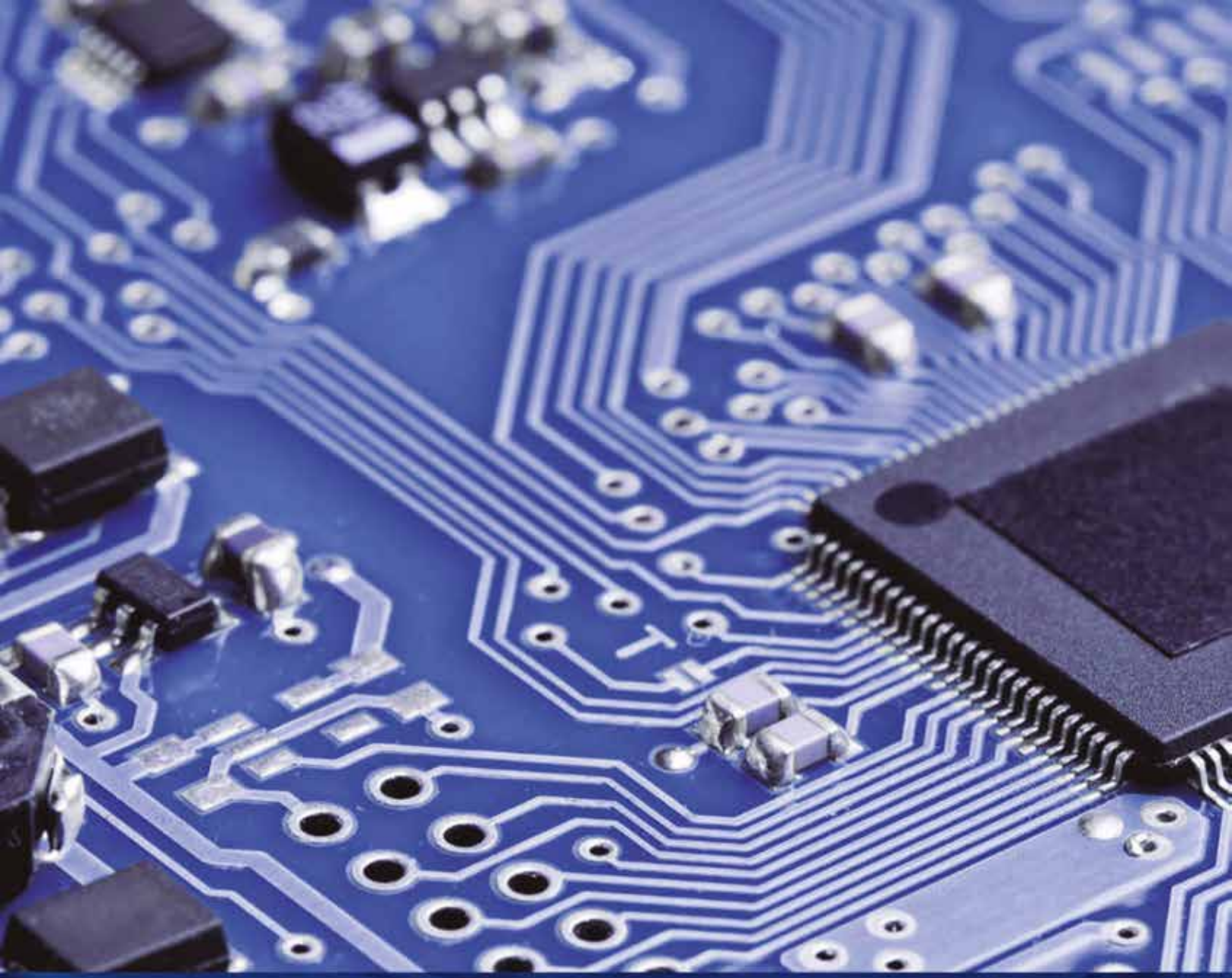
Elaborar el programa que permita a un Microcontrolador controlar un sistema de llenado de botellas que ha de cumplir las siguientes especificaciones:

a. La cinta transportadora de las botellas (y como consecuencia el resto del proceso) se pondrá en marcha mediante un pulsador.
b. Las botellas pasarán por un detector óptico (sensor) para saber si están llenas. En caso de detectar una botella vacía, será apartada de la línea mediante la acción de un cilindro que la empujará.
c. A continuación la botella pasará ante un sensor de proximidad, que cada vez que detecte el paso de una botella deberá bajar el cilindro para cerrarla mediante el tapón correspondiente.

d. Cuando se hayan sellado 100 botellas, la cinta transportadora deberá pararse durante 1 minuto para recargar el contenedor de tapones. Transcurrido este tiempo, la cinta arrancará automáticamente y con ella todo el proceso.

Se requiere automatizar la puerta de acceso de un hospital para que funcione de la siguiente forma:

a. La puerta es accionada por un motor reversible.
b. La puerta debe abrirse automáticamente al acercarse una persona.
c. La puerta debe permanecer abierta mientras se halle alguien en la zona de acceso.
d. La puerta debe cerrarse automáticamente la puerta tras 5 s., un tiempo de espera. Utilizar el TIMER 1 para generar el tiempo de 5 s.
e. Se debe disponer de pulsadores manuales de emergencia para abrir o cerrar la puerta en caso de fallo en los sensores. Deben utilizar interrupciones externas para generar las señales de emergencia.



CAPÍTULO 2

MÓDULOS: ADC Y CCP

MÓDULO ADC

El módulo conversor analógico - digital (A / D) tiene10 entradas para los dispositivos de 28 pines y 13 para los de 40/44 pines. Este módulo permite la conversión de una señal de entrada analógica correspondiente a un número digital de 10 bits.

Características:

- 10 bits de resolución.
- 13 canales multiplexados.
- Señal de reloj configurable.
- Tiempo de adquisición programable (0 a 20 TAD).
- Rango de tensión de conversión configurable mediante tensión de referencia externa.

El módulo tiene cinco registros:

- A / D Registro de resultado alto (ADRESH)
- A / D Registro de resultado bajo (ADRESL)
- A / D Registro de Control 0 (ADCON0)
- A / D Registro de Control 1 (ADCON1)
- A / D Registro de control 2 (ADCON2)

La figura 2.1, muestra el diagrama de bloques del módulo A/D.

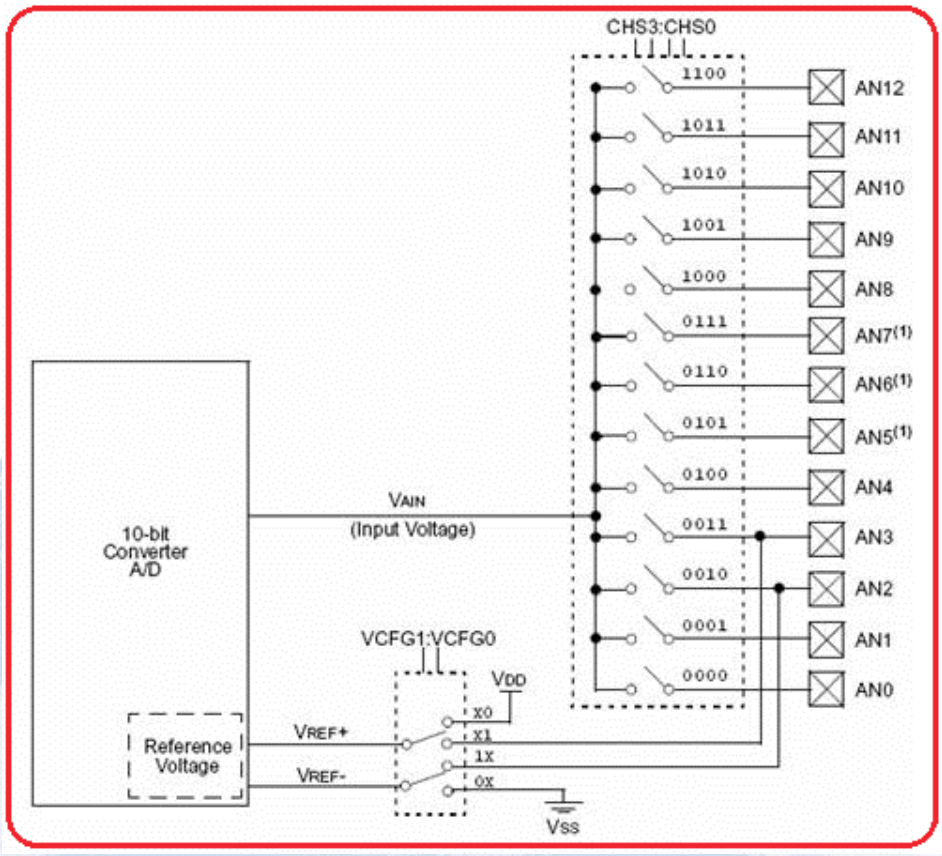


Figura 2.1. Diagrama de bloques del conversor digital – analógico.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Registros del módulo A/D

El módulo A/D tiene cinco registros:
Registro de control A/D ADCON0.
Registro de control A/D ADCON1.
Registro de control A/D ADCON2.
Registro de resultados alto (HIGH) A/D ADRESH.
Registro de resultados bajo (HIGH) A/D ADRESL.

REGISTRO ADCON0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Tabla 2.1. Distribución de bits del registro ADCON0.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

CHS3:CHS0: Bits selección del canal de conversión A/D (13 canales).
GO/DONE: Bit de inicio y de monitorización del estado de la conversión A/D:
- GO/DONE='0': Proceso de conversión parado.
- GO/DONE='1': Proceso de conversión en marcha.

ADON: Bit de habilitación del convertidor A/D
- ADON='0': Convertidor A/D desactivado.
- ADON='1': Convertidor A/D activado.

REGISTRO ADCON1

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Tabla 2.2. Distribución de bits del registro ADCON0.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

VCFG1: Bit de configuración de la tensión de referencia VREF-:
- VCFG1='0': VREF- se conecta a VSS
- VCFG1='1': VREF- se conecta a la línea física RA2

VCFG0: Bit de configuración de la tensión de referencia VREF+:
- VCFG1='0': VREF+ se conecta a VDD
- VCFG1='1': VREF+ se conecta a la línea física RA3

PCFG3:PCFG0: Bits configuración de los puertos de conversión A/D. Mediante estos bits se establecen las líneas físicas (RA5..RA0, RB4..RB0, RE1 y RE0) que van a trabajar como entradas del convertidor A/D.

Selección del canal de conversión

Para que uno de los 13 canales pueda ser seleccionado, previamente debe haber sido configurado como entrada analógica mediante los bits PCFG3:PCFG0 del registro ADCON1 (A: analógico / D: digital).

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Tabla 2.3. Bits de configuración para definir los puertos como digitales/analógicos.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Una vez configurado como línea de entrada analógica, un canal puede ser seleccionado mediante los bits CHS3:CHS0 del registro ADCON0.

CHS3	CHS2	CHS1	CHS0	CANAL SELECCIONADO
0	0	0	0	CANAL AN0 (RA0)
0	0	0	1	CANAL AN1 (RA1)
0	0	1	0	CANAL AN2 (RA2)
0	0	1	1	CANAL AN3 (RA3)
0	1	0	0	CANAL AN4 (RA5)
0	1	0	1	CANAL AN5 (RE0)
0	1	1	0	CANAL AN6 (RE1)
0	1	1	1	CANAL AN7 (RE2)
1	0	0	0	CANAL AN8 (RB2)
1	0	0	1	CANAL AN9 (RB3)
1	0	1	0	CANAL AN10 (RB1)
1	0	1	1	CANAL AN11 (RB4)
1	1	0	0	CANAL AN12 (RB0)
1	1	0	1	NO IMPLEMENTADO
1	1	1	0	NO IMPLEMENTADO
1	1	1	1	NO IMPLEMENTADO

Tabla 2.4. Bits de selección del canal analógico.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Rango de las tensiones de conversión

De acuerdo a los datos proporcionados por Microchip en el datasheet del PIC18F4550 el rango de las tensiones de conversión es de 0 a 5V. Para aumentar la resolución se puede modificar las tensiones de referencia acercando las tensiones de referencia máxima y mínima VREF+ y VREF- a los límites de variación de la señal que se desee digitalizar. Esto se consigue modificando las líneas RA2/AN2/ VREF+ y RA3/AN3/ VREF- como tensiones de referencia DEL CONVERTIDOR A/D, poniendo a 1 los bits VCFG1 y VCFG0 del registro ADCON1.

REGISTRO ADCON2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Tabla 2.5. Distribución de bits del registro ADCON2.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

ADFM: Bit de configuración del tipo de almacenamiento del resultado de la conversión en los registros ADRESH y ADRESL:

- ADFM='0': El resultado de la conversión se almacena con justificación a la izquierda. Se utiliza para conversión de 8 bits.
- ADFM='1': El resultado de la conversión se almacena con justificación a la derecha. Se utiliza para conversión de 10 bits. El byte superior se almacena en el registro ADRESH y los dos bits menos significativo en el registro ADRESL.

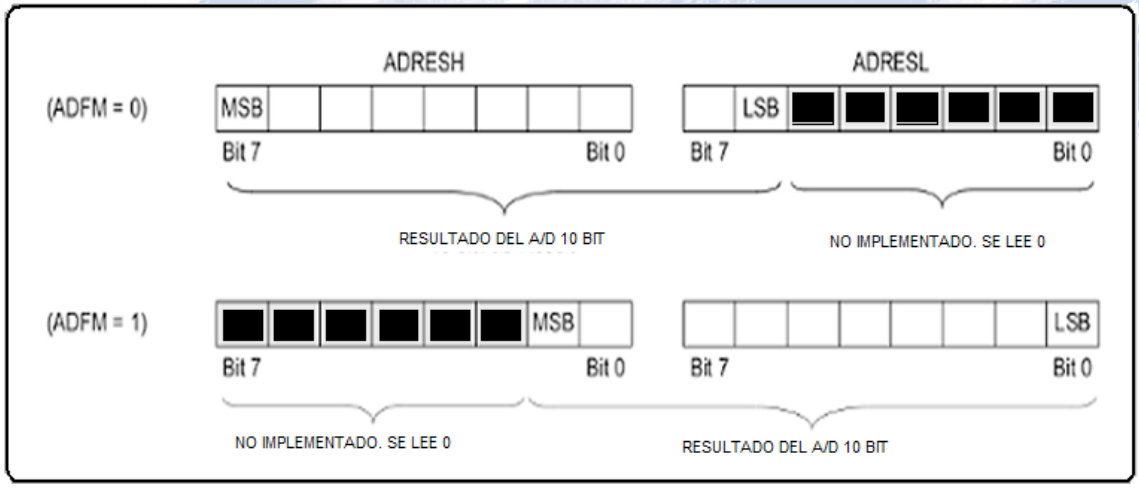


Figura 2.2. Ilustración de la función ADFM en los registros ADRESH y ADRESL.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

ACQT2:ACQT0: Bits de configuración del tiempo de adquisición.

TIEMPO DE ADQUISICIÓN (TAD)

El tiempo para convertir un bit se llama TAD. Según el fabricante, el tiempo de adquisición en los PIC de gama alta está entre:

0.8 us TAD 25 us, basado en el periodo del oscilador (TOSC) y VREF 3 V.

TAD = 1 us máximo, basado en módulo RC interno.

Cuando se utiliza el periodo del oscilador, se ve que el tiempo mínimo es 0.8 us. El TAD puede ser configurado con los bits ACQT2:ACQT0 para obtener diferentes valores del TAD como se presenta en la tabla 2.6.

ACQT2	ACQT1	ACQT0	TAD
1	1	1	20 TAD
1	1	0	16 TAD
1	0	1	12 TAD
1	0	0	8 TAD
0	1	1	6 TAD
0	1	0	4 TAD
0	0	1	2 TAD
0	0	0	0 TAD

Tabla 2.6. Modificación del TAD.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

ADCS2:ADCS0: Bits selección de la señal de reloj del convertidor A/D.

ADCS2	ADCS1	ADCS 0	F_{osc}
1	1	1	FRC
1	1	0	F_{osc} /64
1	0	1	F_{osc} /16
1	0	0	F_{osc} /4
0	1	1	FRC
0	1	0	F_{osc} /32
0	0	1	F_{osc} /8
0	0	0	F_{osc} /2

Tabla 2.7. Selección de la señal de reloj.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

En la tabla 2.7, FRC se refiere que utiliza el módulo del oscilador RC interno y F_{OSC} a la frecuencia del cristal externo.

Una forma práctica de obtener TAD y su correspondiente frecuencia de oscilación es mediante los datos de la tabla 2.8.

Fuente de reloj para el AD (TAD)		Asumido TAD mínimo 0.8 us
OPERACIÓN	ADCS2:ADCS0	Máxima Fosc
2 Tosc	000	2.50 MHz
4 Tosc	100	5.00 MHz
8 Tosc	001	10.00 MHz
16 Tosc	101	20.00 MHz
32 Tosc	010	40.00 MHz
64 Tosc	110	48.00 MHz
RC	x11	1.00 MHz

Tabla 2.8. TAD vs. Frecuencias de operación del dispositivo.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Cuando se utiliza el oscilador interno RC, viene prefijado y cumple con el valor mínimo.

Pasos para conversión

1. Configuración como canales A/D de las líneas que vayan a ser utilizadas (bits PCFG3:PCFG0 del registro ADCON1)
   - Configuración de las tensiones de referencia VREF+ y VREF- (bits VCFG0 y VCFG1 del registro ADCON1)
   - Configuración del reloj de conversión TAD (bits ADCS2:ADCS0 del registro ADCON2)
   - Configuración del tiempo de adquisición (bits ACQT2:ACQT0 del registro ADCON2)
   - Configuración del modo de almacenamiento de la conversión (bit ADFM del registro ADCON2)
2. Activación del conversor (bit ADON del registro ADCON0)
3. Selección del canal (bits CHS3:CHS0 del registro ADCON0)
4. Retardo de espera del tiempo de adquisición (solo en caso de no hacer uso del tiempo de adquisición automático).
5. Inicio de la conversión poniendo a 1 el bit GO/DONE del registro ADCON0.

6. Bucle de espera del final de conversión (comprobación del bit GO/DONE hasta que se ponga a 0).

7. Lectura del resultado de la conversión de los registros ADRESH y ADRESL

8. Procesamiento matemático del valor obtenido de la conversión analógica digital.

Funciones del CCS para el módulo AD

En el compilador CCS las funciones para manejar el convertidor AD son:

setup_adc(mode); donde:

mode, configura el módulo del conversor A/D correspondientes a los bits 7:6 del registro ADCON0. Los posibles argumentos se indican en la tabla 2.9.

Mode	ADCON0(1Fh)
ADC_OFF	ADC apagado
ADC_CLOCK_INTERNAL	ADC usa oscilador interno 32KHZ
ADC_CLOCK_DIV_2	ADC pre-escalar 2 (F_{OSC} /2)
ADC_CLOCK_DIV_8	ADC pre-escalar 8 (F_{OSC} /8)
ADC_CLOCK_DIV_32	ADC pre-escalar 32 (F_{OSC} /8)

Tabla 2.9. TAD vs. Frecuencias de operación del dispositivo.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

setup_adc_ports(value);

value: Definición de las entradas analógicas correspondientes a los bits 3:0 del ADCON1 (ver tabla 2.1).

set_adc_channel(channel);

Donde ***cannel,*** selecciona el canal analógico correspondiente a los bits 5:2 del registro ADCON0 (ver tabla 2.2).

Value=read_adc();

Lectura del resultado del ***adc,*** donde ***value*** es un entero de 16 bits. La directiva utilizada determina el número de bits del ADC. Por ejemplo #device 18F4550, debe incluir la indicación del número de bits del conversor, así: ***#device adc = 10.*** Esta directiva trabaja con los datos presentados en la tabla 2.10.

DEVICE	8 BIT	10BIT	11 BIT	16 BIT
ADC = 8	00-FF	00-FF	00-FF	00-FF
ADC = 10	-	0-3FF	-	-
ADC = 11	-	-	0-7FF	-
ADC = 16	0-FF00	0-FFC0	0-FFE0	0-FFFF

Tabla 2.10. Números de bits para la directiva *#device adc*.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

read_adc() permite tres modos de funcionamiento:

ADC_START_AND_READ	Inicio y lectura del convertidor. Opción por efecto.
ADC_START_ONLY	Sólo inicio la conversión.
ADC_READ_ONLY	Sólo lee los registros del conversor.

Tabla 2.11. Modos de funcionamiento para la función read_adc().
Fuente: Rangel. Microcontroladores PIC Gama Alta. Internet.

Ejercicio 2.1. ADC básico.

Para el siguiente ejercicio, se va a utilizar el conversor ADC para 10 bits. Por el canal AN0, se ingresa la señal analógica de 0 a 5 V. Los datos de conversión son indicados en un LCD. La lectura del LCD deberá estar entre 0 y 5 V. El potenciómetro RV1, sirve para cambiar el valor de la señal analógica. La figura 2.3, muestra el circuito indicado.

Definiciones usadas en CCS para el ADC

#device adc=10. Define utilizar el conversor para 10 bits.
setup_adc(ADC_CLOCK_INTERNAL). Usa el reloj interno para el ADC 32KHZ
set_adc_channel(0). Utiliza el canal 0 del ADC
value= read_adc(). Lee el valor de conversión. Se asigna a la variable "*value*".
Resolución del ADC. La resolución del ADC, se encuentra dado por el número de bits. El valor de la conversión se calcula con:

$$Resolución = V_{REF}/(2^N - 1)$$

Donde:
N, es el número de bita del conversor.
$V_{REF,}$ es el voltaje de referencia del conversor.

Ejemplo:

Cuál es la resolución de un ADC de 10 bits, para un voltaje de referencia de 5 V.

Resolución= $V_{REF}/(2^N - 1)$

Resolución= $5/(2^{10} - 1)$

Resolución= 0.004887586

Valor de la conversión. El valor equivalente del conversor podemos calcular con la fórmula:

valor_conversion= value *Resolución

valor_conversión= value*$V_{REF}/(2^N - 1)$

Donde, *value*, es el valor que lee del conversor y es equivalente al valor de la entrada analógica. Si la entrada analógica va de 0 a 5 V, con el ADC para 10 bits, la variable *value,* leerá el valor entre 0 y 1023.

En este caso si la entrada es 1 V, el valor de conversión será igual a 1, y así sucesivamente.

Supongamos que deseamos alterar el valor de la conversión en factor A, entonces la ecuación quedaría así:

valor_conversión= A*value*$V_{REF}/(2^N - 1)$

En este caso supongamos que A= 10, la resolución para 10 bits y la entrada analógica de 0 a 5. El valor de conversión será igual de 0 a 50. Con esto podemos ampliar o disminuir la escala, siendo esto muy útil para realizar medidores eléctricos.

Ejemplo: ADC básico. Medidor de 0 a 5 V.

PROGRAMA:

```
#include <18f4550.h>                                  //Librería para usar el PIC18F4550
#device adc=10                                        //Define el ADC para 10 bits.
#fuses XT,WDT,NOPROTECT,NOPUT,NOPBADEN //Configuración de fusibles.
#use delay (clock=4000000)                            //F_OSC =4MHz.
#include <lcd.c>                                      //Incluye el LCD.
#include <stdlib.h>                                   //Librería stdlib.h
#byte port_b = 0xF81                                  //Identifica el Puerto b.
void main(){                                          //Función principal main.
  float valor, valor_conversion;                      //Variables para la conversión.
  set_tris_b(0x00);                                   //Define el puerto B  como salida.
  port_b = 0x00;                                      //Puerto B reseteado.
  setup_adc_ports(AN0);                               //Usar entrada analógica AN0.
  setup_adc(ADC_CLOCK_INTERNAL);                      //Usar reloj interno para el ADC 32KHZ.
  set_adc_channel(0);                                 //Usar canal 0 del ADC.
  lcd_init();                                         //Inicializar LCD.

  while (TRUE)   {                                    //Bucle infinito.
        //El LED se prende y se apaga para indicar que el conversor está funcionando.
    output_low(PIN_b0);  //LED off.
    delay_ms(200);                                    //Retardo de 20 ms.
    output_high(PIN_b0);                              //LED on.
    delay_ms(200);                                    //Retardo de 200 ms.
    valor= read_adc();                                //Lee el adc y asigna a valor.
    valor_conversion= valor*5/1023;                   //Conversión matemática.
    lcd_gotoxy(1,1);                                  //Ubica el cursor en la posición 1,1 del LCD.
    printf(lcd_putc, "ADC=%6.3f",valor);              //Muestra el valor del conversor.
    lcd_gotoxy(1,2);                                  //Ubica el cursor en la posición 1,2 del LCD.
    printf(lcd_putc,"Voltaje=%6.3f",valor_conversion);        //Medida en voltios.
  }                                                   //Fin del bucle infinito.
}                                                     //Fin del main.
```

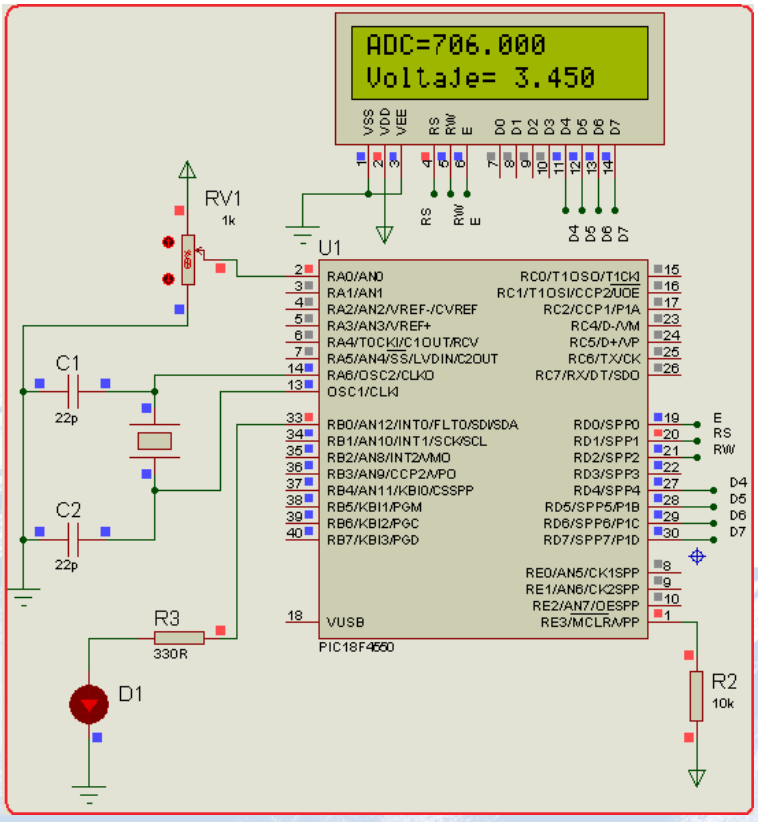


Figura 2.3. ADC básico.
Elaborado por: Los Autores.

Ejercicio 2.2. Control de teclado matricial con ADC.

Es posible controlar un teclado matricial mediante una sola línea del microcontrolador, con lo cual el ahorro de pines es muy significativo. Esto es necesario cunado se trabajan en proyectos donde el espacio físico y el cableado son factores que inciden en la selección del microcontrolador y se necesitaría, elegir uno de gama baja como los de la familia 12FXXX. El principio de funcionamiento se basa en generar diferentes niveles de voltaje al accionar las teclas, esto se consigue mediante divisores de tensión. Luego se envía esta tensión a la entrada analógica del ADC y dependiendo de los valores que proporcione el ADC, se asigna el valor correspondiente a la tecla pulsada. El circuito de la figura 2.4, muestra las conexiones para el manejo del teclado mediante el ADC. La tensión VA generada en la resistencia de referencia de 12K, se ingresa al canal 0, de la entrada analógica AN0 y el ADC, convierte a un valor digital proporcionando un rango adecuado se asigna al número respectivo. La tabla 2.12, indica los valores obtenidos para cada tecla (número) y se ha dado un rango de aproximadamente del ±1%

por la variación que las resistencias pueden tener al implementar en forma práctica el circuito. Es importante elegir las resistencias adecuadas, de tal forma que el voltaje entre cada columna del teclado sea diferente para que pueda generar el ADC, valores con márgenes distintos.

Tecla	Conversión del ADC	Valor asignado en el programa
1	877	867-887
4	818	808-828
7	753	743-763
*	694	684-704
2	620	610-630
5	590	580-570
8	556	546-566
0	523	513-533
3	491	481-501
6	472	462-482
9	450	440-460
#	428	418-438

Tabla 2.12. Valores del ADC y su equivalencia para los caracteres del teclado.
Elaborado por: Los Autores.

Como se ve en la tabla 2.12, cada número tiene valores diferentes, lo que garantiza la correcta visualización de los números ingresados. Por ejemplo, al pulsar el número 1, el ADC del micro genera en condiciones normales de las resistencias un número que está comprendido entre 867 – 887 (en el simulador 877), con lo cual haciendo una comparación se determina que es el número 1.

```
if ((valor >867)&(valor<887)){
lcd_putc("1");
```

Además se debe señalar que se ha utilizado una interrupción #*int_ad*, que se activa cada vez que finaliza la conversión, con lo cual se potencializa el programa y los recursos del PIC.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550
#device adc=10                                //Define el ADC para 10 bits.
#fuses HS,NOWDT,NOPROTECT,NOPUT, NOMCLR       //Configuración de fusibles.
#use delay (clock=12000000)                   //F_OSC =12MHz
#include <lcd.c>                              //Incluye librería del LCD
#include <stdlib.h>                           //Librería stdlib.h
long valor;                                   //Recibe dato del ADC
#int_ad                                       //Interrupción del ADC.
void ad_interrupt(){                          //Función interrupción del ADC.
     if ((valor >867)&(valor<887)){           //Si valor es mayor a 867 y menor a 887……
       lcd_putc("1");                         //……imprima en el LCD el 1.
     }
       if ((valor >610)&(valor<630)){         //……
       lcd_putc("2");
     }
     if ((valor >481)&(valor<501)){
       lcd_putc("3");
      }
     if ((valor >808)&(valor<828)){
       lcd_putc("4");
     }
     if ((valor >580)&(valor<600)){
       lcd_putc("5");
     }
     if ((valor >462)&(valor<482)){
       lcd_putc("6");
     }
     if ((valor >743)&(valor<763)){
       lcd_putc("7");
     }
     if ((valor >546)&(valor<566)){
       lcd_putc("8");
     }
     if ((valor >440)&(valor<460)){
       lcd_putc("9");
     }
     if ((valor >684)&(valor<704)){
       lcd_putc("*");
     }
     if ((valor >418)&(valor<438)){
       lcd_putc("#");
     }
     if ((valor >513)&(valor<533)){
       lcd_putc("0");
     }
  }
void main(){                                  //Función principal  main.
     LCD_INIT();                              //Inicializa el LCD.
     set_tris_a(0xff);                        //Configura el Puerto A como entrada.
     enable_interrupts(GLOBAL);               //Habilita las interrupciones.
     enable_interrupts(INT_AD);               //Habilita  la interrupción de fin de conversión.
     setup_adc_ports(AN0);                    //Utiliza el Puerto ANO.
     setup_adc(ADC_CLOCK_INTERNAL);           //Utiliza el reloj interno para el ADC.
     setup_adc_ports(ALL_ANALOG);             //Todas las entradas analógicas habilitadas.
     set_adc_channel(0);                      //Utiliza el canal 0 para leer el dato del ADC.
     lcd_putc("\fTECLA PULSADA ES\N");        //Mensaje en el LCD….
 while(true){                                 //Bucle infinito.
       valor=read_adc();                      //Asigna a "valor" el dato del ADC.
```

La figura 2.4, indica las conexiones del circuito utilizado para controlar un teclado matricial mediante el conversor ADC.

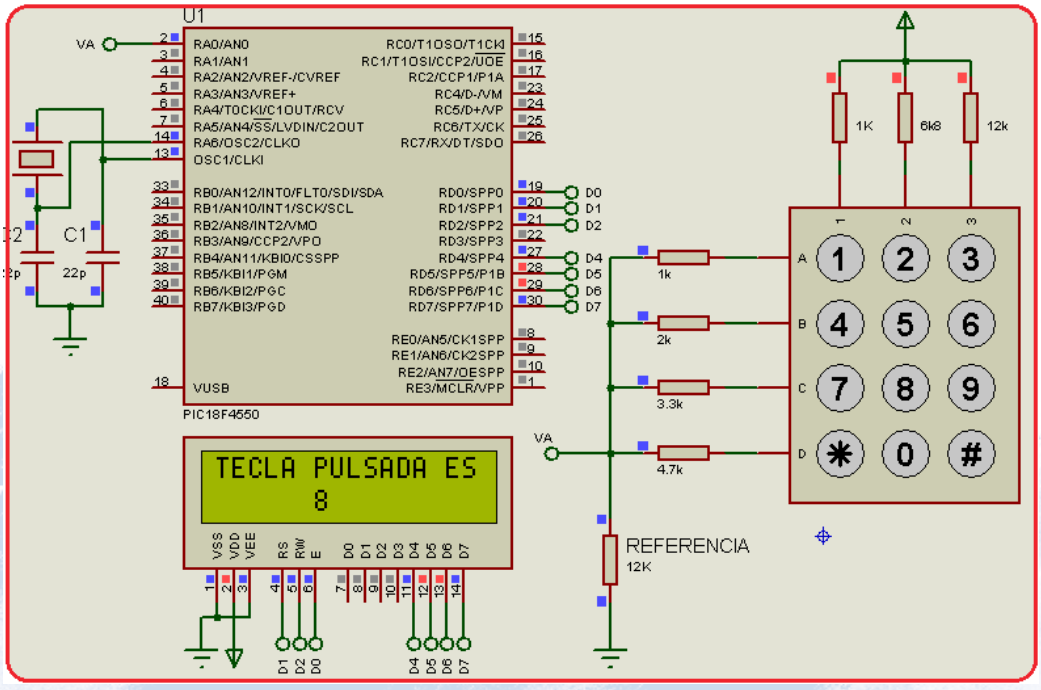


Figura 2.4. Conexiones de manejo matricial con línea del ADC.
Elaborado por: Los Autores.

MÓDULO CCP (CAPTURA/ COMPARACIÓN/ PWM)

El módulo CCP, tiene tres modos de funcionamiento:

Modo de Captura. Se utiliza para medir eventos externos como la duración de pulsos digitales.

Modo de Comparación. Genera señales digitales como temporizaciones programables. Es útil para control de etapas de potencia para convertidores DC/DC, DC/AC, AC/DC, AC/AC.

Modo PWM. Genera señales moduladas en ancho de pulso para control de motores DC, servomotores y motores paso a paso.

Los módulos CCP utilizan temporizadores 1, 2 o 3, dependiendo en el modo seleccionado. Timer1 y Timer3 están disponibles a los módulos de captura o en modos de comparación, mientras Timer2 está disponible para los módulos en el modo PWM. La tabla 2.13, muestra las interacciones entre los módulos CCP y los temporizadores

MODO CCP1	MODO CCP2	INTERACCIÓN

CAPTURA	CAPTURA	Cada módulo puede utilizar el TMR1 o TMR3 como la base de tiempo. La base de tiempo puede ser diferente para cada CCP.
CAPTURA	COMPARACIÓN	CCP1 puede ser configurado para disparador de eventos especiales para restablecer TMR1 o TMR3. En eventos de disparo automáticos, conversiones A/D, se pueden hacer. La operación del CCP1 podría verse afectada si se está utilizando el mismo temporizador como base de tiempo.
COMPARACIÓN	CAPTURA	CCP1 puede ser configurado para disparador de eventos especiales para restablecer TMR1 o TMR3 (dependiendo de utiliza la base de tiempo). La operación de CCP2 podría verse afectada si se está utilizando el mismo temporizador como base de tiempo.
COMPARACIÓN	COMPARACIÓN	Cualquier módulo puede ser configurado para disparador de eventos especiales para restablecer la base de tiempo. En eventos de disparo automáticos, conversiones A/D, se pueden hacer. Los conflictos pueden ocurrir si ambos módulos están utilizando la misma base de tiempo.
CAPTURA	PWM	Ninguno
COMPARACIÓN	PWM	Ninguno
PWM	CAPTURA	Ninguno
PWM	COMPARACIÓN	Ninguno
PWM	PWM	Ambos PWMs tendrán la misma frecuencia y velocidad de actualización (TMR2 interrupción).

Tabla 2.13. Interacciones entre CCP1 y CCP2 para los temporizadores.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

MODO PWM

El modo PWM (Pulse Width Modulation) o Modulación de Ancho de pulso, obtiene en los pines CCPx (CCP1, CCP2; pines RC1 y RC2 del puerto C) una señal periódica modificable el factor de trabajo (Duty Cycle). O sea, se puede variar el tiempo en alto (Ton), aumentando o disminuyendo el tiempo en bajo (Toff) de la señal. La figura 2.5, indica lo mencionado.

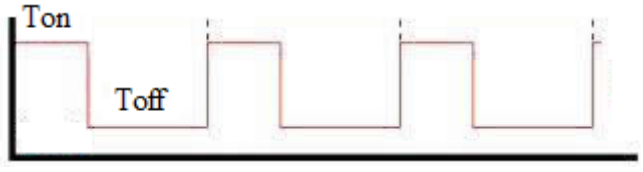


Figura 2.5. Señal PWM.
Elaborado por: Los Autores.

El período de la señal PWM se obtiene de configurar el TIMER2 y el registro PR2 (dirección 0x92).

El periodo de la señal PWM, $T_{PWM,}$ se puede calcular con la ecuación:

$$T_{PWM} = (1/F_{OSC})*4*t2div*(PR2+1)$$

$$F_{OSC} = \textit{Frecuencia del oscilador}$$

Dónde:

F_{OSC} es la frecuencia del oscilador del microcontrolador,

t2div el pre-escalar del TMR2 (1, 4 y 16),

PR2 es el valor del registro PR2 (0 a 255).

- TMR2 se borra.
- El pin CCPx se pone en alto, según el Duty Cycle.

El pin CCPx se pone en alto, según el Duty Cycle.

Si el ciclo de trabajo es mayor que el periodo de la onda PWM, la señal que sale del pin CCPx siempre será 1.

La resolución máxima en bits viene dada por la expresión:

$$resolucion = \frac{\log(\frac{Fosc}{Fpwm})}{\log 2}(bits)$$

Se puede trabajar con resoluciones de 8 y 10 bits. Para resoluciones de 10 bits el PR2 se toma de 10 bits, para valores menores del PR2 la resolución es de 8 bits.

El compilador C suministra varias funciones para el manejo del módulo CCP. Entre estas tenemos:

setup_ccpx(mode);

mode hace referencia a los bits ***CCPxM3:CCPxM0*** del registro ***CCPxCON***. Los posibles valores se indican en la tabla 2.14.

SETUP_CCPX(MODO)	MODO
CCP_OFF	Deshabilitación
CCP_CAPTURE_FE	Captura en flanco de bajada.
CCP_CAPTURE_RE	Captura en flanco de subida
CCP_CAPTURE_DIV_4	Captura tras 4 pulsos
CCP_CAPTURE_DIV_16	Captura tras 16 pulsos
CCP_COMPARE_SET_ON_MATCH	Salida a 1 en comparación.
CCP_COMPARE_CLR_ON_MATCH	Salida a 0 en comparación
CCP_COMPARE_INT	Interrupción en comparación
CCP_COMPARE_RESET_TIMER	Reset del TMR en comparación
CCP_PWM	Habilitación del PWM

Tabla 2.14. Funciones CCS para el módulo CCP.
Fuente: García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC.

Por ejemplo: *setup_ccp1(CCP_PWM);* configura el *CCP1* para *PWM*.

Ciclo de trabajo (DUTY CYCLE, DC)

Set_pwmx_duty(value);

Value dato de 8 o 16 bits, junto con el pre-escalar del TMR2 determina el ciclo de trabajo.

- Si ***value*** es LONG INT: *duty cycle = value / [4 * (PR2 +1)]*
- Si ***value*** es INT: *duty cycle = value / (PR2 +1)]*

La variable **long int** en ningún caso deberá ser un valor demasiado alto, ya que este valor determina el **Ton** (tiempo en alto de señal) y para un **Ton>T_{PWM}**, siempre tendríamos la señal del PWM en alto.

En todo caso, considerando que los valores máximos del PR2 son 256 y del pre escalar 16, para un DC del 100%, el máximo valor sería

*duty cycle = value / [4 * (PR2 +1)]*

*value = 1*4*(256) = 1024 (0 a 1023)*

value máximo = DC máximo = 1023.

En CCS, el TMR2 se configura en conjunto con el PR2, mediante la función:

setup_timer_2 (mode, period, postscale)

***mode*:** define el pre-escalar: T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16

***period**:* es un *int 0-255* que determina cuando el valor del reloj se resetea.

postscale es un número 1-16 que determina cuantos desbordes del temporizador, antes de una interrupción: (1 significa que una vez, 2 significa dos veces, y así sucesivamente).EscucharLeer fonéticamente

Por lo general, el **postscale** se define en 1, cualquier valor no afecta al periodo de la señal del PWM.

Ejemplo. Con un clock= 12000000 y un periodo= 255 determinar el periodo y la frecuencia del PWM para el pre-escalar de 1, 4 y 16.

$T_{PWM} = T_{OSC}*4*t2div*(PR2+1)$

Para t2div= 1:

$T_{PWM} = T_{OSC}*4*1*256 = T_{OSC}*1024$

$T_{PWM} = (1/12000000)*1024 =$ 85.33 us ó 11.72 kHz

Para t2div= 4:

$T_{PWM} = T_{OSC}*4*4*256 = T_{OSC}*4096$

$T_{PWM} = (1/12000000)*4096=$ 341.33 us o 2.92 kHz

Para t2div= 16:

$T_{PWM} = TOSC*4*16*256 = T_{OSC}*1024$

$T_{PWM} = (1/12000000)*16$ 384= 1365.33 us ó 732 Hz

Ejercicio: 2.3. Realizar un programa para obtener una señal cuadrada de 1 KHz usando CCP1.

El periodo PWM es igual a: $T_{PWM} = 1/ F_{PWM} = 1/1000$ Hz = 1 ms.

Utilizando la ecuación $T_{PWM} = (1/clock)*4*t2div*(PR2+1)$, obtenemos el PR2.

Para elegir un adecuado t2div se debe tener en cuenta dos criterios.

- El valor de la frecuencia del oscilador de MCU, y
- El valor de la frecuencia del PWM que se desea generar.

Si la frecuencia del oscilador es alta y la frecuencia del PWM es baja, necesariamente deberemos elegir un t2div de 16, para dividir lo máximo posible la frecuencia del reloj para el PWM.

En este ejercicio, la frecuencia del oscilador es alta F_{OSC}= 12MHZ y la frecuencia del PWM es baja, en consecuencia, se considera un pres-escalar t2div = 16.

T_{OSC}= 1/12MHZ= 83.33 ms.

PR2 = 1ms / [8,33ms*4*16]-1

PR2 = 186,5.

Se considera PR2= 187, considerando que es un dato tipo int.

Se calcula el ciclo de trabajo. Se asume que vamos a utilizar una variable tipo **int** para el valor del duty cycle. Por tanto, se aplica la ecuación**:**

duty cycle = value / (PR2 +1)

duty cycle = 0.5 (onda cuadrada).

value = 0.5*(187+1) = 94

Como estamos aproximando el valor del PR2, se recalcula el valor de la F_{PWM} exacta que se va a generar.

T_{PWM} = (1/clock)*4*t2div*(PR2+1)

T_{PWM} = 1/12MHz*4*16*(187+1)

T_{PWM} = 1.002 ms

F_{PWM} = 998.003 Hz.

Dando un error de 0.2%.

Cuando se tienen errores altos, se debería probar con oscilador más bajo y calcular con los pre-escalares distintos hasta determinar cuál valor se aproxima al requerido y se logre disminuir el error.

En el programa la instrucción **setup_ccp1(CCP_PWM);** permite configurar el módulo CCP1 como PWM y la instrucción **setup_timer_2(T2_DIV_BY_16, 187, 1);** define los valores del temporizador 2 o sea, el divisor de frecuencia o pre escalar (**T2_DIV_BY_16,**), el registro PR2(**187,**) y el post escalar (**1**).

PROGRAMA:

```
#include <18f4550.h>                  //Librería para usar el PIC18F4550
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN //Configuración de fusibles.
#use delay (clock= 12000000)          //F_OSC =12MHz.
#include <lcd.c>                      //Incluye el LCD.
#include <stdlib.h>                   //Librería stdlib.h

void main(void) {                     //Función principal main
    int value= 94;                    //Define el factor de trabajo a la mitad de la señal.
    setup_ccp1(CCP_PWM);              //Configure CCP1 como PWM.
    setup_timer_2(T2_DIV_BY_16, 187, 1); //Fija el divisor para 16, PR2 = 187 y un postscale de 1.
 while( TRUE ){
  set_pwm1_duty(value);
 }                                    //Fin del bucle.

        }                                        //Fin del main.
```

El circuito usado para la simulación se indica en la figura 2.6. Como carga se encuentra conectado un motor DC, utilizando un transistor que funciona como interruptor. El tipo de transistor dependerá de la corriente del motor DC.

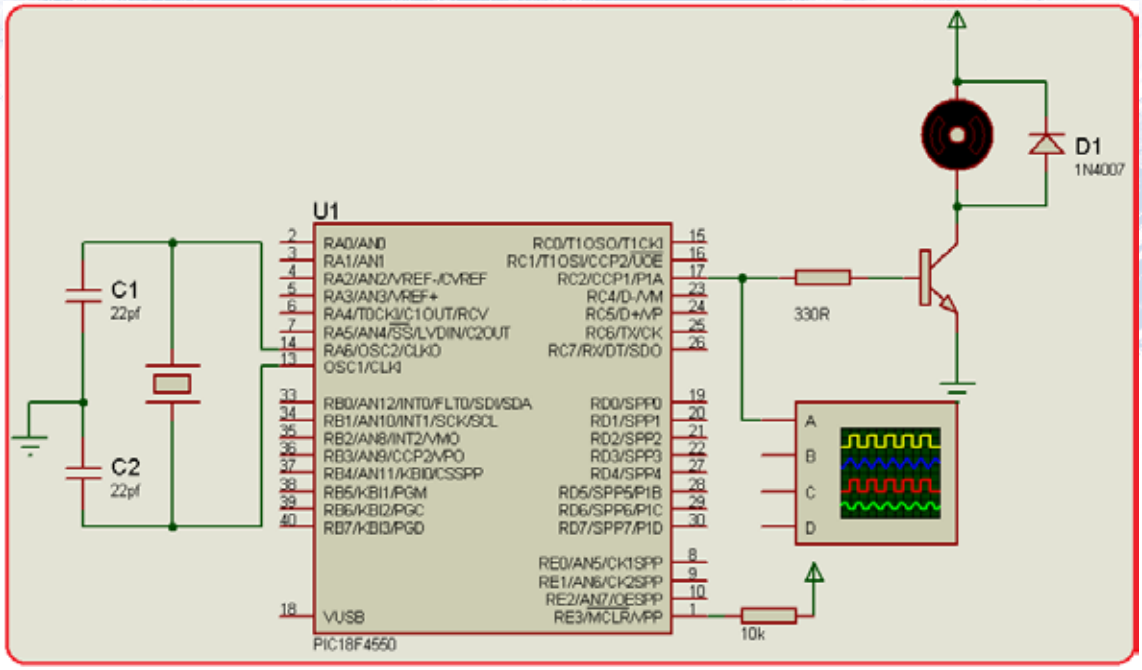


Figura 2.6. Generación de señal PWM por salida CCP1.

La figura 2.7, muestra una onda cuadrada de 1kHz (500 us en alto y un periodo de 1ms), obtenida en el osciloscopio virtual de ISIS.

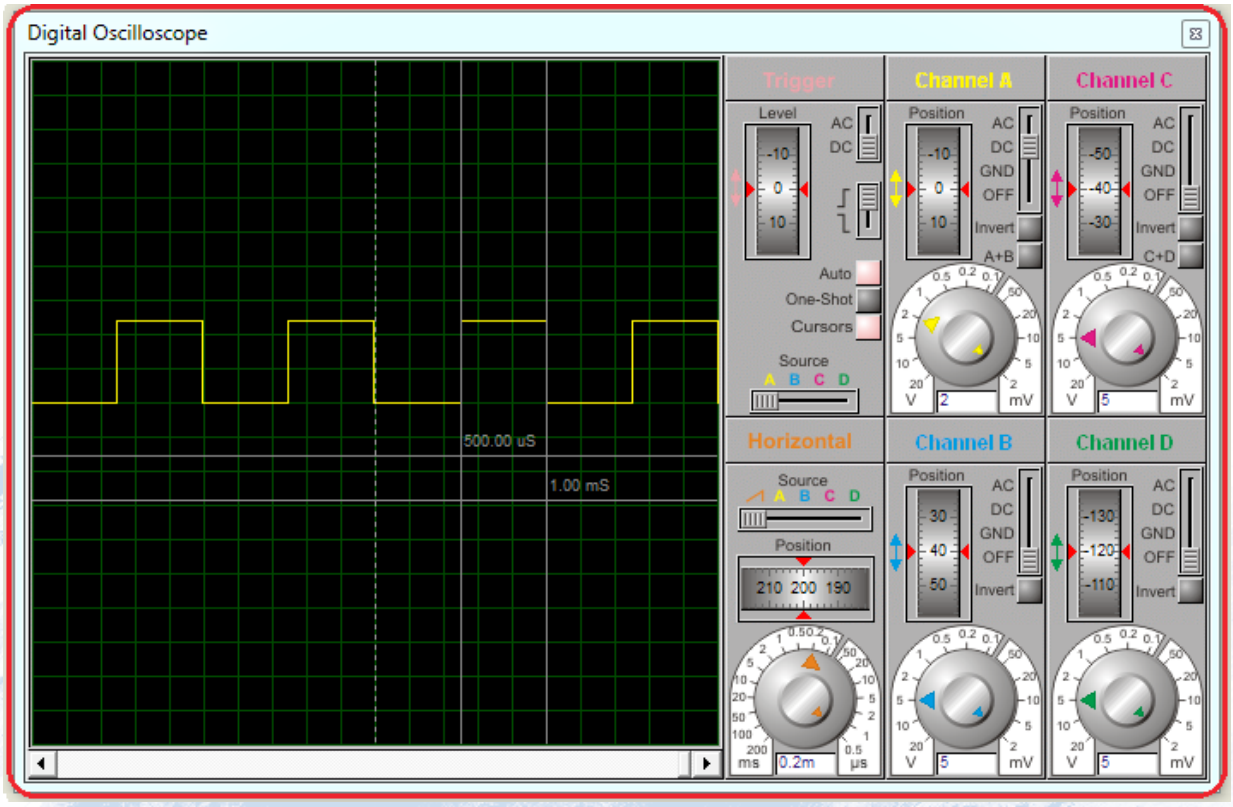


Figura 2.7. Onda cuadrada de 1kHz, obtenida en el osciloscopio virtual.
Elaborado por: Los Autores.

Modo captura

El modo captura, como su nombre lo indica, sirve para capturar eventos de subida o bajada en los pines RC2 y RC1 (puede cambiarse por RB3 mediante configuración).

En el modo de captura, la CCPRxH: CCPRxL el par de registros de captura se cargan con el valor de los registros de 16 bits del TMR1 o TMR3 produce un evento en el pin CCPx correspondiente. Un evento se define como una de las siguientes:

- cada flanco de bajada
- cada flanco ascendente
- cada 4 flancos ascendentes
- cada 16 flancos ascendentes.

Ejercicio 2.4. Medición del ancho de pulso de una señal.

Sea una señal cuadrada de frecuencia f. El ancho del pulso se debe tomar en el flanco ascendente y en el flanco descendente de la señal.

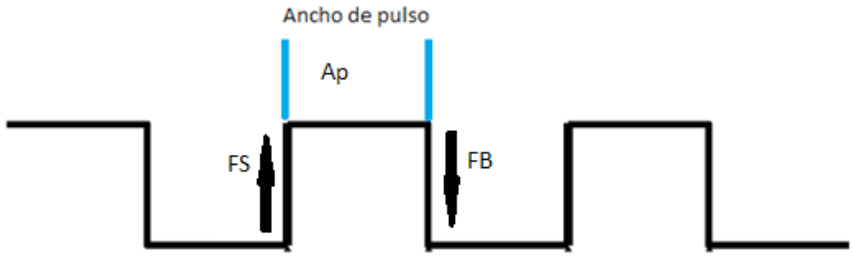


Figura 2.8. Ancho de pulso (Ap) y flanco de subida (FS) y flanco de bajada (FB) en una señal cuadrada.
Elaborado por: Los Autores.

Es aconsejable utilizar la interrupción por captura para realizar la medición del ancho de pulso.

En el programa, la instrucción ***setup_ccp1(CCP_CAPTURE_FE);*** *fija el módulo en flanco de bajada y para el flanco de subida se usa la instrucción* ***CCP_CAPTURE_RE.*** *Mediante la variable* ***cambio*** *se controla el cambio de los flancos (****cambio= 0****, flanco de subida y* ***cambio= 1****, flanco de bajada).*

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 Mhz.
#include <stdlib.h>                     //Librería stdlib.h.
#include <lcd.c>                        //Incluye el driver para manejar el LCD.
#byte PIR1 = 0Xf9E                      //Identificador del registro PIR1.

int16 TFB = 0, TFS = 0, TF = 0;          //Variables para tiempos de los flancos.
float Ap = 0.0;                         //Valor del ancho del pulso.
int1 Np = 0;                            //Ingresa un nuevo pulso.
int1 cambio =0;                         //Controla el cambio de flanco.

//Interrupción por captura.
#int_ccp1
void ccp1_int(){                        //Función para la interrupción.
  if (cambio ==0)                       //Flanco de subida.
    {
     TFS = CCP_1;                       //Carga el valor del registro CCPR1 en flanco de subida.
     setup_ccp1(CCP_CAPTURE_FE);        //Configuración en modo de captura en flanco de bajada.
     cambio = 1;                        //Control de cambio de flanco.
    }
   else
    {
     TFB = CCP_1;                       //Carga el valor del registro CCPR1 en flanco de bajada.
     setup_ccp1(CCP_CAPTURE_RE);        //Configuración en modo de captura en flanco de subida.
     cambio = 0;                        //Control de cambio de flanco
     if (Np==0)                         //Fin de pulso
      {
      NP = 1;                           //Nuevo pulso a medir
      }
    }
}

void main(void)                         //Función principal main
{
     lcd_init();                        //Inicializa el LCD
     setup_timer_1(T1_INTERNAL);        //Configura el TMR1 como temporizador.
              setup_ccp1(CCP_CAPTURE_RE);       //Configuración en modo de captura en flanco de subida.
    cambio = 0;                         //Detecta el cambio de flanco.
    enable_interrupts(global);          //Habilitación de las interrupciones globales.
    enable_interrupts(int_ccp1);        //Habilitación de la interrupción por captura.

  while(TRUE){                          //Inicia bucle infinito.
    if (NP == 1){                       //Si, hay pulso nuevo.
      TF=TFB - TFS;                     //Tiempo total = flanco ascendente – flanco descendente.
      Ap= TF/3 ;                        //Ancho de pulso en us, 1/3 us para 12 MHz[1] .
      lcd_gotoxy(1,1);                   //sitúa el cursor en 3 columna, 1 fila.
      lcd_putc("ANCHO DEL PULSO");
      printf(lcd_putc,"\nPulso= %6.1f uS", Ap); //Imprime el valor del ancho del pulso.
      Np = 0;                           //Listo para nuevo pulso.
    }                                   //Fin del IF.
  }                                     //Fin del bucle infinito.
}                                       //Fin del main.
```

Nota. Recuerde que el registro del TMR1 se incrementa cada 4 ciclos de la frecuencia del oscilador. Por tanto, el periodo de trabajo es (1/12 MHz)*4= 1/3 us.

La figura 2.9, muestra el diagrama del circuito utilizado para medir el ancho del pulso de una fuente externa que ingresa por el pin RC2 (CCP1) .

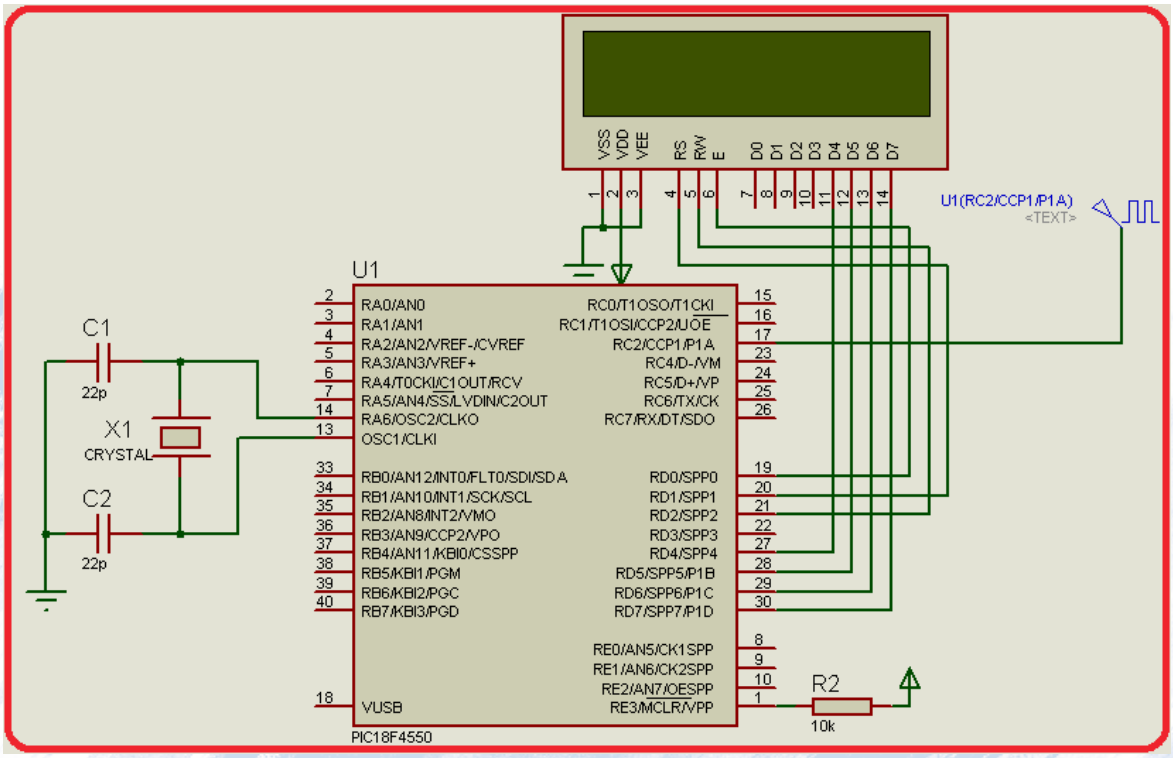


Figura 2.9. Medición del ancho del pulso de una señal externa utilizando el modo captura del módulo CCP.
Elaborado por: Los Autores.

Se aplicó una señal externa de periodo de 1ms, obteniendo como resultado 500 us (0.5ms), como se indica en la figura 2.10.

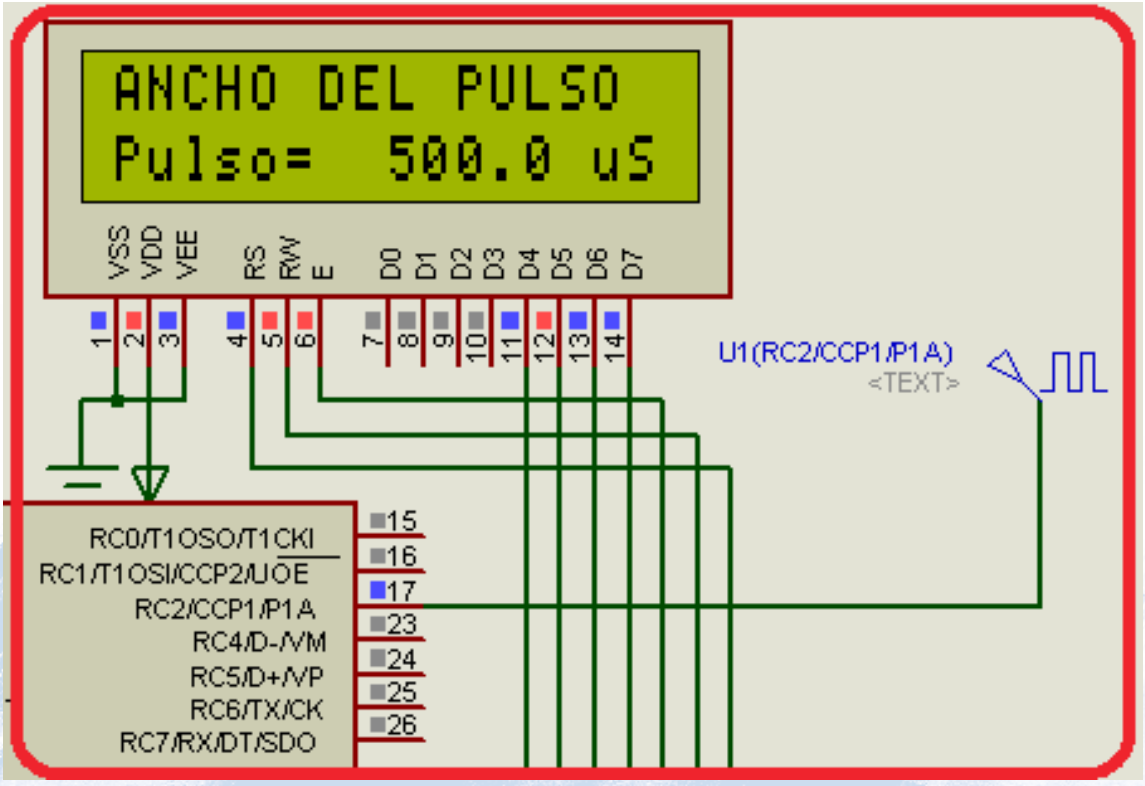


Figura 2.10. Medición del ancho de pulso de una señal cuadrada de 1kHz.
Elaborado por. Los Autores.

Modo comparación

El CCP en modo comparación, compara continuamente el valor del TIMER1 con el valor precargado en el registro CCPRx. Cuando los valores del TMR1 y CCPRx son iguales se producen los siguientes eventos:

- El pin RCy/CCPx se pone a 1.
- El pin RCy/CCPx se pone a 0
- Se genera una interrupción.
- Lanza una acción especial por hardware, consistente en:
- CCP1: Resetea TMR1.
- CCP2: Resetea TMR1 y lanza una conversión A/D (si está activado).

Ejercicio 2.5. Generación de una señal cuadrada de 1KHz mediante comparación del CCP1.

Considerando que se utiliza un oscilador de 12 MHz, calculamos el valor que se carga el registro CCP1.

El Periodo del reloj T_{OSC} = 1/12 MHz = 0.08333ms.

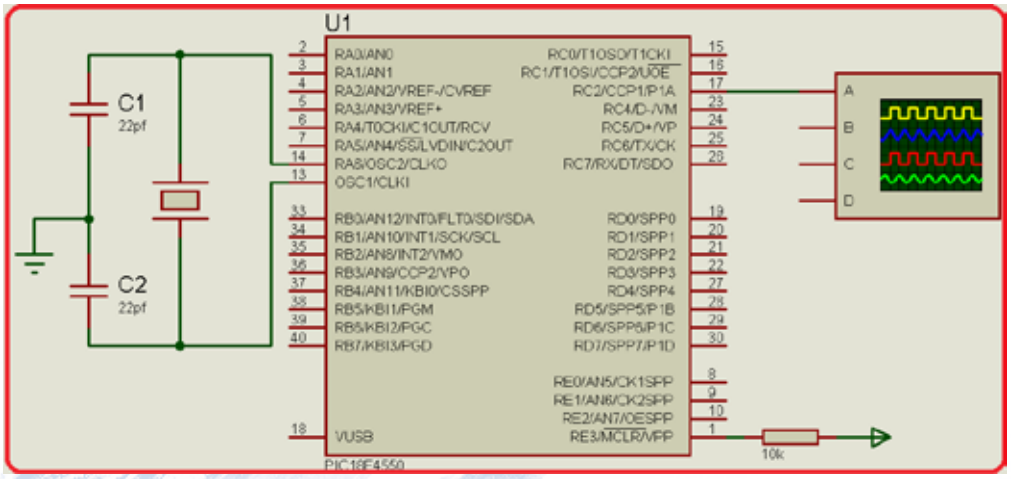
Como el TMR1 cambia cada 4 ciclos de reloj y si utilizamos un pre-escalar de 1, el registro TMR1 incrementará su valor cada: 0.333ms*4 = 0.333ms.

Dado que se requiere obtener una señal cuadrada de 1 kHz, el periodo total del CCP1 es: T_{CCP}= 1/1kHz= 1ms, o sea 0.5ms= 500us el tiempo en alto y el tiempo en bajo. Por tanto, considerando que cada pulso que recibe el TMR1 es cada 0.333ms, necesita de: 500us/0.333us = 1515 pulsos para completar los 500us. Entonces el CCP1 = 1515.

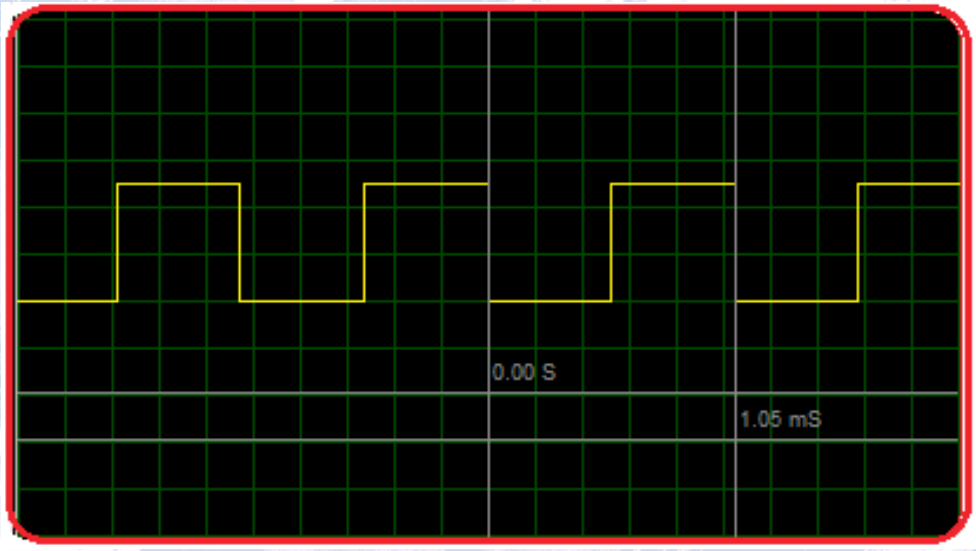
PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN         //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <stdlib.h>                     //Librería stdlib.h.
int1 estado=0;                          //Variable de control de cambio de estado de la señal.
#int_ccp1                               //Interrupción del CCP1.
void ccp1_int(){                        //Función de interrupción del CCP1.
  ++estado;                             //Incrementa en 1 la variable estado.
  if(estado==1){                        //Si estado =1.
    setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);       //Modo Comparación, fija a 0.
  }
else{
  setup_ccp1(CCP_COMPARE_SET_ON_MATCH);         //Modo Comparación, fija a 1.
 }
  set_timer1(0);                        //TMR1= 0
  CCP_1 = 1515;                         //Inicialización del registro CCPR1 para un Duty del 50%.
}
void main() {                           //Función principal main.
    disable_interrupts(global);         //Deshabilita las interrupciones globales.
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);       //Configuración TMR1.
    setup_ccp1(CCP_COMPARE_SET_ON_MATCH);       //Configuración inicial módulo CCP.
    CCP_1 = 1515;       //Inicialización del registro CCPR2 para un Duty Cycle del 50%.
    enable_interrupts(int_ccp1);        //Habilitación interrupción módulo CCP1.
    enable_interrupts(global);          //Habilitación interrupciones globales.
    while (TRUE){                       //Inicio del bucle infinito.
      //Otras instrucciones para que ejecute el MCU.
      }                                 //Fin del bucle infinito.
}                                       //Fin del main.
```

La figura 2.11 a, presenta el diagrama de conexiones para la generación de una señal mediante el CCP1 y en la figura 2.11 b, la señal obtenida en el osciloscopio virtual.



a. Uso del módulo CCP1, como comparación.



b. Onda cuadrada de 1 kHz.

Figura 2.11. Generación de una señal de 1kHz utilizando el módulo CCP1.
Elaborado por: Los Autores.

MÓDULO ECCP

El módulo CCP mejorado (Enhanced CCP) trabaja de forma similar al CCP estándar, con la diferencia que se puede generar hasta 4 PWM diferentes en los terminales RC2/P1A, RD5/P1B, RD6/P1C y RD7/P1D. Los módulos ECCP y CCP comparten los mismos temporizadores asociados, por lo que no es recomendable utilizar los dos módulos simultáneamente.

Para que las líneas RC2/P1A, RD5/P1B, RD6/P1C y RD7/P1D puedan trabajar en modo PWM mejorado es necesario configurarlas como líneas de salida (registros TRISC y TRISD).

Cuando alguna de estas líneas trabaja asociada al módulo ECCP, no está disponible en ninguna de sus otras funciones (líneas de I/O u otras funciones secundarias).

El modo PWM mejorado se puede utilizar para la generación de las señales de disparo de la etapa de potencia de convertidores DC/DC (control unidireccional y bidireccional de motores DC) y AC/AC (inversores, control de motores AC).

El ECCP tiene las siguientes características:

- Múltiple modos de salida.
- Polaridad seleccionable.
- Tiempo muerto programable.
- Auto-shutdown y auto-restart

El ECCP trabaja en modo comparador, captura o PWM, siendo este último el más utilizado por la serie de variantes que ofrece para generar PWM.

Configuración de la salida del PWM

El ECCP en modo PWM, tiene cuatro configuraciones de salida:

- Salida simple o PWM estándar.
- Salida medio Puente (Half-Bridge)
- Salida Puente complete (Full-Bridge) modo directo.
- Salida Puente complete (Full-Bridge) modo inverso.

La relación general de las salidas en todas las configuraciones para activación en alto del módulo ECCP, se resume en la tabla 2.15.

	ACTIVACIÓN EN ESTADO ALTO			
MODO ECCP	**P1A**	**P1B**	**P1C**	**P1D**
SIMPLE	MODULADA	I/O	I/O	I/O
HALF-BRIDGE	MODULADA	MODULADA	INACTIVA	INACTIVA
FULL-BRIDGE DIRECTO	ACTIVA	INACTIVA	INACTIVA	MODULADA
FULL-BRIDGE INVERSO	INACTIVA	MODULADA	ACTIVA	INACTIVA

Tabla 2.15. Activación en alto, estado de las salidas del módulo ECCP.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

La activación en alto significa que la señal modulada empieza en bajo y pasa a alto.

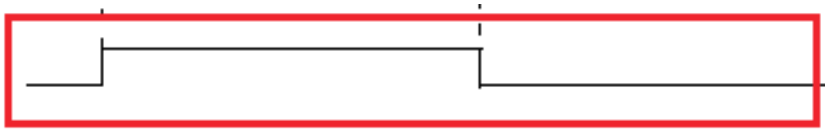


Figura 2.12. Señal modulada activa en alto.
Elaborado por: Los Autores.

La relación general de las salidas en todas las configuraciones para activación en bajo del módulo ECCP, se resume en la tabla 2.16.

ACTIVACIÓN EN ESTADO BAJO				
MODO ECCP	**P1A**	**P1B**	**P1C**	**P1D**
SIMPLE	MODULADA	I/O	I/O	I/O
HALF-BRIDGE	MODULADA	MODULADA	INACTIVA	INACTIVA
FULL-BRIDGE DIRECTO	ACTIVA	INACTIVA	INACTIVA	MODULADA
FULL-BRIDGE INVERSO	INACTIVA	MODULADA	ACTIVA	INACTIVA

Tabla 2.16. Activación en alto, estado de las salidas del módulo ECCP.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

La activación en bajo significa que la señal modulada empieza en alto y pasa a alto.

Figura 2.13. Señal modulada activa en bajo.
Elaborado por: Los Autores.

Funciones del ECCP en CCS

Para el manejo del módulo ECCP, el compilador CCS utiliza las mismas funciones del módulo estándar CCP, con lo cual se puede utilizar la mayor parte del módulo ECCP en forma transparente. La función principal es:

setup_ccp1 (mode); *o*

setup_ccp1 (mode, pwm)

Donde, el parámetro ***mode*** es una constante definida en el archivo ***.h*** del dispositivo. La tabla 2.17, muestra el resumen de los parámetros aceptados por ***mode:***

Mode	Descripción
CCP_PWM_H_H	Modo PWM: P1A, P1C active alto; P1B, P1D activo alto
CCP_PWM_H_L	Modo PWM: P1A, P1C active alto; P1B, P1D activo bajo
CCP_PWM_L_H	Modo PWM: P1A, P1C activo bajo; P1B, P1D activo alto.
CCP_PWM_L_L	Modo PWM: P1A, P1C activo bajo; P1B, P1D activo alto
CCP_PWM_FULL_BRIDGE	Puente completo directo.
CCP_PWM_FULL_BRIDGE_REV	Puente completo inverso.
CCP_PWM_HALF_BRIDGE	Medio puente.
CCP_SHUTDOWN_ON_COMP1	Desactivación en el cambio del Comparador 1.
CCP_SHUTDOWN_ON_COMP2	Desactivación en el cambio del Comparador 2.
CCP_SHUTDOWN_ON_COMP	Cambio de cualquier comparador 1 o comparador 2.
CCP_SHUTDOWN_ON_INT0	VIL en pin INT (RB0).
CCP_SHUTDOWN_ON_COMP1_INT0	VIL en pin INT o cambio en Comparador 1.
CCP_SHUTDOWN_ON_COMP2_INT0	VIL en pin INT o cambio en Comparador 2.
CCP_SHUTDOWN_ON_COMP_INT0	VIL en pin INT o cambio en Comparador 1 o 2.
CCP_SHUTDOWN_AC_L	Driver pines A y C en alto.
CCP_SHUTDOWN_AC_H	Driver pines A y C en bajo.
CCP_SHUTDOWN_AC_F	Driver pines A y C tri-estado.
CCP_SHUTDOWN_BD_L	Driver pines B y D en alto
CCP_SHUTDOWN_BD_H	Driver pines B y D en bajo
CCP_SHUTDOWN_BD_F	Driver pines B en D tri-estado
CCP_SHUTDOWN_RESTART	Reinicio del dispositivo después de un evento de desactivación.
CCP_DELAY	Uso de la banda del tiempo muerto.

Tabla 2.17. Parámetros opcionales mode para configurar el módulo ECCP en PWM mejorado.
Fuente: CCS C Compiler Manual PCD.

El parámetro **pwm** en la función ***setup_ccp1 (mode, pwm)*** es opcional para los chips que incluye módulo ECCP. Este parámetro permite ajustar el tiempo de apagado o tiempo muerto. El valor puede ser de 0-255. Más adelante se verá el uso de este parámetro.

Parámetros de control del PWM

PERIODO DEL PWM. El periodo del ECCP se puede calcular con la ecuación:

$$T_{PWM} = [(PR2) + 1] \cdot 4 \cdot T_{OSC} \cdot (\text{TMR2 Prescale Value})$$

CICLO DE TRABAJO (DUTY CYCLE). El ciclo de trabajo o **duty cycle** se calcula con la misma ecuación del PWM normal, es decir:

duty cycle = value / [4 • (PR2 +1)]

siendo, ***value*** un ***long int.***

Modo ECCP PWM estándar

En el modo estándar el módulo trabaja de manera similar al módulo CCP, utiliza el mismo pin CCP1, pudiendo utilizar la función ***setup_ccp1 (mode, pwm),*** para configurar el módulo. El parámetro PWM, no tiene efecto en este caso y la activación en alto o bajo no tiene ningún efecto en los valores del periodo o ciclo de trabajo, sino en la forma de generación inicial de la señal, como se explicó en el apartado 2.3.1. La señal modulada se obtiene por el pin P1A, y los pines P1B, P1C y P1D, pueden utilizarse como entradas o salidas normales.

Ejercicio 2.6. Escribir un programa para comprobar la operación en modo ECCP estándar. Mediante un pulsador seleccionar el modo de salida activo en alto o en bajo para obtener una señal modulada de 1 kHz con un DC del 50%.

El periodo de la señal modulada es: T_{PWM}= 1/1000 Hz= 1000 us.

Utilizando la ecuación del periodo de la señal modulada se calculará el valor del registro PR2, considerando un Pre escale de 4 y un oscilador de 4 MHz.

T_{PWM} = [(PR2) + 1] • 4 • T_{OSC} • (TMR2 Prescale)
1000us = [PR2 +1]*4*(0.25 us)*(4)
PR2= 1000/4 -1 = 250 -1
PR2= 249.
El ciclo de trabajo = 50%, por tanto, se calcula value:
duty cycle = value / [4 * (PR2 +1)]
value= DC* [4 * (PR2 +1)]
value= 0.5[4*250]
value= 500;

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#BYTE port_a= 0xF80                     //Identificador del Puerto A.
#define s bit_test(port_a,0)            //Identificador para seleccionar el modo de activación.
#use fast_io (A)                        //Usa la directiva para manejo i/o rápidas de los puertos
#use fast_io (B)                        //A, B, C, D.
#use fast_io (C)
#use fast_io (D)
long value;                             //Variable para el ciclo de trabajo.

void main()                             //Función principal main.
{
SET_TRIS_A(0XFF);                       //Puerto A como entrada.
SET_TRIS_B(0X00);                       //Puerto B como salida.
SET_TRIS_C(0X00);                       //Puerto C como salida.
SET_TRIS_D(0X00);                       //Puerto D como salida.
value= 250;                             //Valor del ciclo de trabajo equivale al 25%.
setup_timer_2(T2_DIV_BY_4, 249, 1);     //Configura el TMR2, para obtener 1kHz.

while(TRUE){                            //Inicio del bucle infinito.
        if (s==0){                      //Si s es 0, entonces:
          output_high (PIN_B0);         //Activa LED para indicar que está activo en alto.
          setup_ccp1(CCP_PWM_H_H);
        }
        else {
          output_LOW (PIN_B0);          //Apaga LED para indicar que está activo en bajo.
          setup_ccp1(CCP_PWM_L_L );
        }
 set_pwm1_duty(value);                  //50% duty cycle.
 }                                      //Fin del bucle.

        }                                       //Fin del main.
```

La figura 2.14, muestra las conexiones del microcontrolador y el osciloscopio virtual para la generación de la señal modulada PWM estándar utilizando el módulo EECP. El resultado correspondiente a la simulación se aprecia en la figura 2.15. Un diodo LED cambia de estado cuando se selecciona a través del pulsador el modo de activación en alto o en bajo.

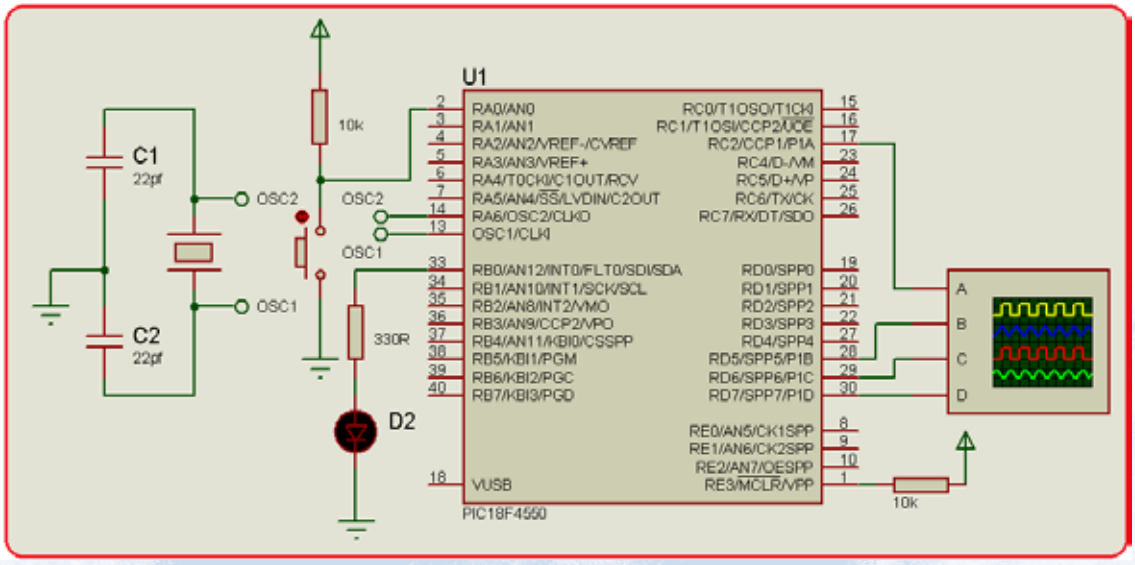


Figura 2.14. Circuito para funcionamiento del módulo ECCP como PWM estándar.
Elaborado por: Los Autores.

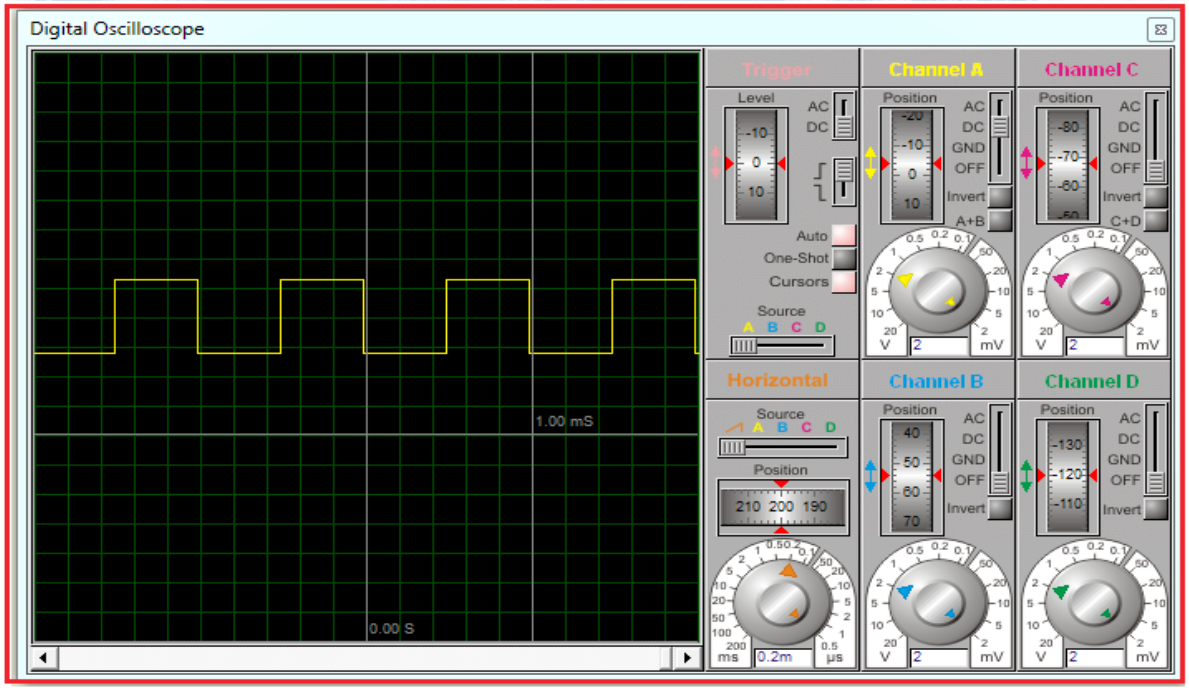


Figura 2.15. Señal de 1kHz del módulo ECCP funcionando en modo PWM estándar.
Elaborado por: Los Autores.

Modo PWM en HALF-BRIGDE

En el modo semi-puente el ECCP genera dos señales PWM complementarias por los pines RC2/P1A y RD5/P1B. Las señales PWM pueden configurarse para que sean activas por nivel alto o por nivel bajo. Se puede programar un tiempo muerto entre las transiciones de las señales con el fin de evitar cortocircuitos en la alimentación de la etapa de potencia.

Este modo se utiliza para controlar etapas de potencia en semi-puente o una etapa en puente completo donde los 4 transistores se modulan mediante señales PWM

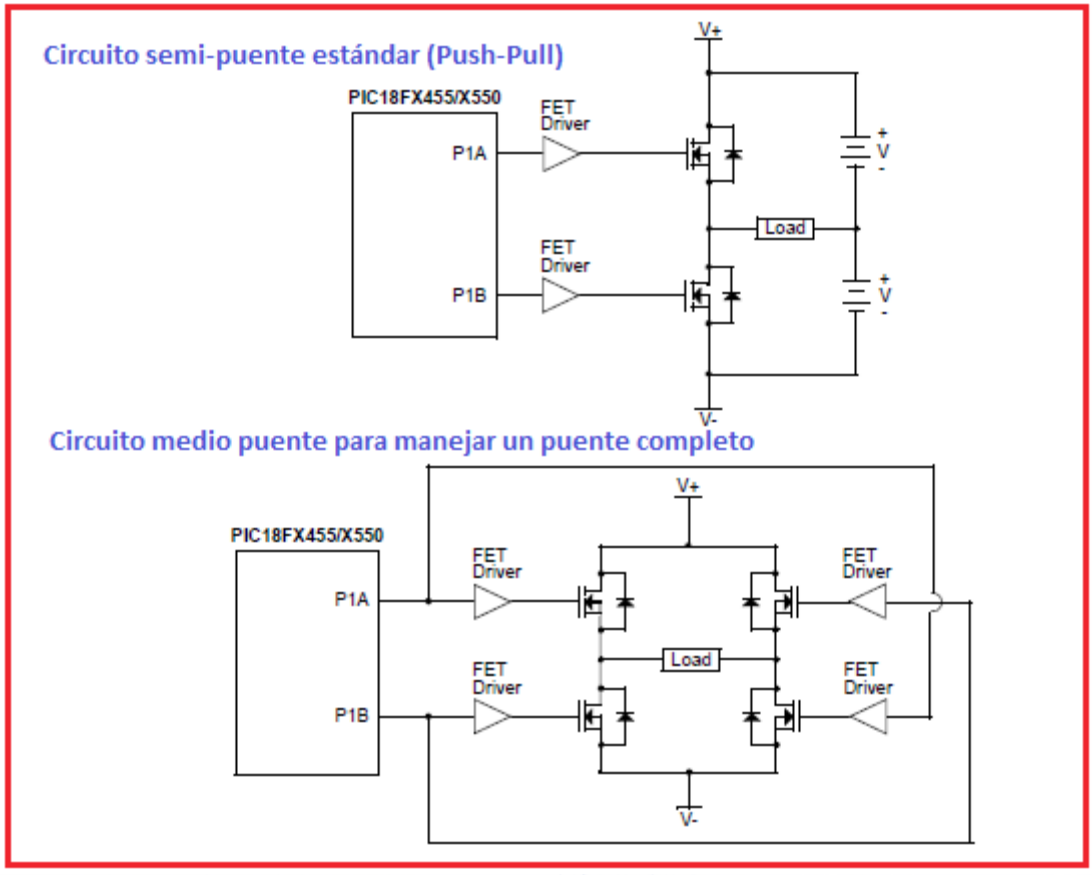


Figura 2.16. Ejemplos de aplicaciones del modo medio puente.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Ejercicio 2.7. Escribir un programa para comprobar la operación en modo HALF-BRIDGE. Mediante un pulsador seleccionar el modo de salida activo en alto o en bajo para obtener una señal modulada de 1 kHz con un DC del 25%.

El periodo de la señal modulada es: T_{PWM}= 1/1000 Hz= 1000 us.

Utilizando la ecuación del periodo de la señal modulada se calcula el valor del registro PR2, considerando un Prescale de 4 y un oscilador de 4 MHz.

T_{PWM} = [(PR2) + 1] • 4 • T_{OSC} • (TMR2 Prescale Value)

1000us = [PR2 +1]*4*(0.25 us)*(4)

PR2= 1000/4 -1 = 250 -1

PR2= 249.

El ciclo de trabajo = 25%, por tanto, se calcula value:

duty cycle = value / [4 * (PR2 +1)]

value= DC* [4 * (PR2 +1)]

value= 0.25[4*250]

value= 250;

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#BYTE port_a= 0xF80                     //Identificador del Puerto A.
#define s bit_test(port_a,0)            //Identificador para seleccionar el modo de activación.
#use fast_io (A)                        //Usa la directiva para manejo i/o rápidas de los puertos
#use fast_io (B)                        //A, B, C, D.
#use fast_io (C)
#use fast_io (D)
long value;                             //Variable para el ciclo de trabajo.

void main()                             //Función principal main.
{
SET_TRIS_A(0XFF);                       //Puerto A como entrada.
SET_TRIS_B(0X00);                       //Puerto B como salida.
SET_TRIS_C(0X00);                       //Puerto C como salida.
SET_TRIS_D(0X00);                       //Puerto D como salida.
value= 250;                             //Valor del ciclo de trabajo equivale al 25%.
setup_timer_2(T2_DIV_BY_4, 249, 1);     //Configura el TMR2, para obtener 1KHZ.

while(TRUE){                            //Inicio del bucle infinito.
        if (s==0){                      //Si s es 0, entonces:
          output_high (PIN_B0);         //Activa LED para indicar que está activo en alto.
          setup_ccp1(CCP_PWM_HALF_BRIDGE | CCP_PWM_H_H);
        }
        else {
          output_LOW (PIN_B0);          //Apaga LED para indicar que está activo en bajo.
          setup_ccp1(CCP_PWM_HALF_BRIDGE | CCP_PWM_L_L );
        }
 set_pwm1_duty(value);                  //25% duty cycle.
}                                       //Fin del bucle.
}                                       //Fin del main.
```

Un pulsador conectado en el puerto RA0, permite cambiar de bajo a alto, como se indica en la figura 2.17.

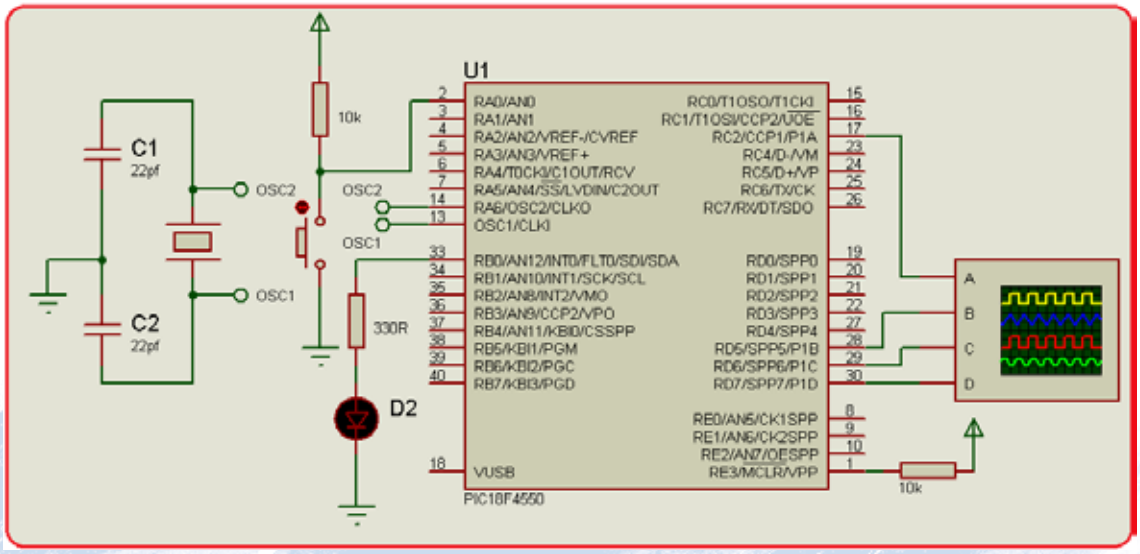


Figura 2.17. Circuito para generar la señales PWM mejorado en activo en alto o bajo.
Elaborado por: Los Autores.

Las figuras 2.18 y 2.19, presentan los resultados de las señales obtenidas en el osciloscopio virtual para los puertos P1A, P1B, P1C y P1D. Observe la señales moduladas son entregadas por los pines P1A y P1B y los pines P1C y P1D están desactivas.

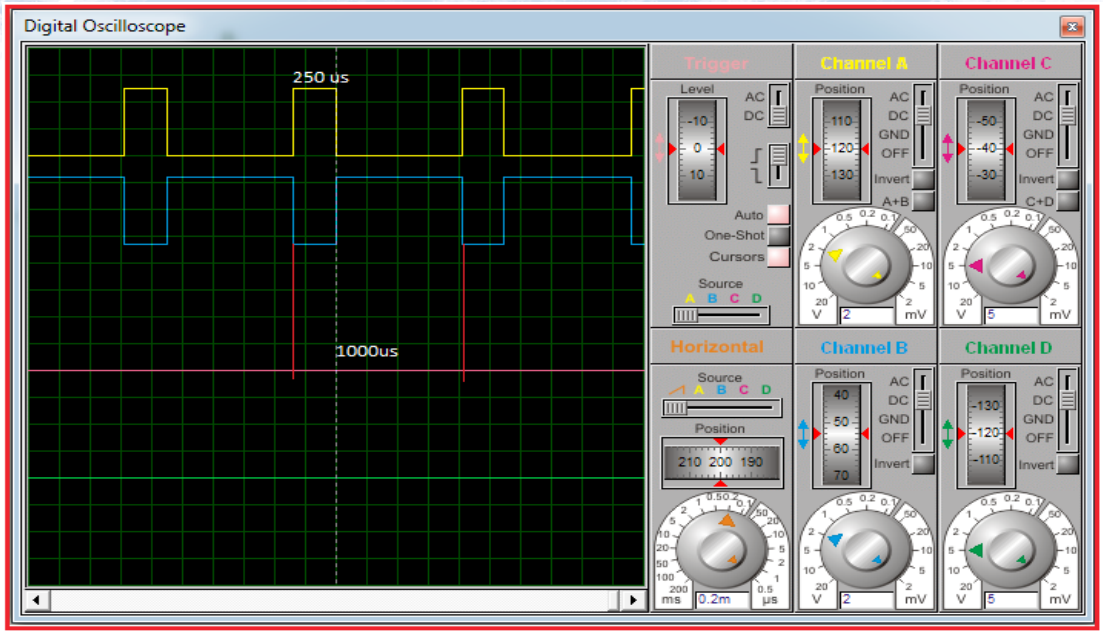


Figura 2.18. Ondas moduladas en P1A y P1B de modo HALF-BRIDGE. Activación en alto.
Elaborado por los autores.

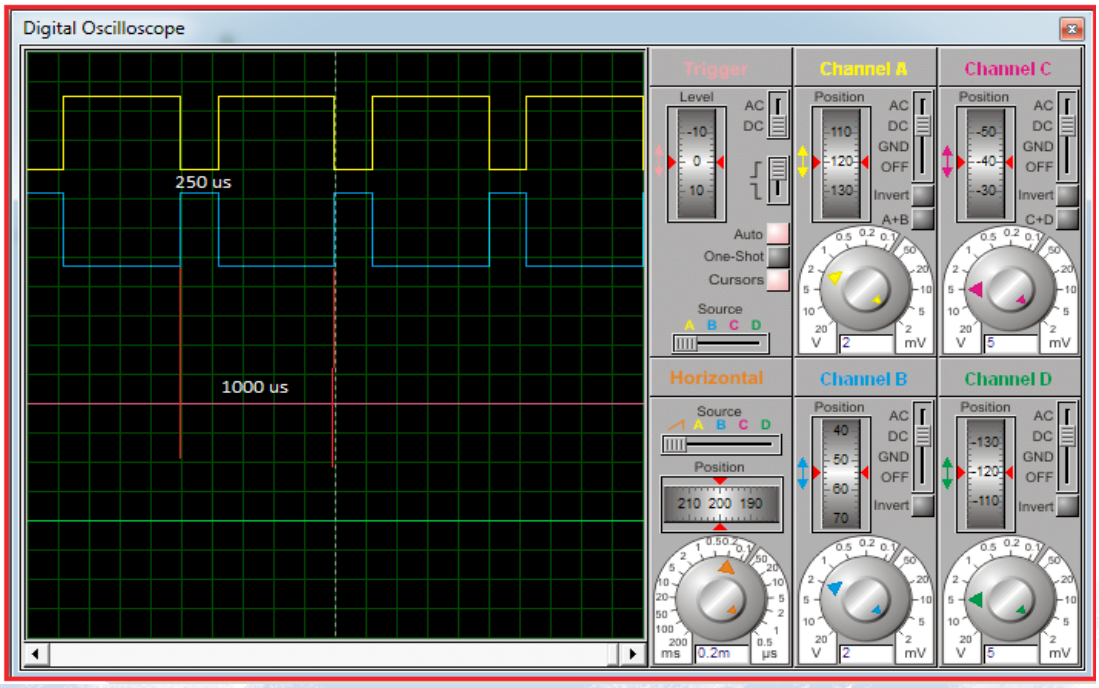


Figura 2.19. Ondas moduladas en P1A y P1B de modo HALF-BRIDGE. Activación en bajo.
Elaborado por: Los Autores.

Tiempo muerto (dead band delay). En el modo de salida Medio Puente, debido que las señales moduladas son invertidas y al manejar un circuito semipuente, puede ocurrir que entre en conducción simultáneamente los dos dispositivos y provocar una corriente que puede destruir a los dispositivos, como se indica en la figura 2.20.

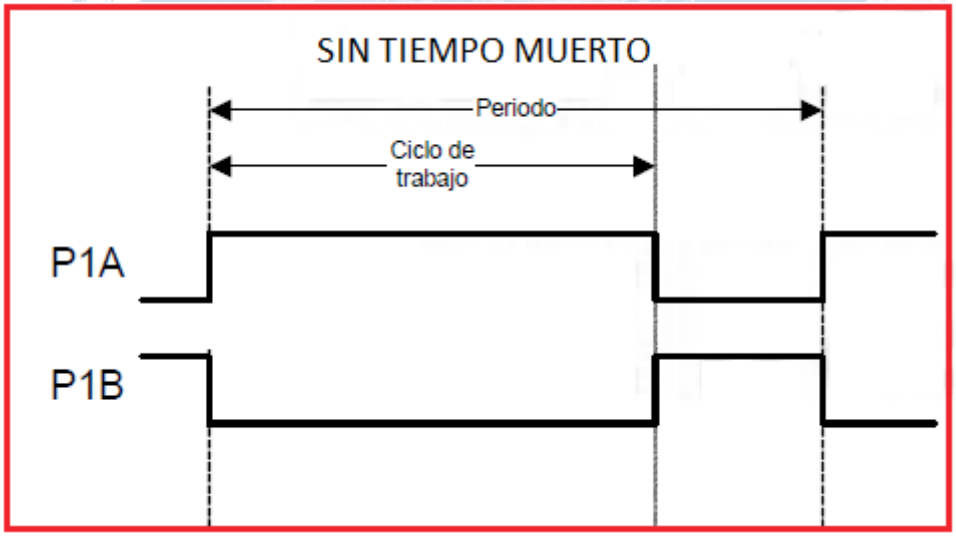


Figura 2.20. Señales moduladas de un circuito HALF-BRIDGE.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

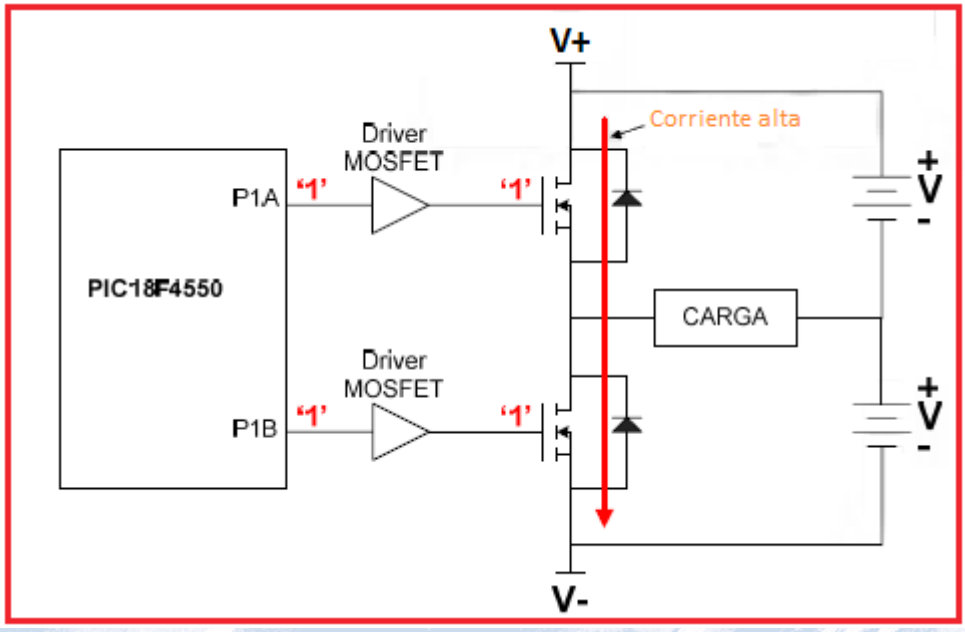


Figura 2.21. Corriente alta simultánea en los dispositivos del semipuente.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Para evitar estas corrientes de disparo, se puede programar la banda de tiempo muerto. En este caso, se da un tiempo antes de la activación de uno de los dispositivos de salida, como se indica en la figura 2.22.

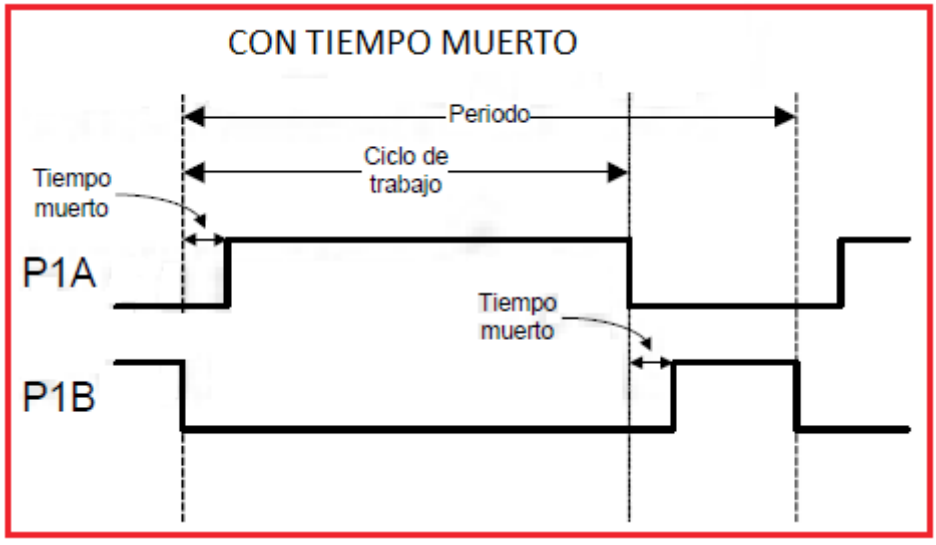


Figura 2.22. Señales en el circuito HALF-BRIDGE con tiempo muerto.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

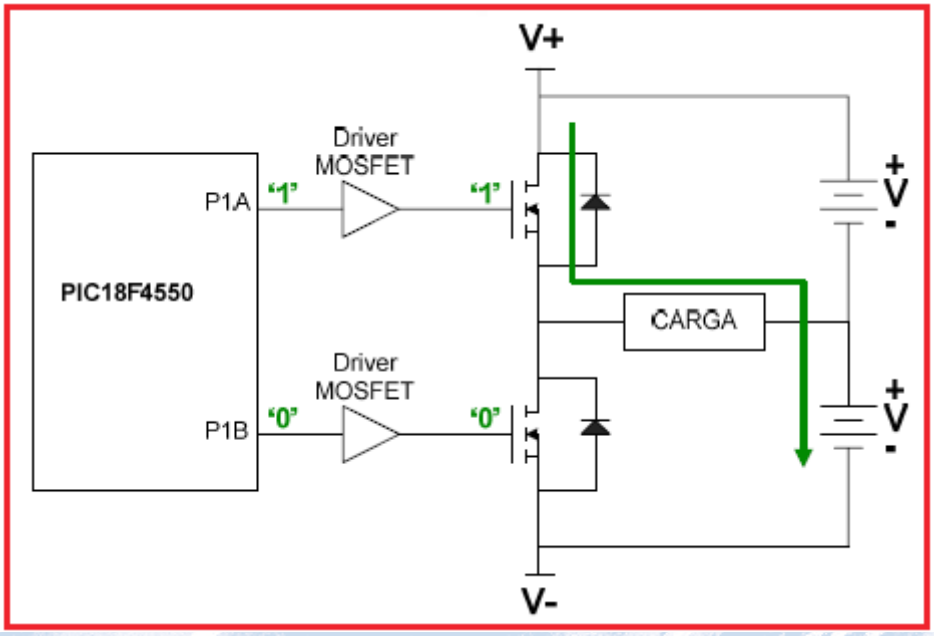


Figura 2.23. Corriente en los dispositivos del semipuente usando tiempo muerto.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

En CCS, para calcular el tiempo muerto se utiliza la ecuación:

$$Delay = 4 \bullet T_{osc} \bullet PWM$$

El parámetro **pwm** en la función ***setup_ccp1 (mode, pwm)*** permite ajustar el tiempo de apagado o tiempo muerto. El valor puede ser es un int8 (0-255).

Ejercicio 2.8. Escribir un programa para la operación en modo HALF-BRIDGE on tiempo muerto. Mediante un pulsador seleccionar el modo de salida activo en alto o en bajo para obtener una señal modulada de 1 kHz con un DC del 50% y un tiempo muerto de 100us.

Los valores del PR2 y el ciclo de trabajo son similares a los ejercicios anteriores, solo falta calcular el delay o tiempo muerto de la señal. Aplicando la ecuación del tiempo muerto el parámetro PWM es:

PWM= Delay/4*T_{OSC}

PWM= 100us /4*0.25

PWM= 100us.

El valor del ciclo de trabajo será:

Value= 0.50*[250]*4

Value= 500

Si el tiempo muerto es delay= 25 us, el valor del PWM para la función de apagado es:

Delay= 4*TOSC*PWM

PWM= Delay / 4*TOSC

PWM= 25 us /4*0.25us

PWM= 100.

Con estos valores se escribe el programa.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#BYTE port_a= 0xF80                     //Identificador del Puerto A.
#define s bit_test(port_a,0)            //Identificador para seleccionar el modo de activación.
#use fast_io (A)                        //Usa la directiva para manejo i/o rápidas de los puertos
#use fast_io (B)                        //A, B, C, D.
#use fast_io (C)
#use fast_io (D)
long value;                             //Variable para el ciclo de trabajo.

void main()                             //Función principal main.
{
SET_TRIS_A(0XFF);                       //Puerto A como entrada.
SET_TRIS_B(0X00);                       //Puerto B como salida.
SET_TRIS_C(0X00);                       //Puerto C como salida.
SET_TRIS_D(0X00);                       //Puerto D como salida.
value= 500;                             //Valor del ciclo de trabajo equivale al 50%.
setup_timer_2(T2_DIV_BY_4, 249, 1);     //Configura el TMR2, para obtener 1KHZ.

while(TRUE){                            //Inicio del bucle infinito.
        if (s==0){                      //Si s es 0, entonces:
          output_high (PIN_B0);         //Activa LED para indicar que está activo en alto.
          setup_ccp1(CCP_PWM_HALF_BRIDGE | CCP_PWM_H_H,100);
        }
        else {
          output_LOW (PIN_B0);          //apaga LED para indicar que está activo en bajo.
          setup_ccp1(CCP_PWM_HALF_BRIDGE | CCP_PWM_L_L,100);
        }
set_pwm1_duty(value);                   //50% duty cycle.
 }                                      //Fin del bucle.
}                                       //Fin del main.
```

El resultado de la simulación se indica en la figura 2.24, en la misma se tiene un periodo del PWM de 1.04 ms próximo al 1 ms y un ciclo de trabajo de 500 us. Como también se observa en la figura el tiempo muerto y su valor de 100 us, como se calculó al inicio del ejercicio.

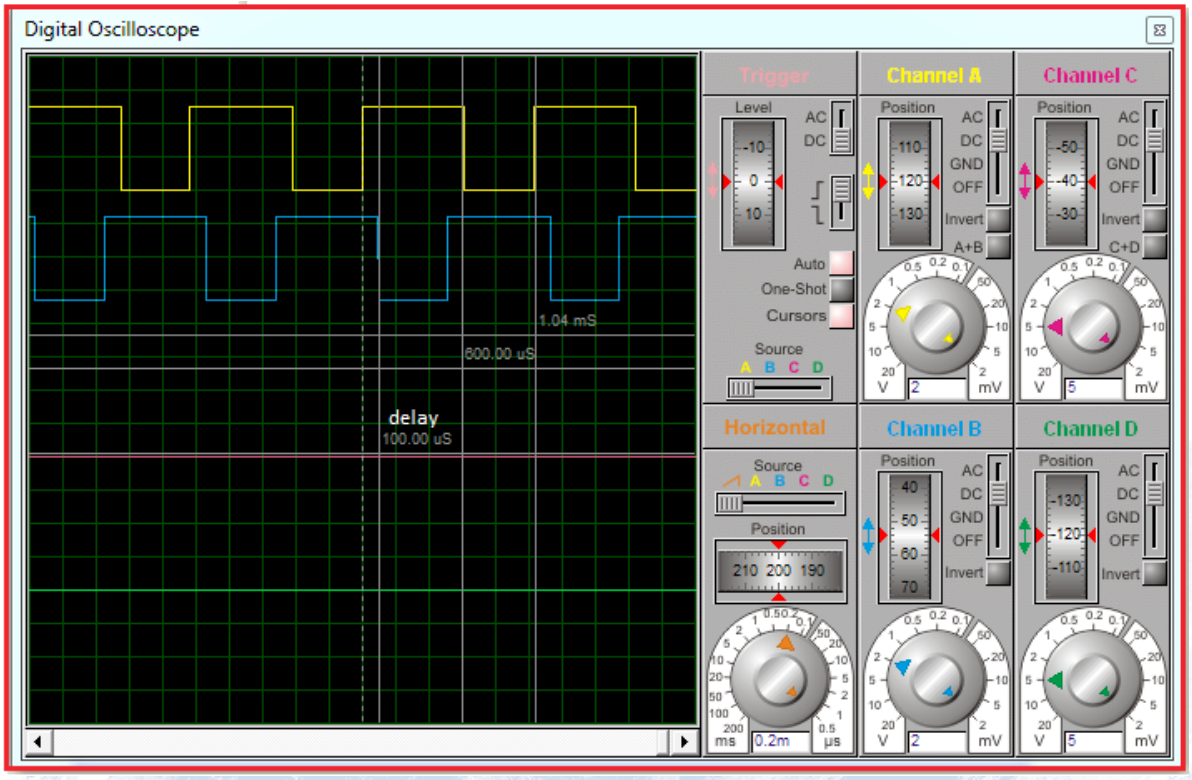


Figura 2.24. Resultado de la simulación para PWM mejorado en HALF-BRIDGE y tiempo muerto.
Elaborado por: Los Autores.

Modo PWM en FULL-BRIDGE

En el modo puente completo el ECCP funcionando en combinación modo directo o inverso puede generar hasta 4 señales de disparo que pueden ser utilizadas para activar una etapa de potencia en puente completo por los pines RC2/P1A, RD5/P1B, RD6/P1C y RD7/P1D.

Como se indicó en la tabla 2.17, el compilador CCS, tiene varios modos que permiten controlar al modo FULL-BRIDGE.

La figura 2.25, indica un ejemplo típico para aplicación FULL-BRIDGE que Microchip sugiere en sus aplicaciones de control circuitos puentes con sus microcontroladores.

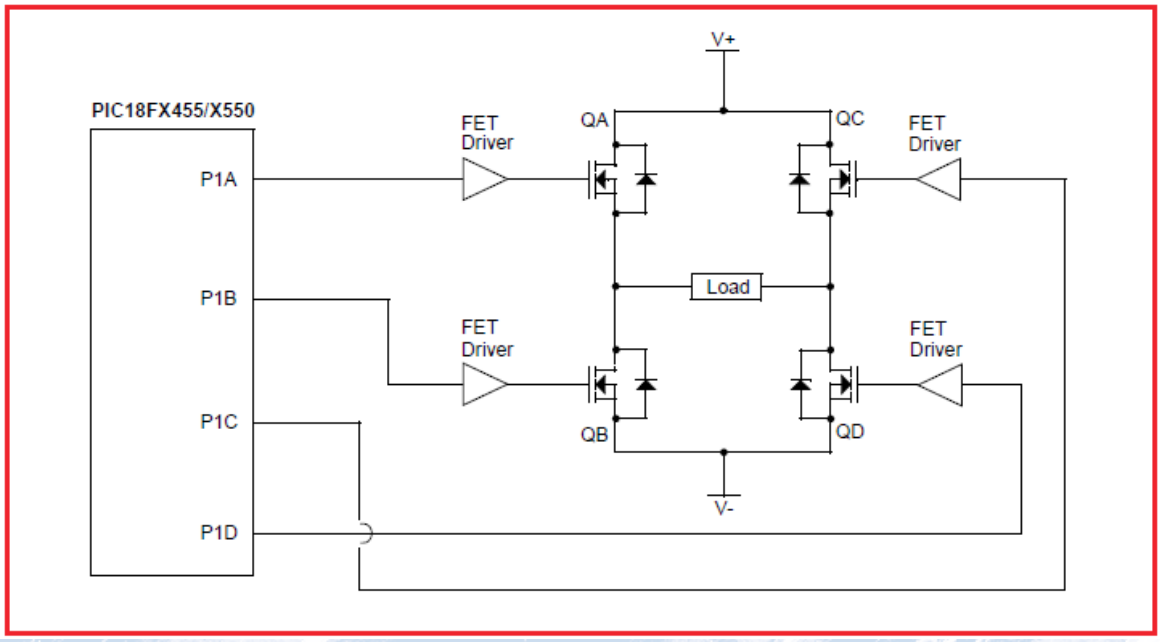


Figura 2.25. Circuito de aplicación de FULL-BRIDGE.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Ejercicio 2.9. Escribir un programa para generar una señal modulada de 250HZ. Al accionar un pulsador cambiar el modo activación de alto a bajo o viceversa. Usar un duty cycle del 50%.

El lector puede comprobar para un pre-escalar de 16, una frecuencia del oscilador de 4 MHZ, el PR2 es 249 y value es 500 del DC.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#BYTE port_a= 0xF80                     //Identificador del Puerto A.
#define s bit_test(port_a,0)            //Identificador para seleccionar el modo de activación.
                                        //Usa la directiva para manejo i/o rápidas de los puertos
#use fast_io (B)                        //A, B, C, D.
#use fast_io (C)
#use fast_io (D)
long value= 500;                        //Variable para el ciclo de trabajo, equivale al 50%.

void main(){                            //Función principal main.
SET_TRIS_B(0X00);                       //Puerto B como salida.
SET_TRIS_C(0X00);                       //Puerto C como salida.
SET_TRIS_D(0X00);                       //Puerto D como salida.
setup_timer_2(T2_DIV_BY_16,249, 1);     //Configura el TMR2.
enable_interrupts ( INT_Timer2 );       //Habilitación de la interrupción del TMR2.
enable_interrupts ( GLOBAL);            //Habilitación Global de las interrupciones.

while (TRUE){                           //Inicio del bucle infinito.
   if (s ==0){
          //Configura el Puente FULL-BRIDGE activo en alto modo directo.
     setup_ccp1(CCP_PWM_FULL_BRIDGE | CCP_PWM_H_H);
     output_high (PIN_B0);              //Activa LED de señalización.
   }
   else{
          //Configura el Puente FULL-BRIDGE activo en alto modo indirecto.
     setup_ccp1(CCP_PWM_FULL_BRIDGE_REV | CCP_PWM_H_H);
     output_low (PIN_B0);                           //Apaga LED.
  }
  set_pwm1_duty(value);                             //50% duty cycle
 }                                                  //Fin del bucle.
}                                                   //Fin del main.
```

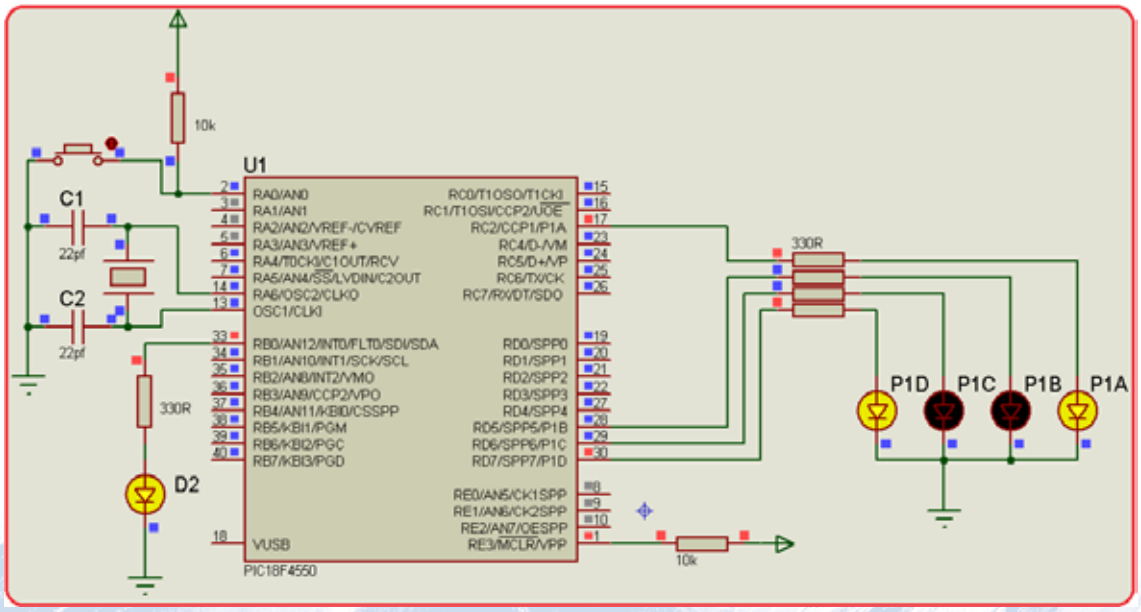


Figura 2.26. Operación FULL-BRIDGE, activo en alto modo directo. P1A activada y P1D señal modulada.
Elaborado por: Los Autores.

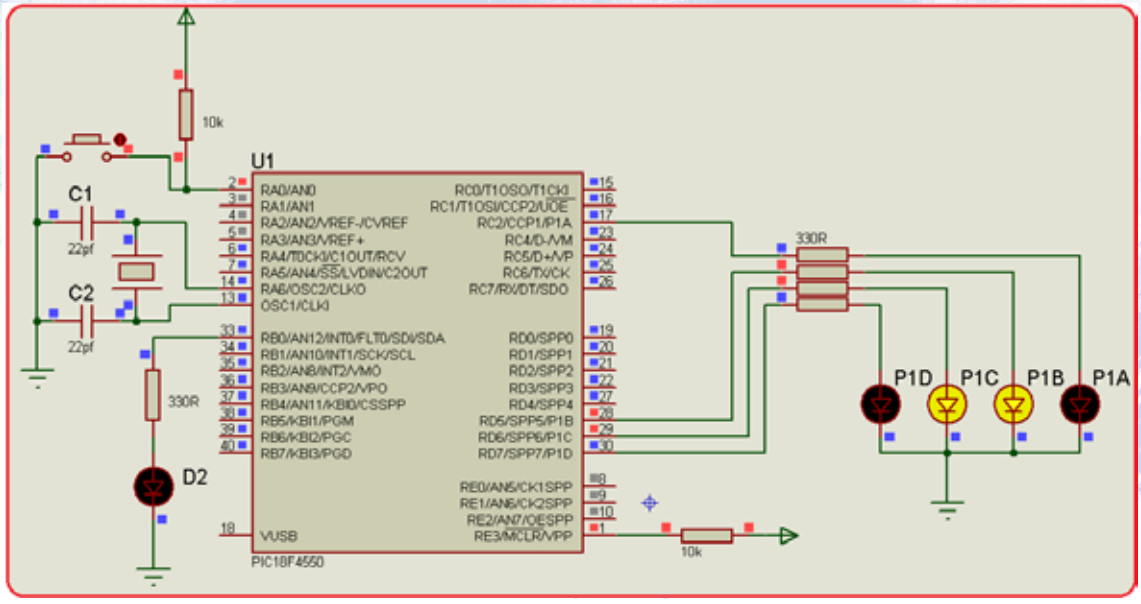


Figura 2.27. Operación FULL-BRIDGE, activo en alto modo inverso. P1C activada y P1B señal modulada.
Elaborado por: Los Autores.

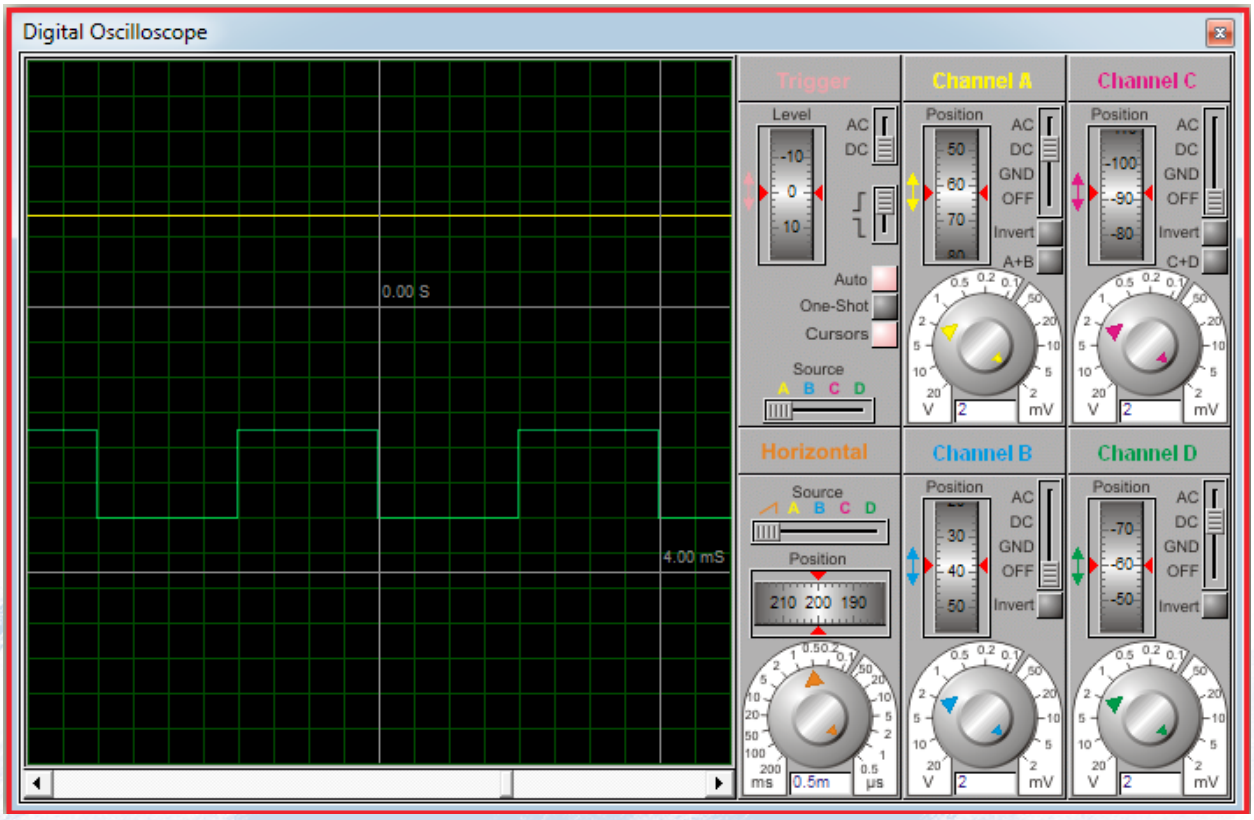


Figura 2.28. Señales en osciloscopio virtual. FULL-BRIDGE modo directo, activo en alto.
Elaborado por: Los Autores.

Transición en el cambio de sentido

Cuando se realiza un cambio de sentido (cambio en P1M1), el ECCP establece el nuevo sentido en el siguiente ciclo PWM. Justo antes de finalizar el último ciclo previo al cambio de sentido, se desactivan las señales moduladas (P1B o P1D) y las salidas no moduladas (P1A o P1C) se ponen en la configuración correspondiente al nuevo sentido. Esto ocurre un intervalo de tiempo antes del inicio del siguiente ciclo PWM:

Intervalo de tiempo de transición=4*TOSC*(Prescaler TMR2)

Con este tiempo de transición se evitan posibles cortocircuitos en las semi-ramas del puente durante la transición.

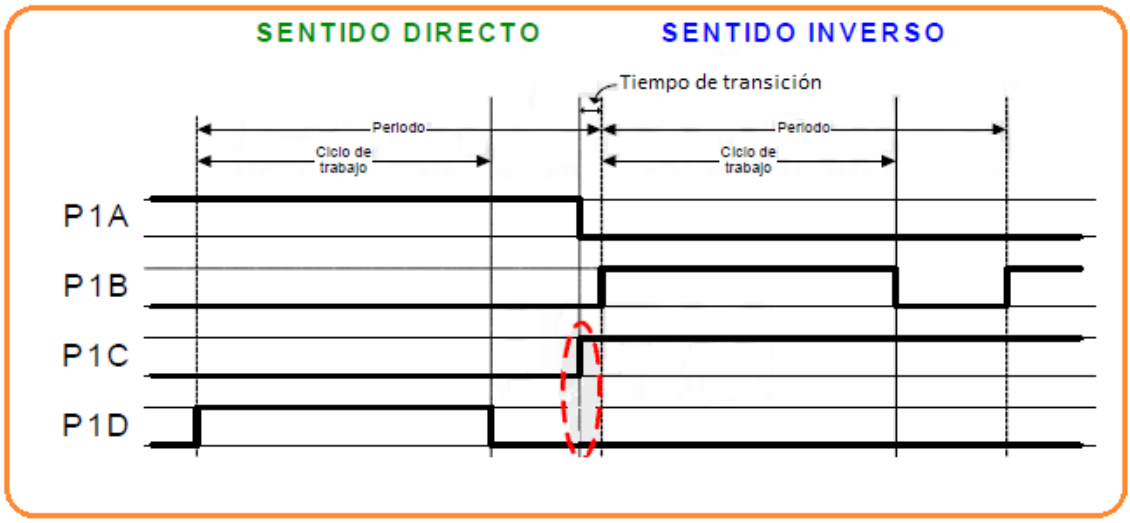


Figura 2.29. Tiempo de transición en modo FULL-BRIDGE.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

En determinadas condiciones de funcionamiento puede producir un cortocircuito en algunas de las semiramas del puente completo, a pesar del intervalo de transición. El cambio de sentido se produce cuando el ciclo de trabajo del PWM está cerca del 100% y los transistores utilizados en el semipuente tienen un tiempo de conmutación a OFF superior al tiempo de conmutación a ON se producirá un cortocircuito en las semiramas del puente.

Existen diferentes soluciones para evitar este problema. Algunas de ellas podrían ser:

- Reducir el ciclo de trabajo ligeramente antes de realizar el cambio de sentido.
- Utilizar transistores o circuitos de disparo de los transistores que permitan que la conmutación a OFF sea más rápida que la conmutación a ON.
- Programar el módulo EECP para que desactive las salidas P1A, P1B, P1C, P1D.

Ejercicio 2.10. Control de circuito FULL-BRIDGE con PWM del módulo ECCP.

Realizar un programa para controlar a un circuito puente completo con el uC PIC18F4550 que genere las señales de disparo de la etapa de potencia de un inversor con carga resistiva utilizando el PWM mejorado del ECCP en modo FULL-BRIDGE. El valor eficaz de la tensión en la carga se establece mediante una constante value con un ciclo de trabajo del 75%, F_{OSC} = 4MHz,

un intervalo de conmutación= 10us, asumiendo que se aplique en forma simétrica a los inversos los pulsos a 60Hz.

El periodo del inversor es:

Tinversor= 1/60Hz

Tinversor=16.66 ms.

El periodo de los pulsos que deben generarse en P1B y P1D será:

T_{PWM}= 16.66ms/10

T_{PWM}= 1666us, considerando que generaremos 10 pulsos para cada ciclo.

Los pulsos se generarán con el módulo ECCP trabajando en el modo PWM mejorado en puente completo con las líneas P1A, P1B, P1C y P1D activas a nivel alto. En el semiperiodo positivo del inversor el módulo ECCP trabajará en sentido directo, mientras que en el semiperiodo negativo el ECCP trabajará en sentido inverso. El cambio de sentido debe realizarse cada 5 pulsos. Para ello se configura el postescalar del Temporizador 2 en 1:5. De esta forma cada vez que hayan generado 5 pulsos se producirá una interrupción del Temporizador 2. Dicha interrupción se utilizará para cambiar el sentido (directo o inverso), mediante una variable tipo bit que se invierte cuando existe una interrupción del temporizador 2. La figura 2.30, ilustra las señales que deseamos obtener en el inversor.

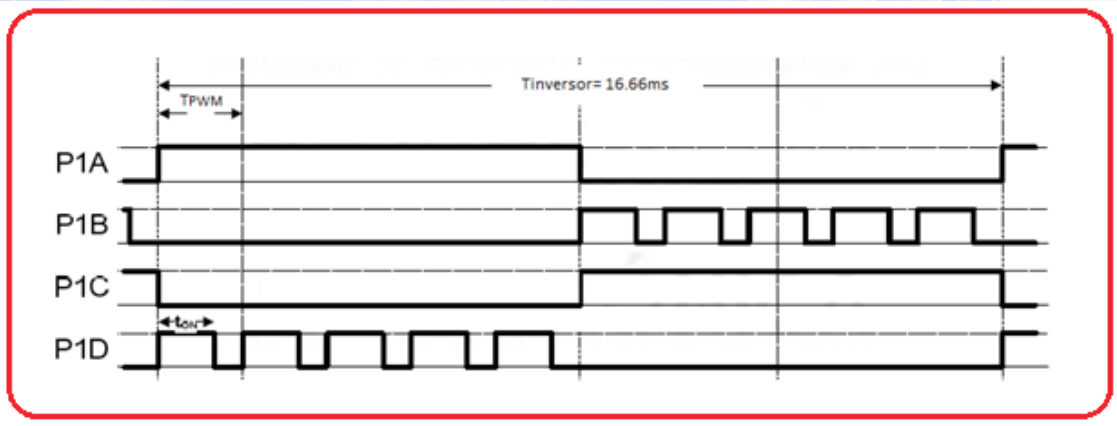


Figura 2.30. Señales del inversor con FULL-BRIDGE.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Utilizando la ecuación para calcular el periodo del PWM del CCP1, el valor del PR2 será igual a:

PR2= T_{PWM} / (4*T_{OSC}*T2_DIV), el pre-escalar del Timer2 tomaremos el máximo 1:16

PR2= 1666 us / (4*0.25 us*16)

PR2= 104.125, utilizaremos la parte entera 104.

PR2= 104.

Considerando que el ciclo de trabajo es DC= 75%, calculamos la variable **value** para la función del programa:

Value= DC / [(4*(PR2+1)]= 0.75 / (4*104)

Value= 312

Con estos datos escribimos el programa:

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADE          //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#BYTE port_a= 0xF80                     //Identificador del Puerto A.
                                        //Usa la directiva para manejo i/o rápidas de los puertos
#use fast_io (B)                        //A, B, C, D.
#use fast_io (C)
#use fast_io (D)
long value= 312;                        //Variable para el ciclo de trabajo, equivale al 75%.
int1 s=0;                               //Variable para el cambio de modo, directo o inverso.
#INT_Timer2                             //Interrupción del TMR2.
void DesbordeTimer2()                   //Función de la interrupción.
{
 s= ~s;                                 //Invierte la variable s.
 output_high (PIN_B0);                  //Activa LED de señalización.
}
void main()                             //Función principal main.
{
SET_TRIS_B(0X00);                       //Puerto B como salida.
SET_TRIS_C(0X00);                       //Puerto C como salida.
SET_TRIS_D(0X00);                       //Puerto D como salida.
setup_timer_2(T2_DIV_BY_16,104, 5);     //Configura el TMR2.
enable_interrupts ( INT_Timer2 );       //Habilitación de la interrupción del TMR2.
enable_interrupts ( GLOBAL);            //Habilitación Global de las interrupciones.
while (TRUE){                           //Inicio del bucle infinito.
   if (s ==0){
        //Configura el Puente FULL-BRIDGE activo en alto modo directo.
    setup_ccp1(CCP_PWM_FULL_BRIDGE | CCP_PWM_H_H);
   }
   else{
        //Configura el Puente FULL-BRIDGE activo en alto modo indirecto.
    setup_ccp1(CCP_PWM_FULL_BRIDGE_REV | CCP_PWM_H_H);
    output_low (PIN_B0);                //Apaga LED.
  }
 set_pwm1_duty(value);                  //75% duty cycle
 }                                      //Fin del bucle.
}                                       //Fin del main.
```

La figura 2.31, indica el inversor completo con transistores MOSFET controlados con PIC 18F4550.

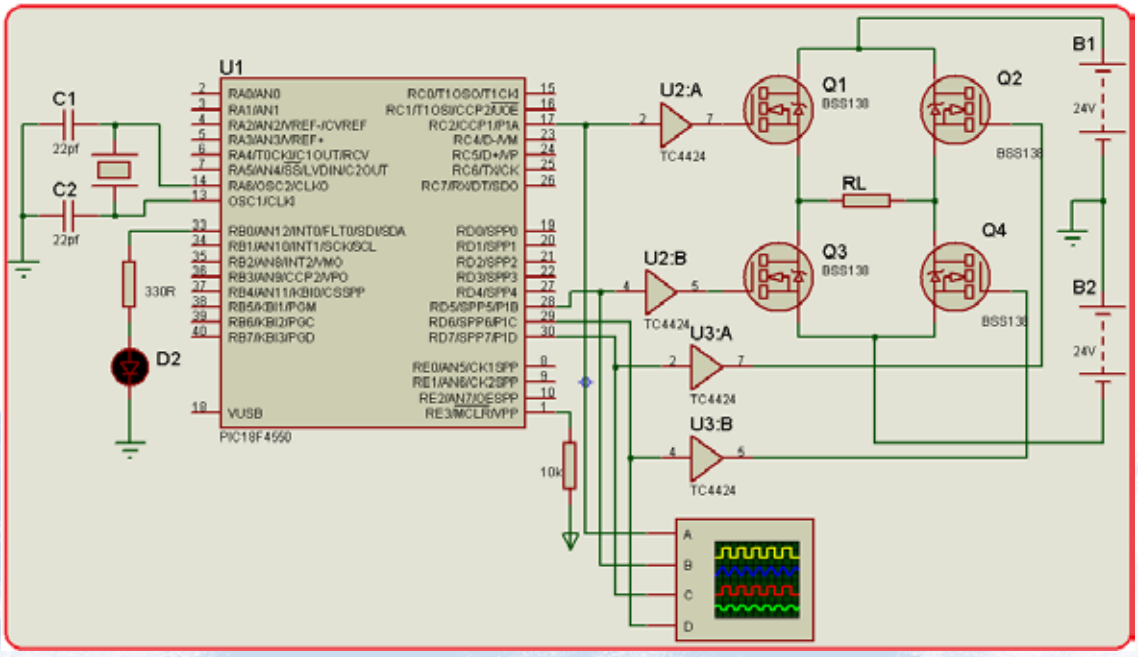


Figura 2.31. Modo FULL-BRIDGE para control de puente completo.
El resultado de las señales obtenidas se indica en la figura 2.32.

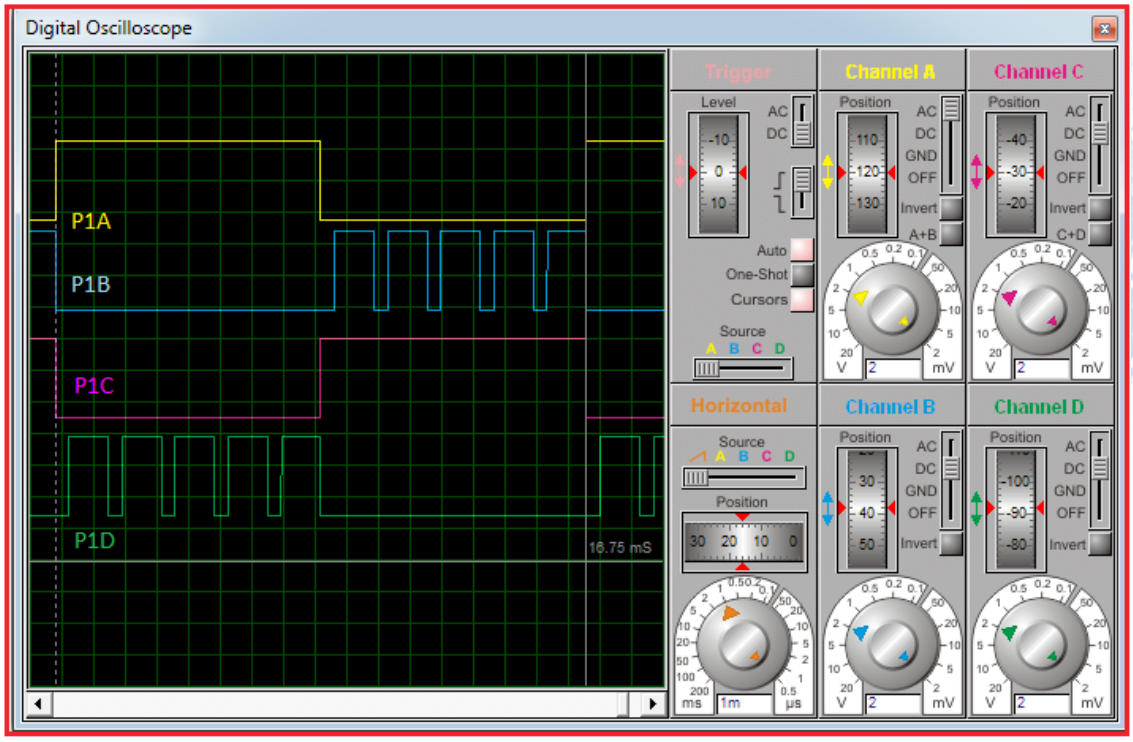


Figura 2.32. Operación del circuito inversor MOSFET.
Elaborado por: Los Autores.

EJERCICIOS ADICIONALES

Ejercicios resueltos

Medir dos valores analógicos y mostrarlos en el LCD.

En el siguiente ejercicio se va a medir directamente dos valores de señales análogas y mostrarlas en el LCD las medidas respectivas.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#device adc=10                          //Define el adc para 10 bits.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN         //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC =4MHz.
#include <lcd.c>                        //Incluye el LCD.
#include <stdlib.h>                     //Librería stdlib.h
#byte port_b = 0xF81                    //Identifica el Puerto b.
void main()                             //Función principal main.
{
  float valor0, valor_conversion0;      //Variables para lectura y conversión del ADC, canal 0.
  float valor1, valor_conversion1;      //Variables para lectura y conversión del ADC, canal 1.
  set_tris_b(0x00);                     //Puerto B como salida.
  puerto_b = 0x00;                      //Puerto B en 0
 setup_adc_ports(AN0);                  //Usa la entrada analógica AN0.
 setup_adc(ADC_CLOCK_DIV_2);            //Reloj del ADC divide para 2.

  lcd_init();                           //Inicializa el LCD.

  while (TRUE) {                        //Bucle infinito
    //Titila el LED para indicar que el circuito está funcionando.
    output_low(PIN_b0);                 //LED off
    delay_ms(200);
    output_high(PIN_b0);                //LED on
    delay_ms(200);

    set_adc_channel(0);                 //Usa el  canal 0 del ADC.
    valor0= read_adc();                 //Asigna el valor del ADC a la variable "valor0".
    valor_conversion0= valor0*5/1023;           //Convierte el valor de la  entrada 0 de  0...5.
    lcd_gotoxy(1,1);                    //Posiciona el cursor en 1,1.
    printf(lcd_putc, "ADC canal 0= %6.3f",valor0);          //Indica el valor del ADC canal 0.
    lcd_gotoxy(1,2);                    //Posiciona el cursor en 1,2.
    printf(lcd_putc,"Voltaje=%6.3f",valor_conversion0);             //Indica la medida equivalente ANO.
    delay_ms(500);                      //Muestra en el LCD el primer valor durante 500ms.
    set_adc_channel(1)                  //Usa el  canal 1 del ADC.
    valor1= read_adc();                 //Asigna el valor del ADC a la variable "valor1".
    valor_conversion1= valor1*5/1023;           //Convierte el valor de la  entrada 1 de  0...5.
    lcd_gotoxy(1,1);                    //Posiciona el cursor en 1,1.
    printf(lcd_putc,"ADC canal 1= %6.3f",valor1);           //Indica la medida equivalente AN1.
    lcd_gotoxy(1,2);                    //Posiciona el cursor en 1,2.
    printf(lcd_putc,"Voltaje=%6.3f",valor_conversion1);             //Indica la medida equivalente AN1.
    delay_ms(500);                      //Muestra en el LCD el segundo valor durante 500ms.
  }                                     //Fin del bucle infinito.
}                                       //Fin del main.
```

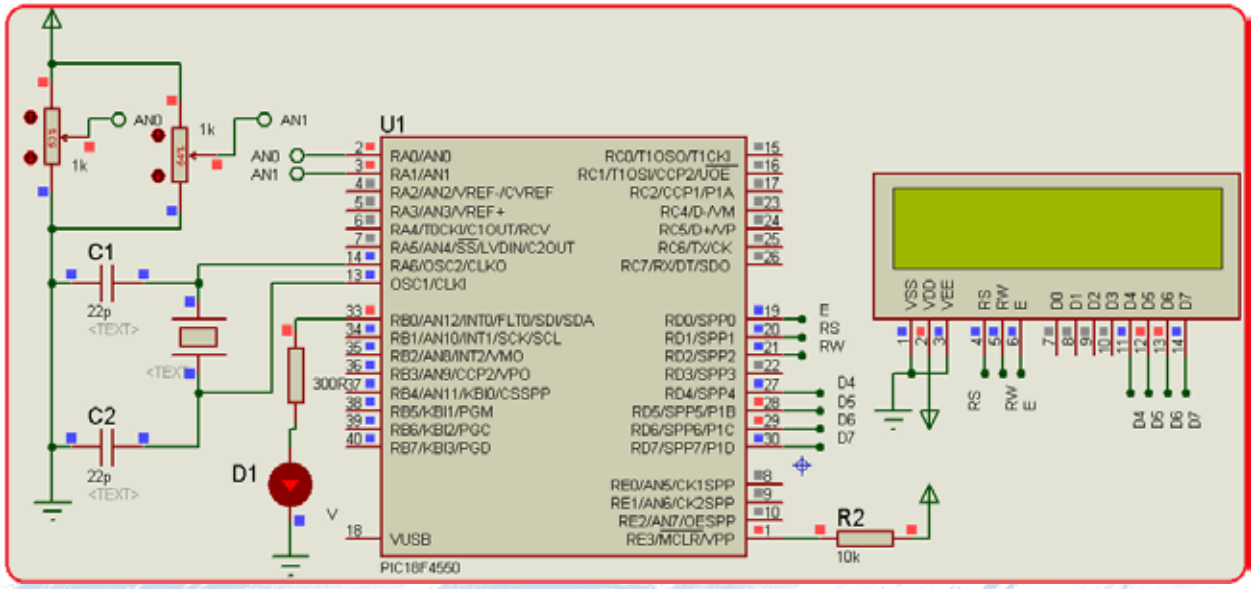


Figura 2.33. Aplicación del ADC, midiendo 2 señales analógicas simultáneamente.

Los resultados se muestran en la figura 2.34.

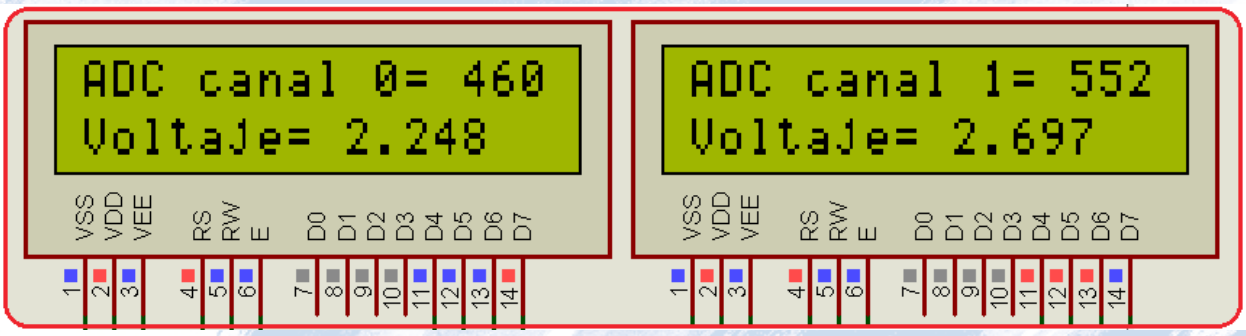


Figura 2.34. Captura de los resultados indicados en el LCD.
Elaborado por: Los Autores.

Se requiere en el ADC que un grupo de LEDS conectados al puerto B se activen de acuerdo a la variación de la entrada. Cuando la entrada llegue a 1, se prende el LED1, a 2 se activa el LED2, y así hasta el 5.

El ejercicio propone realizar un vúmetro progresivo de LEDS, similar a un vúmetro de sonido, un grupo de LEDS se activan según el nivel de audio. En este caso aumenta el voltaje de entrada los LEDS se prenden desde un valor mínimo a un máximo.

De acuerdo al ejercicio planteado debemos tomar en cuenta si la entrada es 1 el LED conectado RB0 se activa, si es la entrada es 2 el LED conectado en RB1 se prende y así sucesivamente hasta el puerto RB4. La tabla 2.18, resume el funcionamiento de los LEDS y el valor que adquiere el puerto.

ESTADO DEL PUERTO B	LED4	LED3	LED2	LED1	Entrada(i)
0	0	0	0	0	0
1	0	0	0	1	1
3	0	0	1	1	2
7	0	1	1	1	3
15	1	1	1	1	4

Tabla 2.18. Estado del puerto B.
Elaborado por: Los Autores.

Por tanto, la ecuación que describe el funcionamiento del estado del puerto es:

Puerto = $2^i - 1$.

Para realizar la operación de potenciación se utiliza la función **pow(x,y)**, que viene incluida en la librería **math.h.**

Ahora ¿cómo obtenemos el número entero **i**? La conversión matemática del ADC es un número decimal (float), como necesitamos solo la parte entera, la operación de las variables se hace en tipo enteros (long int). Es decir, si se efectúa la operación i= n*5/1023; siendo i y n tipo long int, entonces el resultado **i** es la parte entera de la división.

PROGRAMA:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550
#device adc=10                                //Define el adc para 10 bits.
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN       //Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC =4MHz.
#include <lcd.c>                              //Incluye el LCD.
#include <stdlib.h>                           //Librería stdlib.h
#include <math.h>                             //Include la librería math.h
#byte port_b = 0xF81                          //Identifica el Puerto b.
void main()                                   //Función principal main.
{
   float valor, valor_conversion;     //Variables para la conversión.
   long int i, n;                             //Variables para el vúmetro.
   set_tris_b(0x00);                          //Define el puerto B  como salida.
   port_b = 0x00;                             //Puerto B reseteado.
   setup_adc_ports(AN0);                      //Usar entrada analógica AN0.
   setup_adc(ADC_CLOCK_INTERNAL);     //Usar reloj interno para el ADC 32KHZ.
   set_adc_channel(0);                        //Usar canal 0 del ADC.
   lcd_init();                                //Inicializar LCD.

   while (TRUE)                               //Bucle infinito.
   {
         //El LED se prende y se apaga para indicar que el conversor está funcionando.
     output_low(PIN_b0);                      //LED off.
     delay_ms(200);                           //Retardo de 20 ms.
     output_high(PIN_b0);                     //LED on.
     delay_ms(200);                           //Retardo de 200 ms.
     valor= read_adc();                       //Lee el adc y asigna a valor.
     valor_conversion= valor*5/1023;                  //Conversión matemática.
     lcd_gotoxy(1,1);                         //Ubica el cursor en la posición 1,1 del LCD.
     lcd_putc ("   ADC - PIC");               //Mensaje.
     lcd_gotoxy(1,2);                         //Ubica el cursor en la posición 1,2 del LCD.
     printf(lcd_putc,"Voltaje=%6.3f",valor_conversion);        //Muestra el valor equivalente en voltios.
    //Vúmetro..
     n= valor;                                //Conversión de float a int.
     i=  n*5/1023;                            //Obtiene el entero de la conversión.
     port_b= pow(2,i)-1;              //Eleva a la potencia i la base 2, resta 1 y asigna ese valor al puerto B.
   }                                          //Fin del bucle infinito.

         }                                          //Fin del main.
```

La figura 2.35, ilustra las conexiones del circuito usado. La activación de los LEDS es proporcional con la entrada analógica.

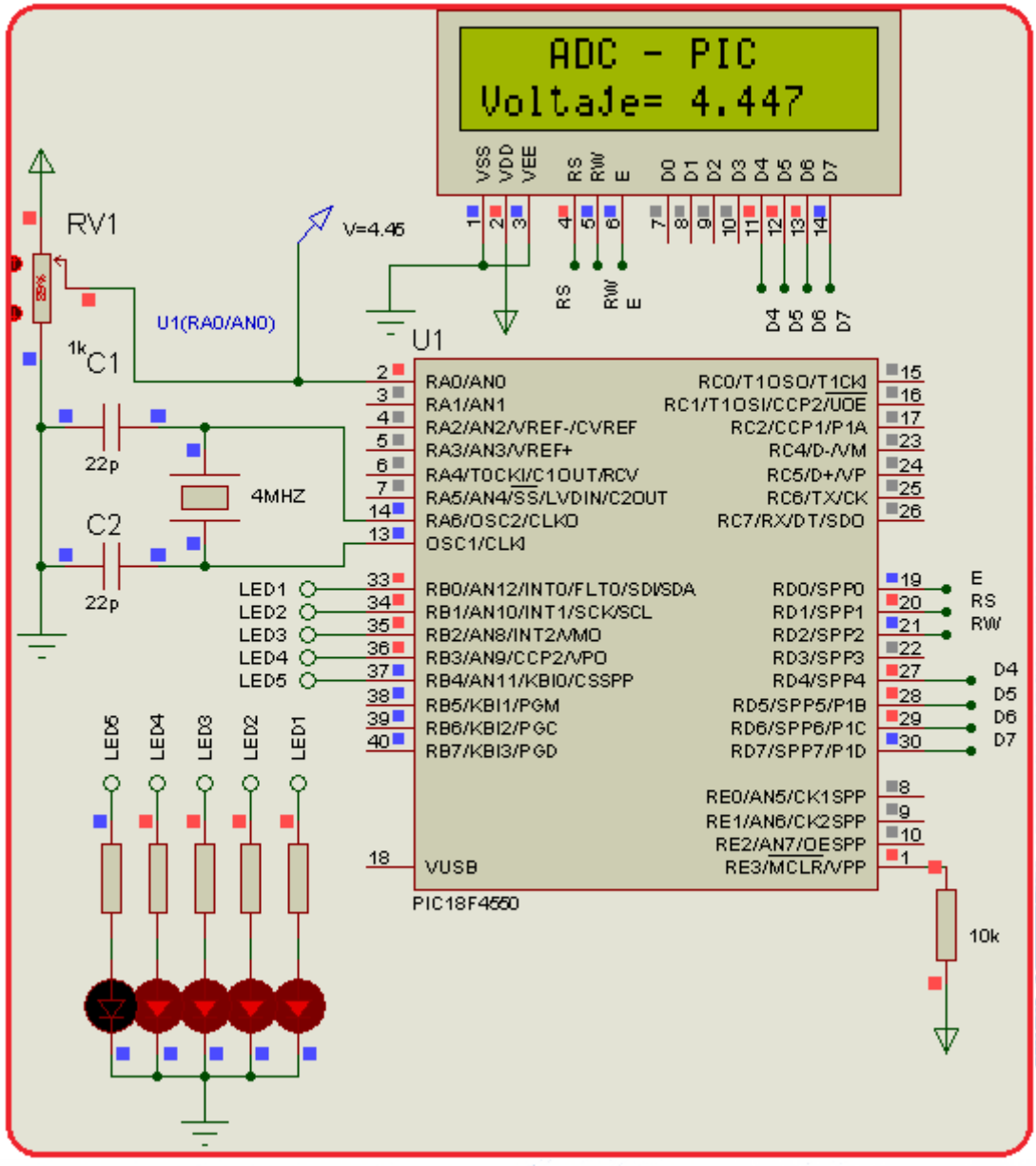


Figura 2.35. ADC con vúmetro digital de LEDS.
Elaborado por: Los Autores.

Realizar un programa para generar frecuencias de 5, 10 y 20kHz. Usar un DC de 50%, 25% y 10% respectivamente. Seleccionar la frecuencia mediante pulsadores y utilizar el pre-escalar adecuado.

Para F_{PWM}= 5 kHz y DC= 50%.
TPWM= 1/5kHz= 0.2 ms= 200us.
TOSC= 1/CLOCK = 1/12MHz= 0.08333 us.
PR2= (T_{PWM}/T_{OSC}*4*t2div) – 1

Tomamos un t2div= 4. (El lector puede probar con 1 y 16, y observará que esta fuera de límites el PR2 y el otro caso da un número decimal).

PR2= (200 us / 0.08333 us *4*4) -1
PR2= 149.
El duty cycle:
DC= value/4*(PR2+1)
value= DC*4*(PR2+1)
value= 0,5*4(149+1)
value= 300

Aplicando las mismas ecuaciones se obtienen los valores del PR2 y value para 10kHz y 20kHz.

El lector puede comprobar:

- Para F_{PWM}= 10 kHz, DC= 25% y t2div= 4
  PR2= 74
  value= 75
- Para F_{PWM}= 20 kHz y DC= 10% y t2div= 1
  PR2= 149
  value= 60

Es necesario asignar los valores de duty cycle a una variable long int, y no asignarla a la instrucción directamente, porque en este caso el valor tomaría como una variable int dando resultados erróneos en la frecuencia generada.

Por ejemplo:

```
value= 60;
set_pwm1_duty(value);
```

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550,
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <stdlib.h>                     //Librería stdlib.h.
#BYTE port_b= 0xF81                     //Identificador para el puerto b en  la localidad 0xF81.
                                        //Definición de los pulsadores en los puertos b.0, b.1 y b.2.
#define s1 bit_test(port_b,0)
#define s2 bit_test(port_b,1)
#define s3 bit_test(port_b,2)

void main(void)                         //Función principal main.
{
     long int  value;                   //Variable para el DC.
int seleccion = 0;                      //Variable para seleccionar el PWM.
set_tris_b(0xFF);                       //Configura el puerto B como entrada.
     setup_ccp1(CCP_PWM);               //Configura CCP1 como PWM.
 while( TRUE ){                         //Bucle infinito.
  if (s1==0){                           //Si s1=0, selecciona PWM1.
    seleccion =1;
  }
  if (s2==0){                           //Si s2=0, selecciona PWM2.
    seleccion =2;
  }
  if (s3==0){                           //Si s3=0, selecciona PWM3.
    seleccion =3;
  }
  switch(seleccion) {
    case 1 : setup_timer_2(T2_DIV_BY_4, 149, 1);
             value= 300;
             set_pwm1_duty(value);
            break;
    case 2 : setup_timer_2(T2_DIV_BY_4, 74, 1);
             value= 75;
             set_pwm1_duty(value);
            break;
    case 3 : setup_timer_2(T2_DIV_BY_1, 149, 1);
             value= 60;
             set_pwm1_duty(value);
           break;
  }
   set_pwm1_duty(value);
 }                                      //Fin del while.
}                                       //Fin del main.
```

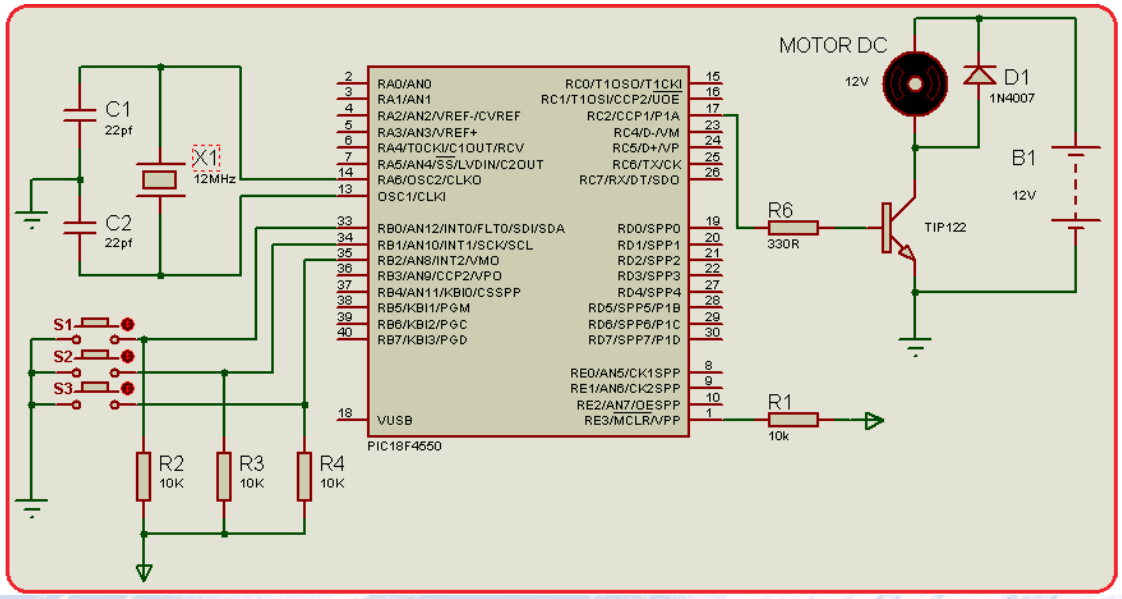
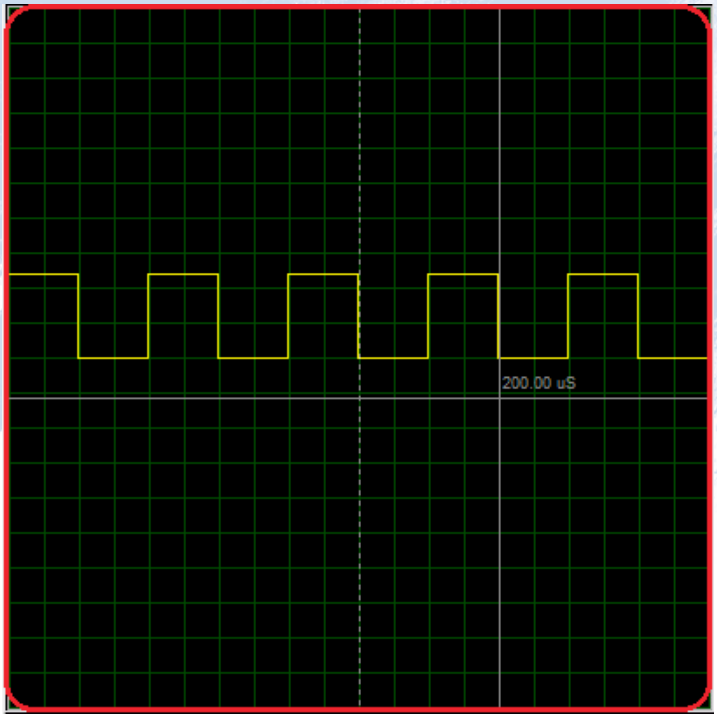
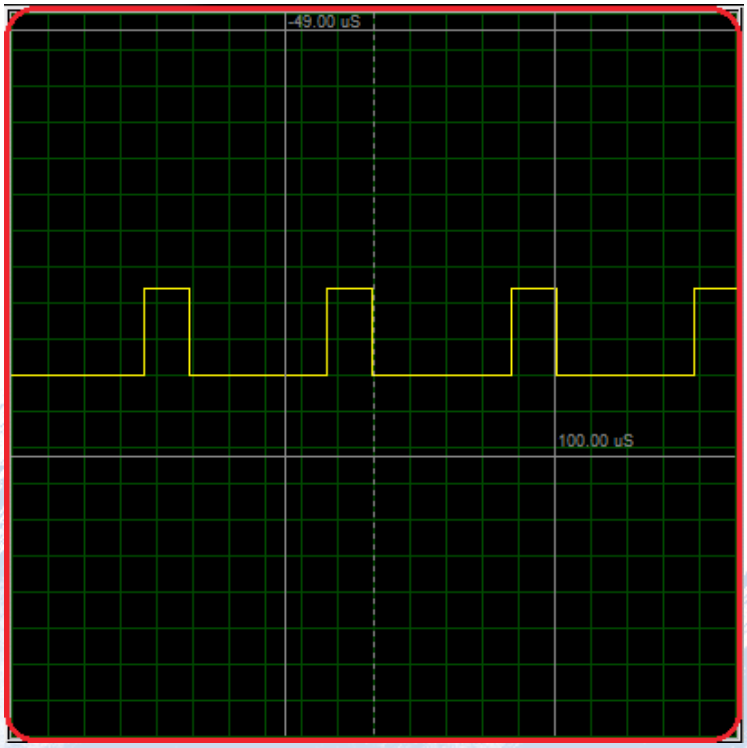


Figura 2.36. Generación de 3 PWM diferentes.
Elaborado por: Los Autores.

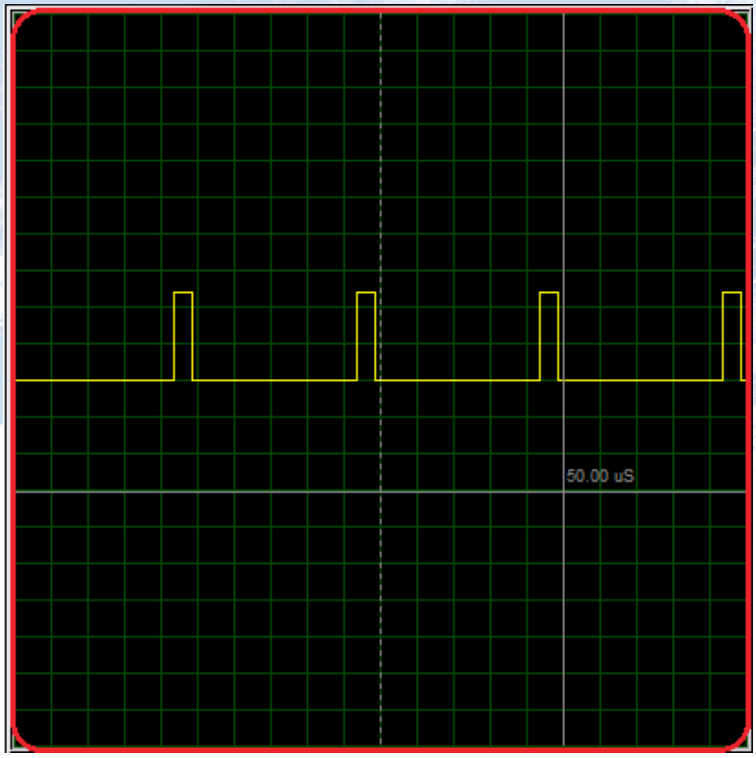
Los resultados obtenidos se indican en la figura 2.37, a) 5kHz, b) 10kHz y c) 20kHz.



a) Periodo 200 us.



b) Periodo 100 us.

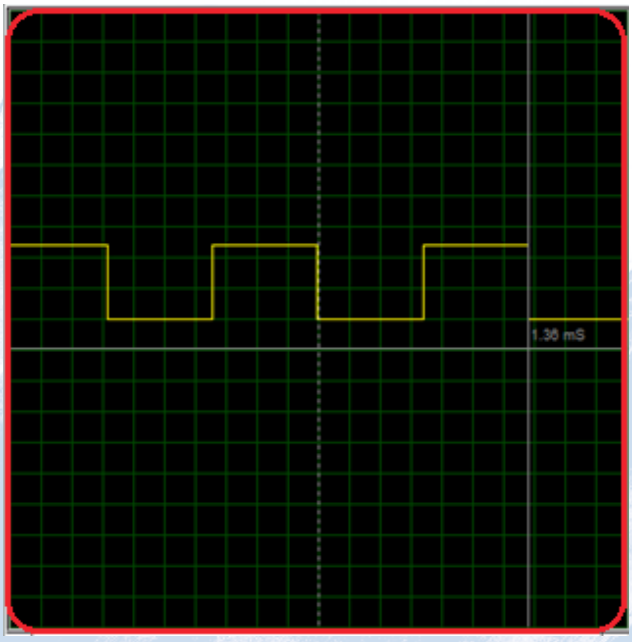


c) Periodo 50 us.

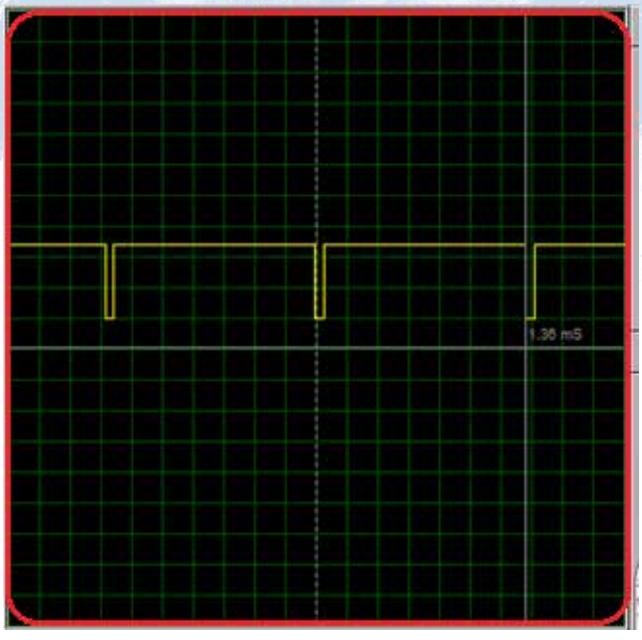
Figura 2.37. Resultado de las frecuencias generadas: 5kHz, 10kHz y 20kHz.
Elaborado por: Los Autores.

PWM variable

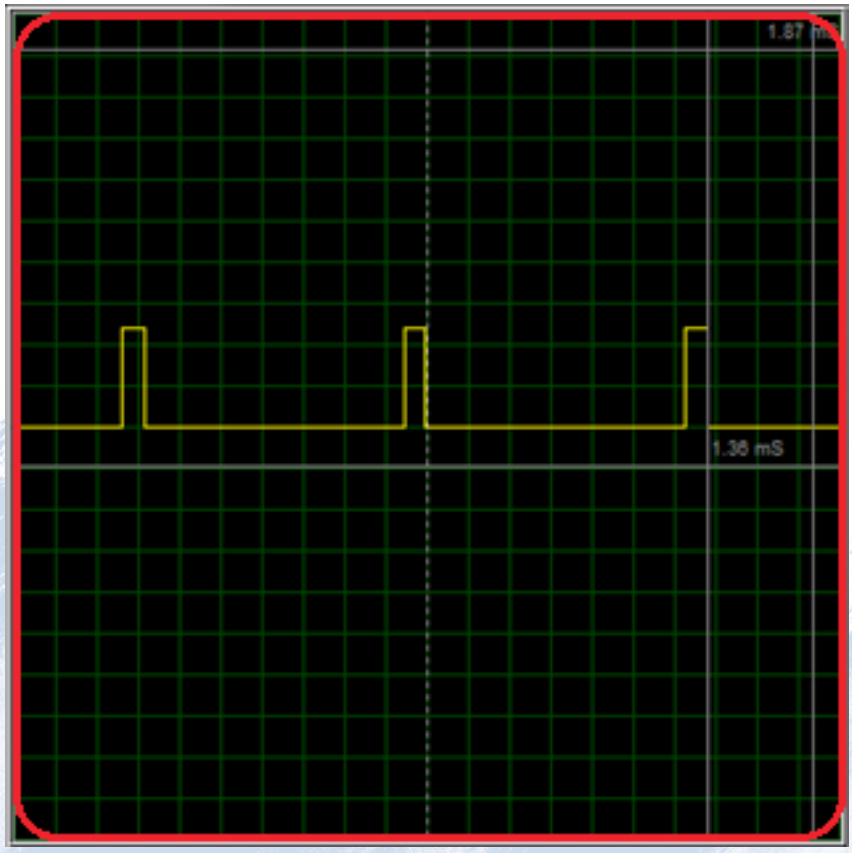
La figura 2.38, muestra la captura de imágenes obtenidas con el osciloscopio virtual. En a), la señal cuando al accionar el pulsador inicio, en b) cuando la señal se ha incrementado a un máximo valor del ancho del pulso y en c) el ancho del pulso del PWM es mínimo. Para los tres casos se mantiene el mismo valor del periodo de la señal PWM.



a) PWM inicial.



b) PWM con máximo ancho de pulso.



c) PWM con mínimo ancho de pulso.
Figura 2.38. Generación de PWM variable.
Elaborado por: Los Autores.

Realice el diseño de cruce por cero para disparar a un SRC, para un ángulo de disparo de 45º.

La frecuencia de la red de alimentación es 60 Hz, en consecuencia el periodo de la señal es:

T= 1/60Hz= 16.67ms, que corresponde a 360º.

A 45º, el tiempo transcurrido del periodo de la onda es:

T_{45^o}= 16,67ms*45/360= 2.083ms= 2083 us.

La puerta del SCR dependiendo de las características del fabricante necesita un pulso de corriente mayor a 1 us. Para el caso en particular que estamos tratando vamos a tomar 10 us.

Principio de funcionamiento

El cruce por cero es detectado por RB0, generándose la interrupción externa, a partir de este momento, se activa el Timer0, para generar un retardo de 2083 us. Al producirse la interrupción del TMR0, se activa el pin RD0 y el Timer1 el cual genera el tiempo del pulso (10 us) para aplicar a la compuerta a través de RD0. Cuando ocurre la interrupción del TMR1, el pin RD0 se pone en 0 y se desactiva el TMR1.

Requerimientos de programación

1. Interrupción RB0 disparado en flanco de bajo a alto.
2. TMR0 trabajando en modo de temporizador. El valor de carga del TMR0, podemos calcular utilizando la ecuación general del tiempo de temporización, o sea:

   TMR0= 65 536 – Tiempo /(4*T_{OSC}*Pre-escalar).
   Considerando F_{OSC} = 4 MHz y un Pre-escalar de 1.
   TMR0= 65 536 – 2083 us/(4*0.25 us*1)
   TMR0= 63453.

3. TMR1 trabajando en modo de temporizador. En este caso el valor de carga del TMR1 con un pre-escalar de 1, es:

   TMR0= 65 536 – 10 us/(4*0.25 us*1)
   TMR0= 65 526.

PROGRAMA:

```
#include <18F4550.h>                    //Incluye PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP   //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#byte PORT_D=0xF83                      //Identificador del Puerto D.
#byte PORT_B=0xF81                      //Identificador del Puerto B.

#int_ext                                //Interrupción externa.
void PortB0_Interrupt(void)             //Función interrupción externa.
{
 ext_int_edge(L_TO_H);                  //Habilita la interrupción externa  en flanco de subida.
 ENABLE_INTERRUPTS( INT_Timer0 );       //Habilita el TMR0,
 SET_TIMER0(63453);                     //Carga el TMR0.
 SETUP_TIMER_0(RTCC_DIV_1|RTCC_INTERNAL);       //Configura el TMR0.
}

#INT_Timer0                             //Interrupción TMR0.
void DesbordeTimer0()                   //Función del TMR0.
{
   ENABLE_INTERRUPTS( INT_Timer1 );             //Habilita la interrupción del TMR1.
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);      //Configura el TMR1.
   SET_TIMER1(65526);                           //Carga el TMR1.
   DISABLE_INTERRUPTS( INT_Timer0);             //Deshabilita el TMR0.
   output_bit(pin_D0,1);                        //Activa a 1 al pin RD0.
}

#int_Timer1                                     //Interrupción TMR1.
void DesbordeTimer1()                           //Función de la interrupción.
{
  output_bit(pin_d0,0);                         //Desactiva pin RD0.
  DISABLE_INTERRUPTS( INT_Timer1);              //Deshabilita TMR1.
}

void main(void)                                 //Función principal main.
{
  set_tris_b(0b00000001);                       //Fija el puerto RB1 como entrada.
  set_tris_d(0b00000000);                       //Fija el Puerto D como salida.
  PORT_D=0;                                     //Puerto D en 0.
  enable_interrupts(GLOBAL);                    //Habilitación de las interrupciones.
  enable_interrupts(INT_EXT);                   //Habilitación de la variable externa.

 while(TRUE){                                   //Bucle infinito.
  //Otras instrucciones.
  }                                             //Fin del bucle.
}                                               //Fin del main.
```

La figura 2.39, muestra el circuito de control para disparo del SCR.

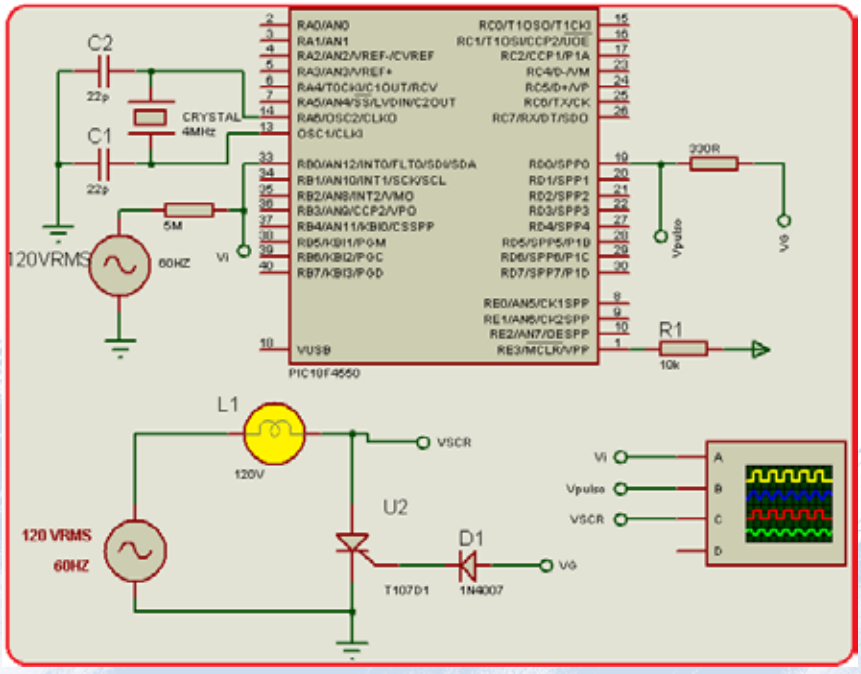
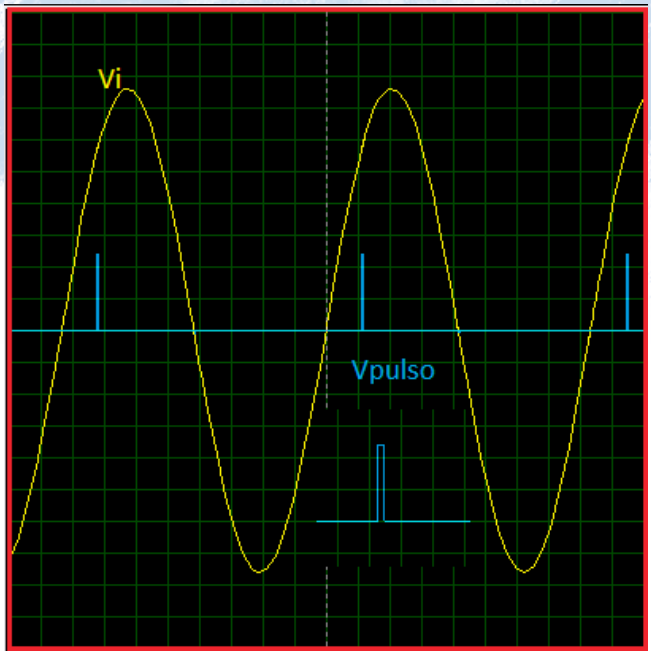
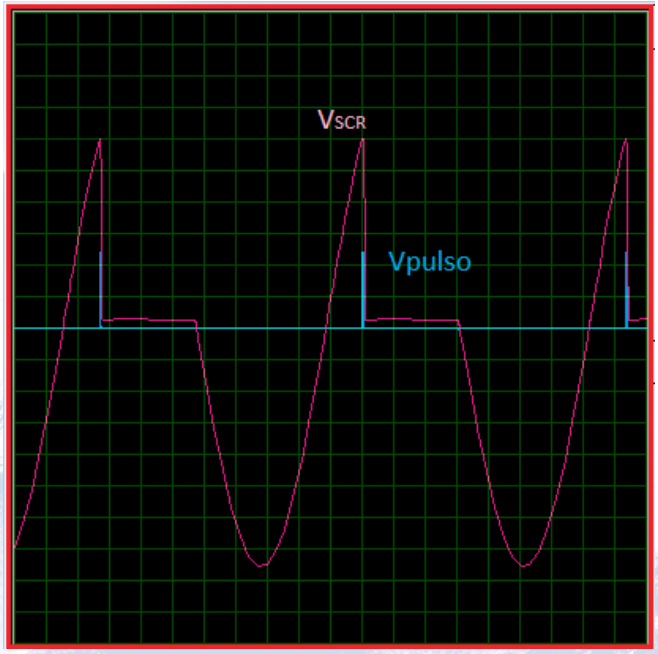


Figura 2.39. Circuito de control para disparo de SCR.

La figura 2.40, describe las curvas resultantes del funcionamiento. En a) se observa la señal de entrada y el pulso generado. En b) La señal del SCR y el pulso correspondiente en el momento que el SCR conduce.



a). Señal de entrada y pulso para la compuerta.



b) Voltaje en el SCR y pulso de la compuerta.

Figura 2.40. Formas de onda obtenidas en el osciloscopio virtual del circuito de disparo para SCR a 15º.
Elaborado por: Los Autores.

Medición del ancho de pulso utilizando la interrupción externa por RB0.

Otra forma de medir el ancho de un pulso es mediante la interrupción externa RB0. El principio de desarrollo del programa se basa en:

- Ingresar el pulso o señal por RB0.
- Habilitar el flanco de subida para que detecte la interrupción por externa.
- Comenzar a contar el tiempo que transcurre en TMR1.
- Habilitar el flanco de bajada para que detecte la interrupción externa.
- Asignar el valor del TMR1 a una variable apropiada.
- Acondicionar el resultado para mostrar en el display.

PROGRAMA:

```
#include <18F4550.h>                //Librería para usar el PIC18F4550.
#fuses XT, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD      //Configuración de fusibles.
#use delay (clock=4000000)          //F_OSC = 4 MHz.
#include <lcd.c>                    //Inicializa el LCD.

int16 TFS;                          //Tiempo flanco de subida
float Ap;                           //Valor del ancho del pulso
int1 nuevopulso = 0;                //Ingresa un nuevo pulso
int1 cambio =0;                     //Control de cambio de flanco

#int_ext
void funcion_ext_int()              //Función para la interrupción
{
  if (cambio ==0)                   //Control de cambio de flanco (subida)
    {
      set_timer1(0);                //Inicia TMR1
      ext_int_edge(H_TO_L);         //Configura para flanco de bajada.
      cambio = 1;                   //Control de cambio de flanco (bajada).
    }
   else
    {
     TFS = get_timer1();            //Carga el valor de TMR1 a TFS.
     ext_int_edge(L_TO_H);          //Configura para flanco de subida.
     cambio = 0;                    //Control de cambio de flanco (subida).
     if (nuevopulso ==0)            //Control de cambio de flanco (bajada).
      {
      nuevopulso = 1;               //Control de cambio de flanco (subida)
      }
    }
}

void main(void)                     //Función principal main.
{
  lcd_init();                       //Inicializa el LCD,
  setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);   //Configura el TRM1.
  ext_int_edge(H_TO_L);                       //Flanco de bajada.
  cambio = 0;                                 //Cambio de flanco (bajada).
  enable_interrupts(int_ext);                 //Habilita interrupción externa.
  enable_interrupts(global);                  //Habilita interrupciones,

  while(TRUE){                                //Bucle infinito.
    if (nuevopulso == 1){                     //Si hay nuevo pulso….
       Ap= TFS;                               //… asigne el TFS a Ap.
      printf(lcd_putc,"\nPulso= %6.1f us", Ap);       //Muestre en LCD el valor.
    }
    nuevopulso = 0;                           //Reinicie variable.
  }                                           //Fin del bucle infinito.
}                                             //Fin del main.
```

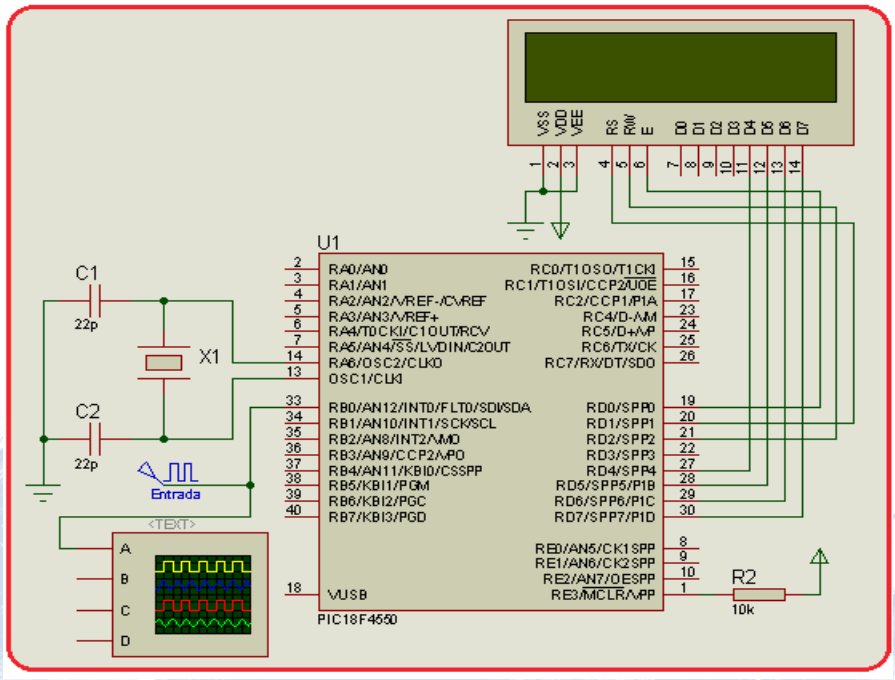


Figura 2.41. Diagrama para medir el ancho de pulso por RB0.
Elaborado por: Los Autores.

Se ingresa un pulso de 500 us. En la figura 2.42, se muestra la medida indicada en el osciloscopio virtual.

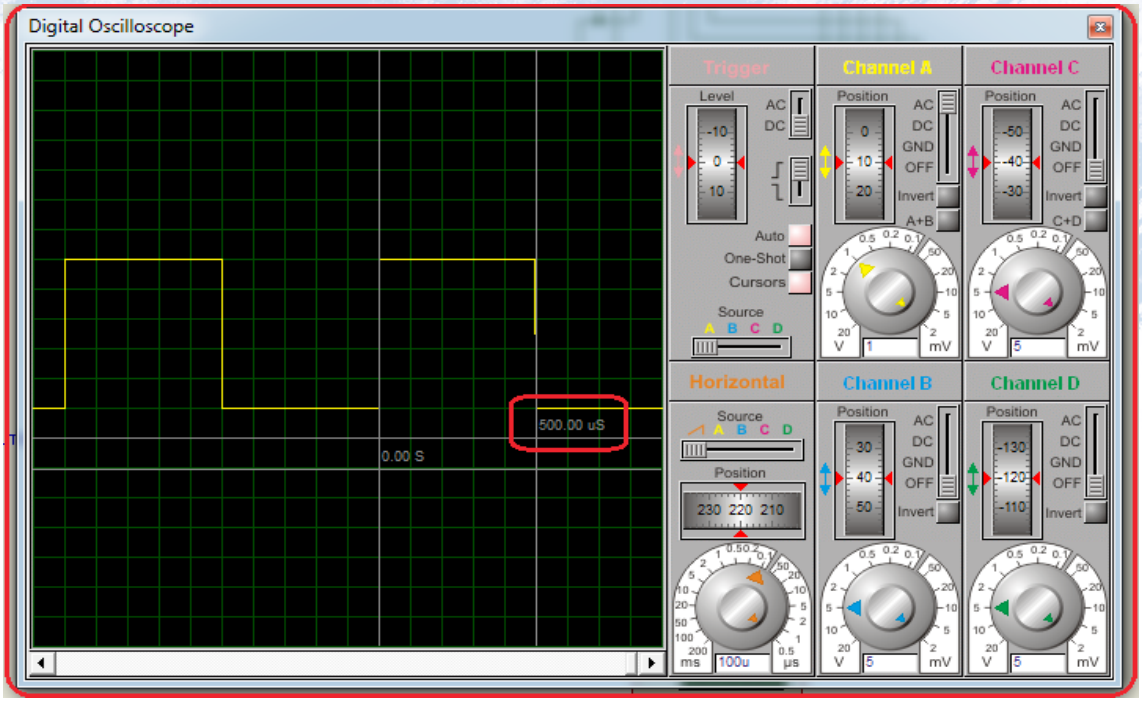


Figura 2.42. Señal de entrada. Pulso de 500us.
Elaborado por: Los Autores.

La figura 2.43, indica el resultado de la medida obtenida en el LCD.

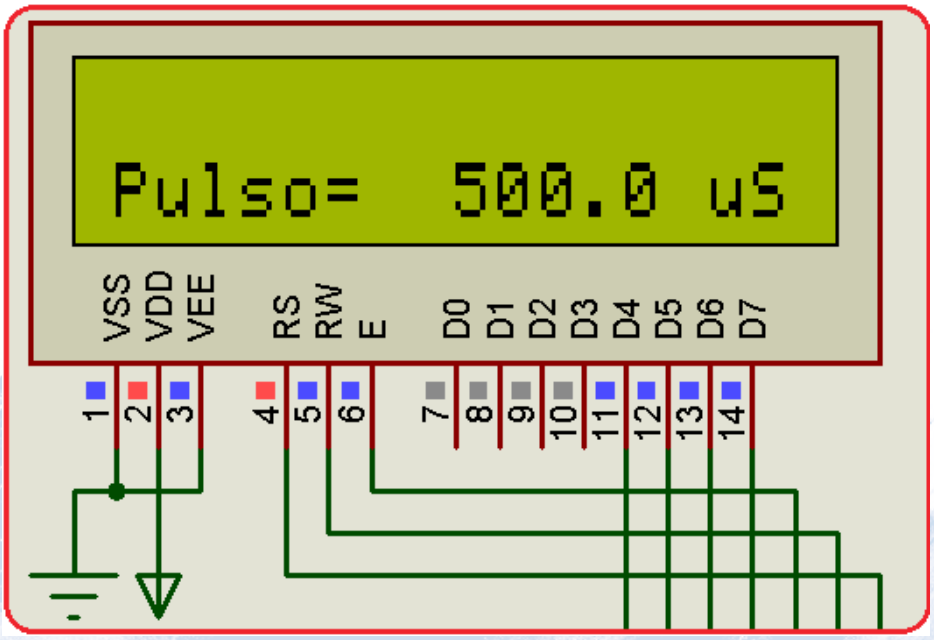


Figura 2.43. Captura del resultado obtenido.
Elaborado por: Los Autores.

Ejercicios propuestos

Diseñar un termómetro digital con un rango de temperatura de 0 a 50°C. Debe incluir toda la circuitería externa, sensores, amplificadores y la respectiva visualización en LCD.

Diseñar un controlador de temperatura on – off con histéresis. El elemento calefactor activarse a los 30°C y desactivarse a los 50°C. El diseño debe tener todos los elementos necesarios para la adquisición de señal y las visualizaciones necesarias.

Diseñe el programa y el circuito para controlar el avance y retroceso de un servomotor. Utilice los temporizadores para generar los pulsos correspondientes. Tome en cuenta que los servomotores trabajan con señales de 50Hz y pulsos de 1ms y 2ms para posicionar el eje en el extremo 1 (0 y en el extremo 2 (180°). Utilice el módulo CCP2.

Suponer que el servomotor ha sido trucado. Un servomotor trucado no se puede controlar la posición, pero si la velocidad y el sentido de giro. Elaborar un programa y el circuito para controlar la velocidad e inversión de giro del servomotor trucado.

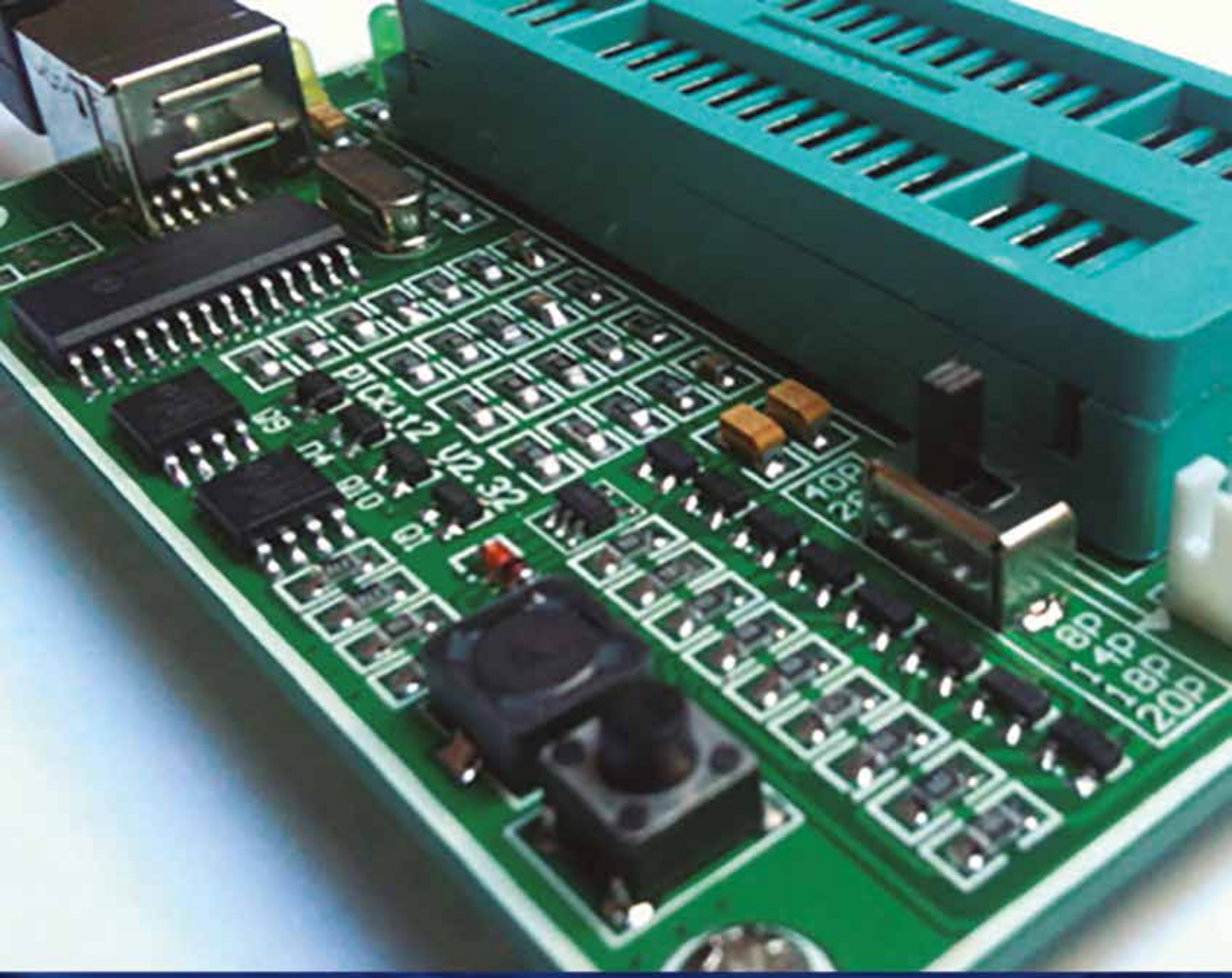
Realice el diseño de cruce por cero para disparar a un TRIAC. Diseñar el control para ángulos de disparo para 15^{o}, 30^{o} y 60^{o}. Los ángulos seleccionar por pulsador o teclado y visualizarlos en LCD.

Diseñe un circuito puente completo contralado con PWM mejorado funcionando en modo HALF_BRIDGE. Los valores son F_{OSC}= 4MHz, delay= 20 us, DC= 50%.

Escriba un programa para el modo FULL-BRIDGE activación H-L, L-H. Considere F_{OSC} = 4MHz, DC=60% y PWM de 5kHz. Mediante el osciloscopio virtual, observe las ondas generadas y compruebe si cumple con los resultados esperados.

Realice un programa y el circuito para generar mediante el módulo ECCP dos señales de 1kHz en fase con un tiempo con un retraso entre las dos de 10us. Un LED cambiará de estado cada 10 pulsos de la señal modulada.

Elabore un programa para controlar 4 servomotores. Los servos recibirán pulsos en forma secuencial asumiendo que están en el extremo 1 (0°) y avanzarán hasta alcanzar el extremo 2 (180°). Luego de esto retornarán a su posición original y continuarán el funcionamiento. Asuma que para avanzar los servos necesitan 10 pulsos de 1 ms y para retroceder 10 pulsos de 2 ms. La aplicación de los pulsos será a una frecuencia de 50 Hz. Utilice PWM mejorado, combinando los modos de funcionamiento del ECCP para obtener las señales de control moduladas.



CAPÍTULO 3

MÓDULOS DE COMUNICACIÓN DEL MICROCONTROLADOR

La mayor parte de microcontroladores de gama media y alta incluyen varios módulos de comunicación entre los que podemos citar los siguientes:

Transmisión serie: puerto serie sincrónico (SSP).

El SSP se utiliza en la comunicación con otros microcontroladores o con periféricos. Utilizan como interfaz de trabajo:

1. Transmisión serie: puerto serie sincrónico (SSP).

El SSP se utiliza en la comunicación con otros microcontroladores o con periféricos. Utilizan como interfaz de trabajo:

• Interfaz serie de periféricos (SPI). Se usa para comunicación entre microcontroladores de la misma o diferente familia, en tipo maestro-esclavo en modo full-dúplex.

• Interfaz Inter-Circuitos (I2C). Utilizada para comunicar microcontroladores, memorias EEPROM y periféricos en modo half dúplex.

2. Interfaz de comunicación serie (SCI) o receptor transmisor serie sincrónica (USART), o asincrónico universal (UART). Trabaja en modo asincrónico (full-dúplex) entre microcontroladores o con computadores y en modo half –dúplex con periféricos.

3. Otras formas de comunicación modernas que incluyen los PIC de gama alta en especial, se puede citar los módulos de USB, CAN (Controlled Área Network), LIN (Local Interconnect Nerwork), etc.

Para la comunicación serial del microcontrolador hemos tomado la información teórica, que proporciona el manual en inglés y en español del Compilador CCS y los sitios web:

• http://www.aquihayapuntes.com
• http://robotypic.blogspot.com
• http://aprendiendoelectronicafacil.blogspot.com

COMUNICACIÓN SERIAL ASINCRÓNICA USART

La comunicación serial en CCS se define a través de la directiva **#use** y utiliza el estándar RS232. El formato de la directiva es:

#use RS232(BAUD,BITS,PARITY=N,XMIT=PIN_B1,RCV=PIN_B2)

Los argumentos de la directiva son:

BAUD, en este argumento se establece la velocidad en baudios para la transmisión de datos por el puerto serie, por lo general, se emplea 9600 baudios.

BITS, es el número de bits que utiliza la transmisión. El estándar establece que pueden ser 8 o 9. Para la comunicación UART entre microcontroladores se usa 8 bits.

PARITY, establece un bit de paridad para la comprobación de errores. Por lo general, en UART está opción se deja en NO.

XMIT está opción configura la patilla del PIC por donde saldrán los datos. El microcontrolador PIC18F4550, utiliza el pin RC6.

RCV, configura la patilla del PIC por donde recibirán los datos. En el PIC18F4550 es el pin RC7.

FUNCIONES DE I/O SERIE RS232

El protocolo de comunicación serie RS232, permite el uso la función printf, getc, getchar, gets, puts y Kbhit para la entrada y salida de datos. De acuerdo al Manual de CCS, se tienen las siguientes funciones:

getc( ), getch( ), getchar( ), fgetc( )

Estas funciones esperan un carácter por la patilla RCV del dispositivo RS232 y retorna el carácter recibido.

Sintaxis:

```
value = getc()
value = fgetc(stream)
value=getch()
value=getchar()
```

stream, es un identificador de flujo (un byte constante)

putc( ), putchar( ), fputc( )

Estas funciones envían un carácter a la patilla XMIT del dispositivo RS232. Es preciso utilizar la directiva #USE RS232 antes de la llamada a esta función para que el compilador pueda determinar la velocidad de transmisión y la patilla utilizada. La directiva #USE RS232 permanece efectiva hasta que se encuentre otra que anule la anterior.

Sintaxis:

putc (cdata)
putchar (cdata)
fputc (cdata, stream)

cdata es un carácter de 8 bits. Stream es un identificador de flujo (un byte constante)

puts( ), fputs( )

Esta función envía cada carácter de string a la patilla XMIT del dispositivo RS232. Una vez concluido el envío de todos los caracteres la función envía un retorno de carro RC o RETURN (ASCII 13) y un avance de línea LF o LINE-FEED (ASCII 10).

Sintaxis:

puts (string).
fputs (string, stream)

string es una cadena constante o una matriz de caracteres (terminada en cero). Stream es un identificador de flujo (un byte constante)

printf([function], string, [values])

La función de impresión formateada PRINTF saca una cadena de caracteres al estándar serie RS-232 o a una función especificada. El formato está relacionado con el argumento que ponemos dentro de la cadena (string).

Cuando se usan variables, string debe ser una constante. El carácter % se pone dentro de string para indicar un valor variable, seguido de uno o más caracteres que dan formato al tipo de información a representar.

Si se coloca %% se obtiene a la salida un solo %. El formato tiene la forma genérica %wt, donde w es optativo y puede ser 1,2,...,9. Esto es para especificar cuántos caracteres son representados; si se selecciona el formato 01,...,09 se indican ceros a la izquierda, o también 1.1 a 9.9 para representación en punto flotante.

t es el tipo de formato y puede ser uno de los siguientes:

C Carácter
U Entero sin signo
x Entero en Hex (en minúsculas)
X Entero en Hex (en mayúsculas)

D Entero con signo
%e Real en formato exponencial(notación científica)
%f Real (Float)
Lx Entero largo en Hex (en minúsculas)
LX Entero largo en Hex (en mayúsculas)
Lu Decimal largo sin signo
Ld Decimal largo con signo
% Simplemente un %

kbhit()

Devuelve verdadero cuando un carácter es recibido en el buffer del hardware RS232 o cuando se envía el primer bit en el pin RCV en caso del software RS232. Útil para detectar sin esperar en getc.

Ejercicio 3.1. Comunicación simplex entre dos PIC.

En el siguiente ejercicio el LED1 se activará al accionar el pulsador S1 y se apaga al accionar el pulsador S2 que están conectados en el PIC U1 (Transmisor). El LED1 se encuentra conectado en el PIC U2 (Receptor), como se indica en la figura 3.1.

Consideraciones del programa

Definimos la comunicación serial mediante la directiva:

```
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
```

La velocidad de TX se establece en 9600 baudios, se utiliza 8 bits y ningún bit de paridad. Los terminales para recibir o transmitir los datos se obtienen de las hojas del fabricante, así para el PIC18F4550, RC6 es TX y RC7 es RX.

Para colocar el dato en la línea de transmisión se utiliza la instrucción:

```
putc(valor);
```

Según se accione los pulsadores se envían los números 1 o 2, que son evaluados en el receptor para prender o pagar el LED.

La recepción de datos se realiza mediante la interrupción de recepción:

```
#int_rda
```

Cada vez que existe un cambio en el bus serial se activa la interrupción para receptar el dato enviado desde el transmisor. El valor es leído y asignado a una variable:

valor= getc();

Además en el receptor se debe utilizar la misma velocidad de transmisión y definir los pines de recepción y transmisión que utiliza el microcontrolador.

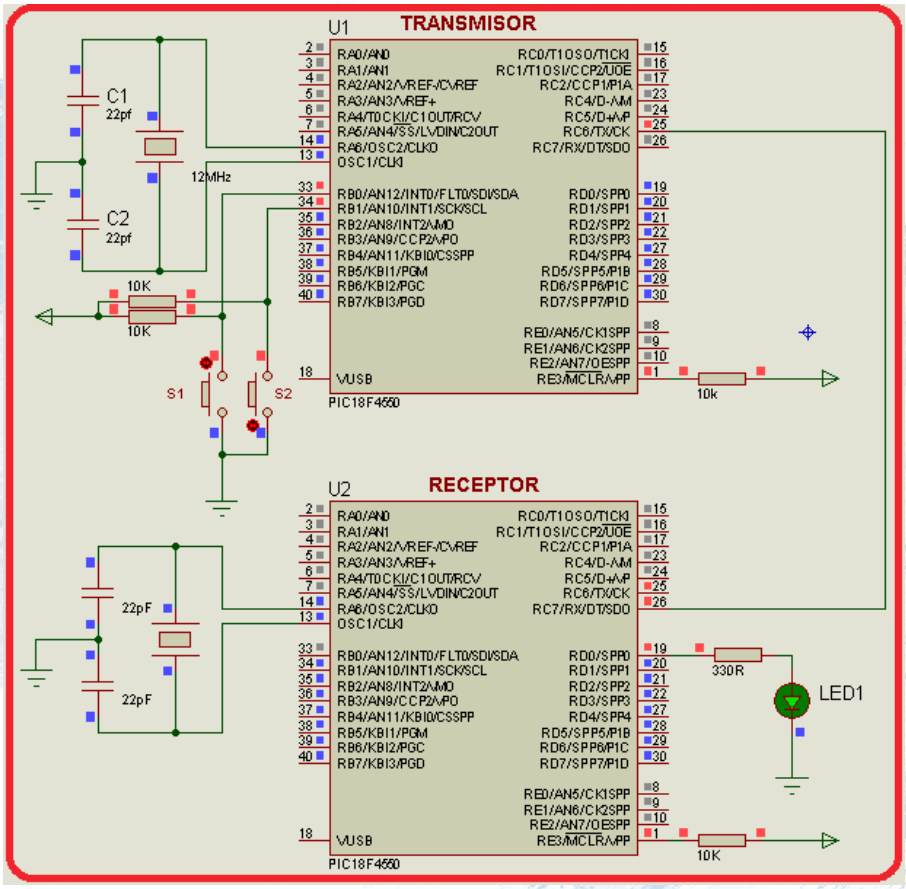


Figura 3.1. Conexión de dos PIC para comunicación simplex.
Elaborado por: Los Autores.

PROGRAMA PARA EL TRANSMISOR:

```
#include <18f4550.h>                          //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN       //Configuración de fusibles.
#use delay (clock=12000000)                   // F_OSC = 12 MHz.
#include <stdlib.h>                           //Librería stdlib.h.
#byte port_b = 0xF81                          //Identificador del Puerto B.
#byte tris_b = 0xF93                          //Identificador del Registro TRISB.
                                              //Directiva para comunicación serial  RS232.
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
#define s1 bit_test(port_b,0)                 //Define el pulsador S1 para el Puerto B0.
#define s2 bit_test(port_b,1)                 //Define el pulsador S2 para el Puerto B1.
int valor = 0;                                //Variable para enviar datos.

void main(void)                               //Función principal main
{
   set_tris_b(0x03);                          //Fija el Puerto B0 y B1 como entrada.

  while( TRUE ){                              //Bucle infinito.
  if (s1 == 0 ){
     valor= 1;                                //Valor = 1 prende el LED1.
     putc(valor);                             //Coloca el dato en la línea TX.
     delay_ms(10);
    }
    if (s2 == 0 ){
     valor= 2;                                //Valor = 2 apaga el LED1.
     putc(valor);                             //Coloca el dato en la línea TX.
     delay_ms(10);
    }
  }                                           //Fin de bucle infinito.
}                                             //Fin del main.
```

PROGRAMA PARA EL RECEPTOR:

Es necesario habilitar la interrupción cuando los datos están disponibles #int_rda. Esta interrupción facilita la detección del dato en el instante que ha recibido el PIC.

```
#include <18f4550.h>                        //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN     //Configuración de fusibles.
#use delay (clock=12000000)                 // F_OSC = 12 MHz.
#include <stdlib.h>                         //Librería stdlib.h.
#byte port_b = 0xF81                        //Identificador del Puerto B.
#byte tris_b = 0xF93                        //Identificador del Registro TRISB.
        //Directiva para comunicación serial  RS232.
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
int valor ;                                 //Variable para enviar datos.
        //Interrupción de recepción de datos disponibles.
#int_rda
void rda_isr()                              //Función para la interrupción.
{
 valor= getc();                             //Asigna el valor de recepción a la variable valor.
}

void main(void)                             //Función principal main.
{
        //Habilitación de la interrupción cuando los datos están disponibles.
  enable_interrupts(int_rda);
  enable_interrupts(global);                //Habilitación de la interrupciones globales.
  set_tris_b(0x01);                         //Fija el Puerto RB0 como entrada.
  set_tris_d(0x00);                         //Fija el puerto RD como salida.

 while( TRUE ){                             //Bucle infinito.

  if (valor==1){                            //¿Hay dato?.
    bit_set(port_d,0);                      //Activa LED.
  }
  if (valor==2){                            //¿Hay dato?.
     bit_clear(port_d,0);                   //Desactiva el LED.
     valor = 0;                             //Resetea la variable valor.
     delay_ms(10);
   }
  }                                         //Fin de bucle.

        }                                   //Fin del main.
```

Ejercicio 3.2. Comunicación half - dúplex entre dos PIC.

En el ejercicio propuesto, con el pulsador S1 del PIC U1, activa al LED – D2 del PIC U2. Transcurrido 500 ms., responde el PIC U2 activando el LED – D1, indicando que recibió el dato. Con el pulsador S2, se desactivan los dos LEDS. La figura 3.2, muestra el diagrama de conexiones del circuito para transmisión – recepción de datos por los dos PIC.

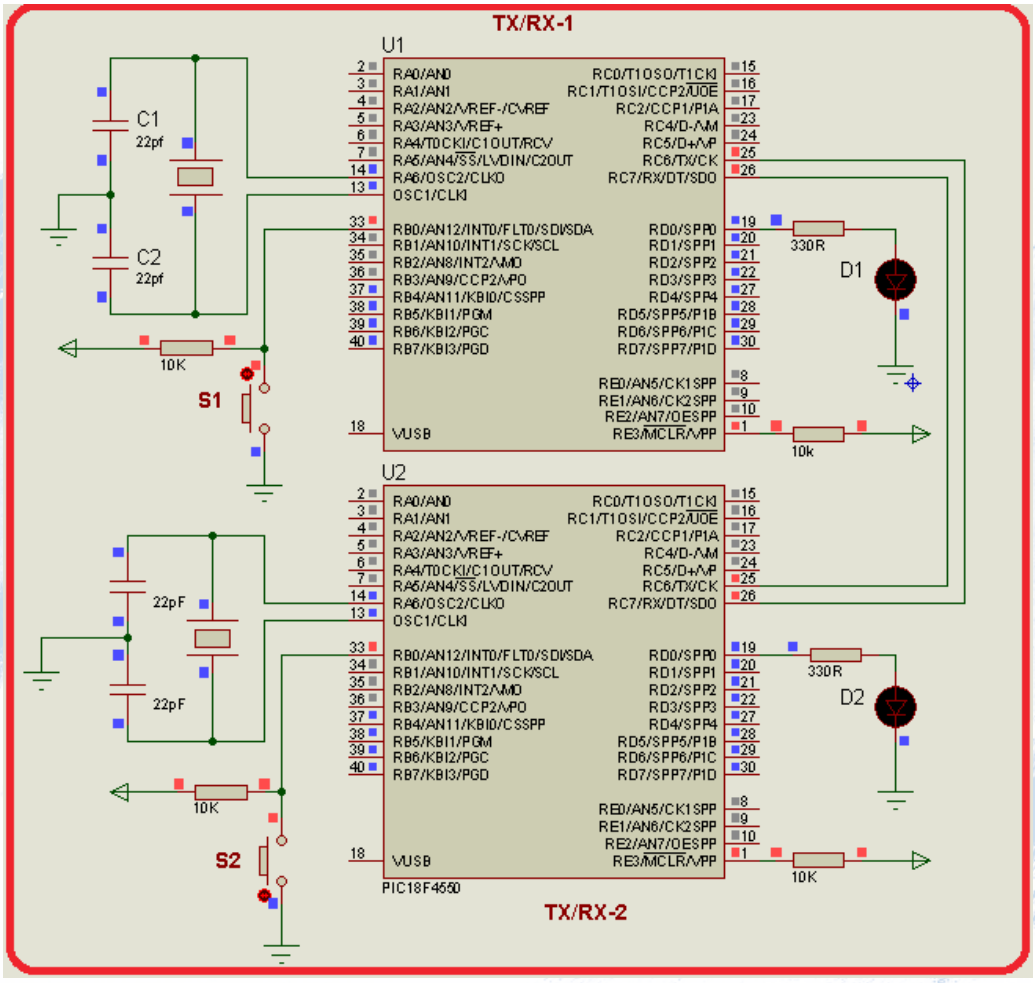


Figura 3.2. Comunicación half - duplex entre dos PIC.
Elaborado por: Los Autores.

PROGRAMA PARA TX/RX-1:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=12000000)             // F_OSC = 12 MHz.
#include <stdlib.h>                     //Librería stdlib.h.
#byte port_b = 0xF81                    //Identificador del Puerto B.
#byte port_d = 0xF83                    //Identificador del Puerto D.
#byte tris_b = 0xF93                    //Identificador del Registro TRISB.
#byte tris_d = 0xF95                    //Identificador del Registro TRISD.
                                        //Directiva para comunicación serial  RS232.
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
int valor = 0 ;                         //Variable para enviar datos.
int recibe= 0;                          //Variable para recibir datos.
                                        //Interrupción de recepción da datos disponibles.
#int_rda
void rda_isr()                          //Función para la interrupción.
{
 recibe= getc();                        //Asigna el valor de recepción a la variable recibe.
}

void main(void)                         //Función principal main.
{
                                        //Habilitación de la interrupción cuando los datos están disponibles.
  enable_interrupts(int_rda);           //Habilitación de la interrupción RDA.
  enable_interrupts(global);            //Habilitación de la interrupciones globales.
  set_tris_b(0x01);                     //Fija el Puerto RB0 como entrada.
  set_tris_d(0x00);                     //Fija el puerto RD como salida.

 while( TRUE ){                         //Bucle infinito.
        //envía datos al PIC 2.
 if (s1 == 0 ){                         //¿Esta accionado el pulsador S1?
    valor= 1;                           //Asigna valor = 1.
    putc(valor);                        //Envía el dato al PIC 2.
    delay_ms(10);
  }

                                        //recibe la respuesta del PIC2
  if (recibe==1){
    bit_set(port_d,0);                  //Activa LED-D1.
   }
  if (recibe==2){
    bit_clear(port_d,0);                //Apaga LED-D1.
    recibe==0;
  }
 }                                      //Fin de bucle.
}                                       //Fin del main.
```

PROGRAMA PARA TX/RX-2:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=12000000)             //F_OSC = 12 MHz.
#include <stdlib.h>                     //Librería stdlib.h.
#byte port_b = 0xF81                    //Identificador del Puerto B.
#byte port_d = 0xF83                    //Identificador del Puerto D.
#byte tris_b = 0xF93                    //Identificador del Registro TRISB.
#byte tris_d = 0xF95                    //Identificador del Registro TRISD.
                                        //Directiva para comunicación serial  RS232.
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
int ingresa= 0 ;                        //Variable para recibir datos.
int respuesta= 0;                       //Variable para responder datos.
                                        //Interrupción de recepción da datos disponibles.
#int_rda
void rda_isr()                          //Función para la interrupción.
{
 ingresa= getc();                       //Asigna el valor de recepción a la variable ingresa.
}

void main(void)                         //Función principal main.
{
                                        //Habilitación de la interrupción cuando los datos están disponibles.
  enable_interrupts(int_rda);           //Habilitación de la interrupción RDA.
  enable_interrupts(global);            //Habilitación de la interrupciones globales.
  set_tris_b(0x01);                     //Fija el Puerto RB0 como entrada.
  set_tris_d(0x00);                     //Fija el puerto RD como salida.

 while( TRUE ){                         //Bucle infinito.
        //recepta  datos del PIC 1.
  if (ingresa==1){                      //¿Hay dato?.
    bit_set(port_d,0);                  //Enciende LED – D2.
    respuesta= 1;                       //variable respuesta = 1, para enviar al PIC 1.
    delay_ms(500);                      //Retardo de 500 ms.
    putc(respuesta);                    //Envía el dato al PIC 1.
    delay_ms(10);
  }
    if (s1== 0){                        //¿Si el pulsador S2 es activado?.
    bit_clear(port_d,0);                //Apaga LED – D2.
    ingresa=0;                          //Encera la variable ingresa.
    respuesta= 2;                       //respuesta= 2
    putc(respuesta);                    //Envía el dato respuesta al PIC 1.
   }
 }                                      //Fin de bucle.
}                                       //Fin del main.
```

Comunicación sincrónica I2C

El protocolo I2C fue desarrollado por la empresa Philips, ahora llamada NXP, se caracteriza porque la interfaz puede comunicarse con varios dispositivos electrónicos, reduciendo el cableado y ahorrando el número de patillas entre las conexiones de los chips.

Características básicas del protocolo I2C

- Velocidad de transmisión estándar de 100 KBaudios. En modo de alta velocidad hasta 400 Kbaudios.
- El bus de comunicación utiliza dos cables:
- SDA. Serial Data Line. Línea para transmitir los datos.
- SCL o SCK. Serial Clock Line. Línea para la señal de reloj.

Las dos líneas se conectan a la alimentación (V_{CC}) a través de resistencias Pull-UP y las tierras deben conectarse entre sí formando una masa común.

- El protocolo de comunicación es del tipo Maestro-Esclavo.
- Es del tipo de comunicación half dúplex bidireccional, por la misma línea pero no simultáneo.
- Cada dispositivo conectado al bus se identifica por una única dirección, el protocolo admite utilizar 7 o 10 bits para el direccionamiento, aunque lo normal es que sean siete.

La directiva que controla las funciones para manejar el protocolo de comunicación I2C, es:

#use I2C (opciones)

Las opciones que se incluyen en la directiva son:

MASTER: Establece el modo maestro

SLAVE: Establece el modo esclavo.

ADDRES=nn: Especifica la dirección en modo esclavo.

SCL=pin: Especifica el pin SCL.

SDA=pin: Especifica el pin SDA.

FAST: Utiliza velocidad alta.

SLOW: Utiliza velocidad baja.

RESTART_WDT: Reinicia o restaura el temporizador Watch Dog, mientras espera una lectura.

FORCE_SW: Utiliza las funciones I2C del Software.
FORCE_HW: Utiliza las funciones I2C del hardware.

Una vez definida la directiva #*use I2C,* se utiliza las siguientes funciones para controlar el bus I2C. Las más importantes son:

i2c_isr_state(). Detecta el estado del bus i2c. Devuelve un int8.

i2c_poll(). Esta función devuelve TRUE si el hardware tiene un byte recibido en el buffer. Cuando se devuelve un TRUE, una llamada a I2C_ READ() devolverá inmediatamente el byte que se recibió.

i2c_read(). Lee el buffer del dispositivo que actúa como slave.

i2c_slaveaddr(). Esta función establece la dirección para la interfaz I2C en modo esclavo.

i2c_start(). Emite una condición de arranque en el modo maestro I2C. Después de la condición de arranque el reloj se mantiene bajo hasta que i2c_write () es llamado. Si otro i2c_start se llama en la misma función antes de llamar a un i2c_stop, se emite una condición especial de reinicio.

i2c_stop(). Emite una condición de parada cuando está en el modo maestro I2C.

i2c_write(). Envía un byte a través del interfaz I2C. En el modo master esta función generará un reloj con los datos y en modo esclavo esperará al reloj del maestro.

Ejercicio 3.3. Comunicación serial I2C PIC18 – PIC16

En el siguiente ejercicio se va a realizar una comunicación entre un PIC18 (PIC18F4550) como MASTER y un PIC16 (PIC16F887) como SLAVE. Se debe tener en cuenta los pines SDA y SLC que utilizan estos micros. La tabla 3.1, presenta las líneas utilizadas para el SDA y SLC.

LÍNEA	PIC18F4550	PIC16F887
SDA	RB0	RC4
SLC	RB1	RC3

Tabla 3.1. Líneas correspondientes a SDA y SLC, para los PIC18F4550 y PIC16F887.
Elaborado por: Los Autores.

El ejercicio consiste en enviar datos del puerto D para ser mostrados en un LCD que está conectado en el PIC SLAVE. El estado del puerto D cambia

al accionar los pulsadores del puerto D. Por ejemplo si se acciona el pulsador del puerto RD3, el valor será $2^2 = 4$. Para cumplir con esto, el dato leído del puerto D se invierte y asigna a la variable enviar:

enviar= ~port_d;//Asigna enviar el valor invertido del puerto d.

PROGRAMA DEL PIC MASTER:

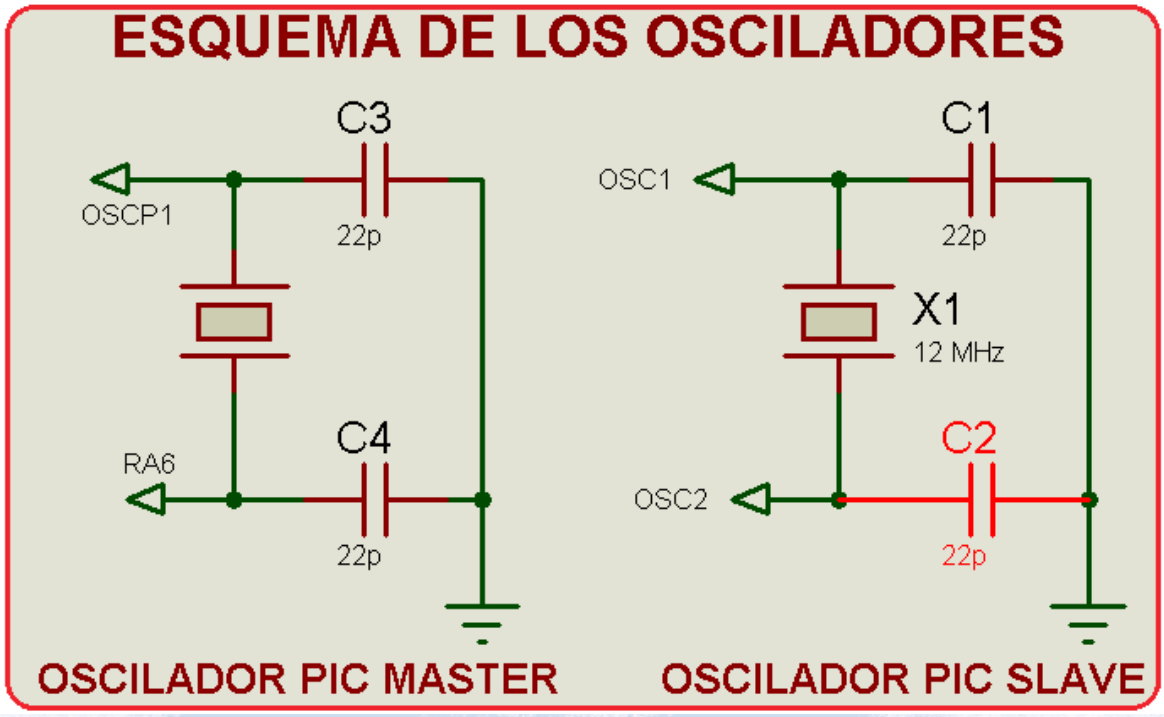
```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT, NOPBADEN //Configuración de fusibles.
#use delay (clock=12000000)             // F_OSC = 12 MHz.
#include <stdlib.h>                             //Librería stdlib.h.
          //Directiva para comunicación serial  I2C.
#use i2c(MASTER, SDA=PIN_B0, SLOW, SCL=PIN_B1, NOFORCE_SW)
#byte port_a = 0xF80                    //Identificador del Puerto A.
#byte port_b = 0xF81                    //Identificador del Puerto B.
#byte port_d = 0xF83                    //Identificador del Puerto D.
#byte tris_a = 0xF92                    //Identificador del Registro TRISA.
#byte tris_b = 0xF93                    //Identificador del Registro TRISB.
#byte tris_d = 0xF95                    //Identificador del Registro TRISD.
#define s1 bit_test(port_d,0)           //Identificador del Puerto D.0 como s1.
int enviar;                             //Dato a transmitir
                                        //Función para enviar datos.
void envio_I2C (){
    i2c_start();                        //Comienzo de la comunicación I2C.
    i2c_write(0xa0);                    //Dirección del PIC esclavo 0xA0..
    delay_ms(100);                      //Retardo de 100 ms.
    i2c_write(enviar);                  //Envía dato.
    i2c_stop();                         //Finalización de la transmisión.
 }

void main(){                            //Función principal main.
 set_tris_a(0xFE);                      //Fija el Puerto A.0 como salida.
 set_tris_b(0XFF);                      //Fija el Puerto d como entrada.
 set_tris_d(0xFF);                      //Fija el Puerto d como entrada.

  setup_adc_ports(NO_ANALOGS|VSS_VDD);       //Fija el Puerto A como I/O digitales.

  while (TRUE){                         //Bucle infinito.
                                        // Lectura del puerto D.
    enviar= ~port_d;                    //Asigna enviar el valor invertido del puerto d.
     if(enviar !=0){                    //Si se acciona un pulsador.
        envio_I2C();                    //Se envía el dato.
                                        //Indicador que el dato se envía.
        bit_set(port_A,0);              //Activa el LED.
        delay_ms(500);                  //Retardo de 500ms.
        bit_clear(port_A,0);            //Apaga el LED.
    }
  }                                     //Fin de bucle.
}                                       //Fin del main.
```

En el puerto están conectados los pulsadores de tal forma que en reposo todo el puerto está en alto, es decir, tiene un valor de 255. En este ejercicio se envía el valor del puerto al momento que se acciona el pulsador. Por tanto, se indicará el 1, 2, 4, 8, 16,32 y 64 respectivamente. Cuando se accione el pulsador del puerto RB7 el dato invertido del puerto D es 128 y en este caso en el PIC SLAVE se ha determinado que el LCD se limpie.

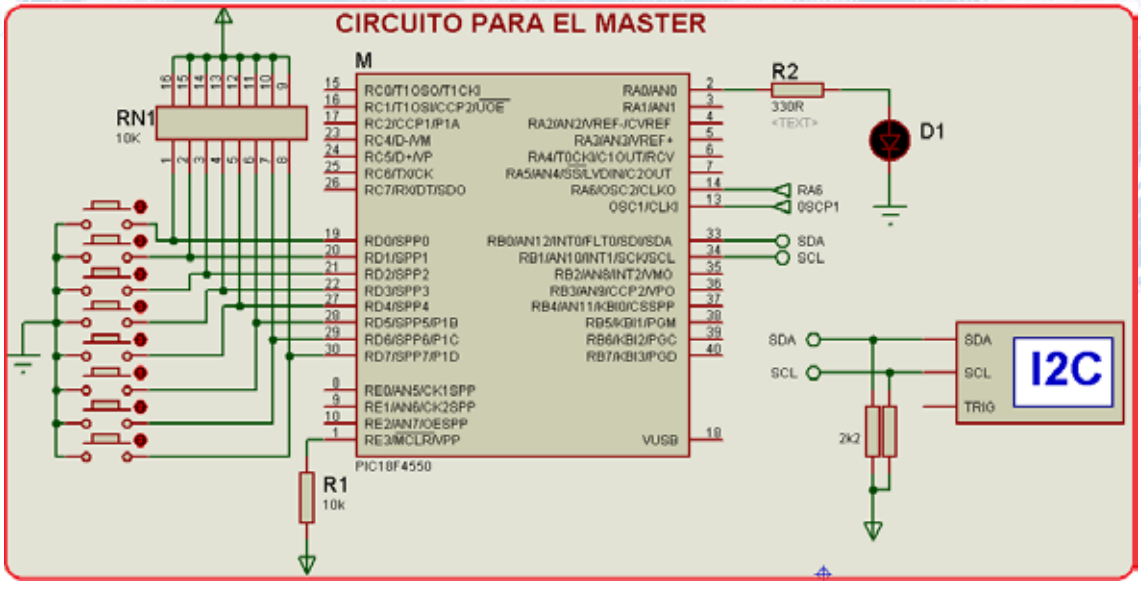
Programa del PIC SLAVE:

```
#include <16F887.h>              //Librería para usar el PIC16F887A.
#fuses HS,WDT,NOPROTECT,NOPUT    //Configuración de fusibles.
#use delay (clock=12000000)      //F_OSC = 12 Mhz.
#include <stdlib.h>              //Librería stdlib.h.
#use standard_io(b)              //Usa librería estándar para el puerto B.
#use standard_io(c)              //Usa librería estándar para el puerto C.
#use standard_io(d)              //Usa librería estándar para el puerto D.
                                 //Directiva para comunicación serial  I2C.
#use I2C(SLAVE, SDA=PIN_C4,SLOW, SCL=PIN_C3, ADDRESS=0xa0, NOFORCE_SW)

#include <lcd.c>                 //Usa driver para manejar el LCD.

void main() {                    //Función principal main.
set_tris_d(0x00);                //Fija el Puerto D como salida.
set_tris_b(0xff);                //Fija el Puerto B como entrada.

  int dato;                      //Dato recibido.
  lcd_init();                    //Inicializa el LCD.

  while (TRUE) {                 //Bucle infinito.
                                 //Recepción por comunicación I2C
    lcd_gotoxy(1,1);             //Posiciona el cursor en la columna 1 y fila 1.
    lcd_putc("DATO RECIBIDO ES");      //Mensaje.
    if(i2c_poll()) {             //Detecta si hay dato en el buffer.
      dato=i2c_read();           //Lee el dato y asigna a la variable respectiva.
     if (dato>= 1 & dato<=96 ){  //Compara los datos para ser mostrados en el LCD.
       lcd_gotoxy(5,2);          //Posiciona el cursor en la columna 5 y fila 2.
       printf(lcd_putc," %d ",dato);//Muestra dato recibido por pantalla
     }
      if (dato==128) lcd_putc("\f");  //Si es 128 borra la pantalla LCD.
    }
  }                              //Fin del ciclo.
}                                //Fin del main.
```

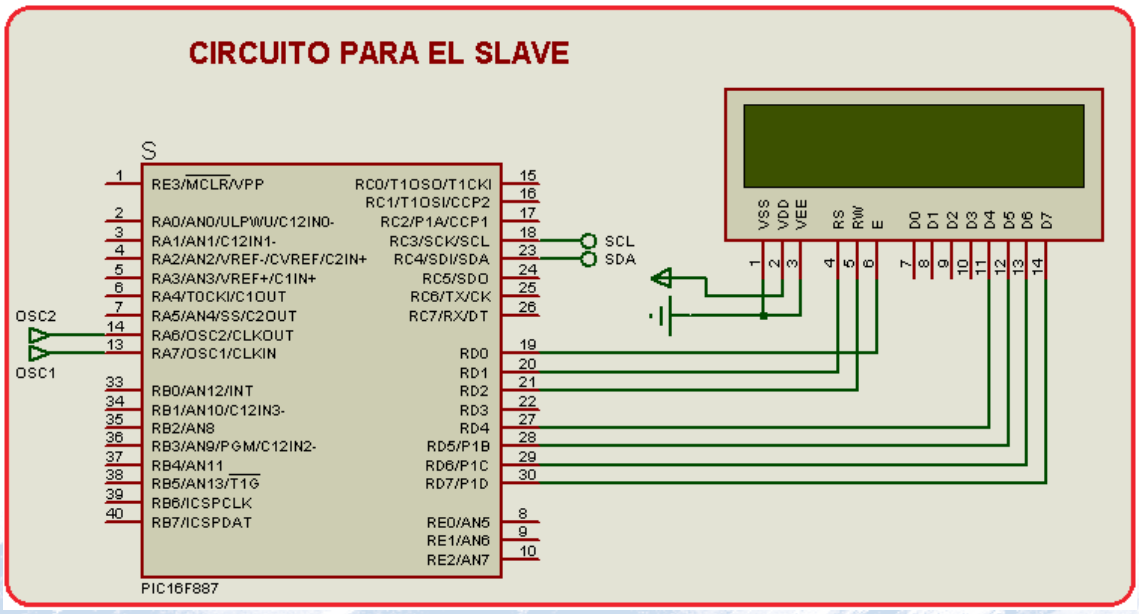
La figura 3.3, muestran los circuitos a) circuitos osciladores, b) MASTER y c) SLAVE



a). Circuitos osciladores, para los microcontroladores.



b) Circuito del PIC MASTER.



c) Circuito del PIC SLAVE.
Figura 3.3. Comunicación I2C PIC-PIC, MASTER-SLAVE.
Elaborado por: Los Autores.

Comunicación PIC con memoria EEPROM utilizando protocolo I2C.

Las memorias EEPROM de Microchip serie 24XXX tienen un bus I2C, han ganado un amplio espacio en el desarrollo de hardware por la durabilidad y facilidad de uso que presentan y en especial cuando se requiere guardar datos en una memoria EEPROM externa. Estas memorias vienen con diferentes capacidades, pero por lo general, tienen en común las líneas del bus I2C: SLC y SDA. Además las líneas de dirección de la memoria "address" A0, A1, A2 y un pin para habilitación de la protección de escritura (WP, Write Protected) del bloque de la memoria, como se indica en la figura 3.4.

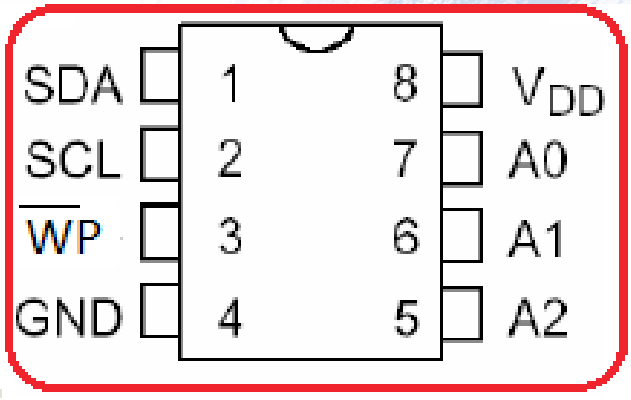


Figura 3.4. Ejemplo típico de una EEPROM I2C.
Fuente: 64K I2C™ Serial EEPROM. Data Sheet. Microchip Technology INC.

Ejercicio 3.4. Lectura y escritura de una EEPROM mediante comunicación I2C con PIC.

En un vector tipo char vamos a almacenar "Comunicación I2C". Este dato guardamos en la memoria EEPROM M24512. En el LCD indicará que se está escribiendo en la memoria. Luego que se ha finalizado la escritura de la EEPROM, se lee carácter por carácter y se indica en el LCD.

PROGRAMA:

```
void main()          /                    /Función principal main.
{
        short int status;      /           /Detecta el estado del bus.
        lcd_init();   /                    /Inicializa el LCD.
        lcd_putc("PIC - EEPROM 24512");//Muestra el mensaje en el LCD.

                    //Escribe los datos desde la dirección 0 hasta la 15.
        for(n= 0; n< 15; n++){
                    //Escribe el dato que indica "n" en el vector "mensaje".
i          2c_start();                     //Inicio de la transmisión...
                    //... con la dirección I2C correspondiente a la eeprom en modo escritura
    i     2c_write(0xA0);
          i2c_write(address>>8);           //Envía parte alta dirección.
          i2c_write(address);              //Envía parte baja dirección.
          i2c_write(mensaje[n]);           //Envía el dato "n" del mensaje.
          i2c_stop();                      //Fin de la transmisión.
          address++;           /           /Incrementa la dirección.
          i2c_start();                     //Reinicia  la comunicación...
        status=i2c_write(0xa0);            //... para lectura de bit ACK (escritura correcta).

    while(status==1){                      //Si es 1 espera a que responda EEPROM.
      i2c_start();    /                    /Reinicia  la comunicación...
      status=i2c_write(0xa0);              //Asigna a "status" el estado del bus.
    }
      lcd_gotoxy(1, 2);
      if(!p){                      /       /Parpadeo para indicar que está "escribiendo " en la memoria.
        lcd_putc("Escribiendo...");
        p= 1;
      }
      else{
        lcd_putc("           ");     /     /Borra el mensaje Escritura cada….
        p= 0;
      }
        delay_ms(200);    /              /….200 ms.
      }
//Lectura de datos de la EEPROM:
      lcd_putc("           ");       /     /Borra el mensaje contenido de la segunda fila.
      lcd_gotoxy(1, 2);      /             /Fija el cursor en la columna 1  fila 2.
//Lee los datos comenzando de la dirección 0 hasta la 15.
    for(address= 0; address< 15; address++){
//Lee el dato de la dirección address y asigna a la variable "dato".
i2c_start();          /                    /Inicio de la transmisión...
//...con la dirección correspondiente a la EEPROM en modo escritura.
```

```
i    2c_write(0xA0);                  //Modo de escritura de la EEPROM.
i    2c_write(address>>8);     /      /Envía parte alta dirección.
 i2c_write(address);                  //Envía parte baja dirección.
 i2c_start();                         //Reinicio...
 i2c_write(0xA1);                     //...con EEPROM en modo lectura.
 dato=i2c_read(0);                    //Lectura del dato de la dirección enviada.
 i2c_stop();                          //Fin de la transmisión.
 printf(lcd_putc, "%c", dato);  /     /Muestra los datos leídos cada.....
```

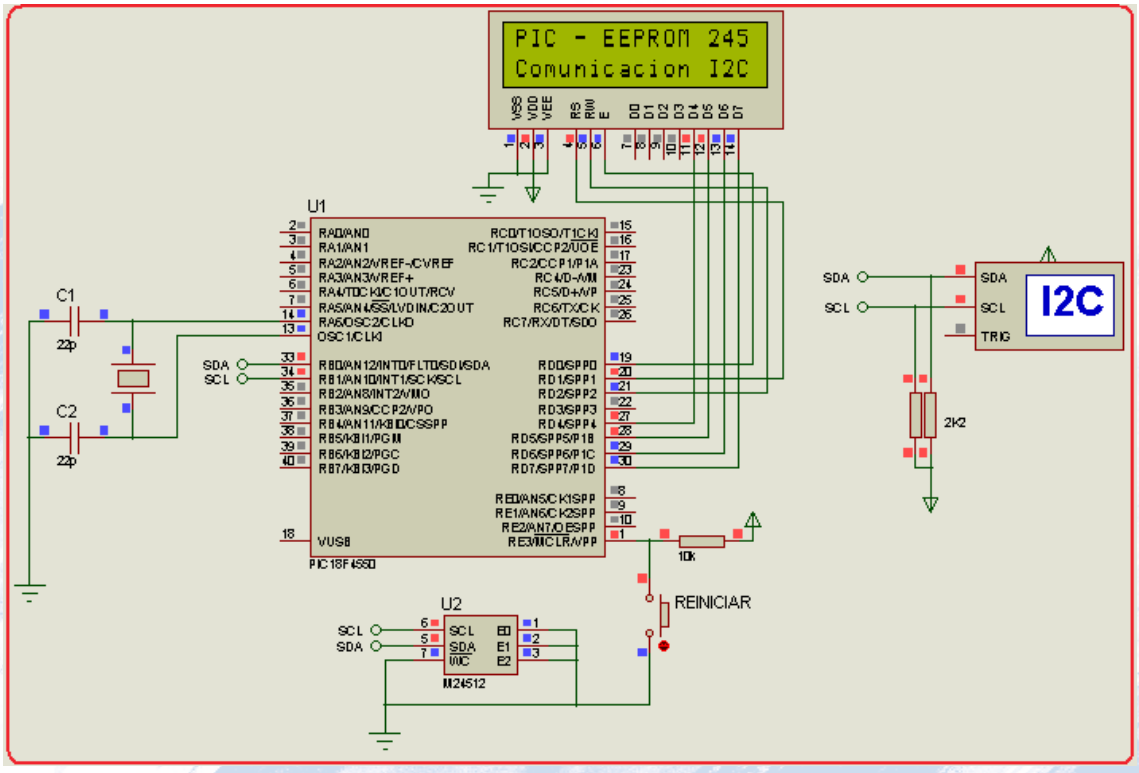


Figura 3.5. Comunicación I2C entre PIC con memoria EEPROM M24512.

Comunicación SPI de PIC con memoria EEPROM externa

Las memorias serie 25XXX de Microchip tienen una mayor capacidad y velocidad de comunicación que las serie 24XX, porque utilizan el protocolo SPI (Serial Peripheral Interface).

Una de las memorias más populares que trabaja con protocolo SPI es la 25080 en sus diferentes versiones como indica la tabla 3.2 de Microchip.

Part Number	Vcc Range	Max. Clock Frequency	Temp. Ranges
25AA080	1.8-5.5V	1 MHz	I
25LC080	2.5-5.5V	2 MHz	I
25C080	4.5-5.5V	3 MHz	I,E

Tabla 3.2. Memorias EEPROM de la serie 25080.
Fuente: 64K I2C™ Serial EEPROM. Data Sheet. Microchip Technology INC.

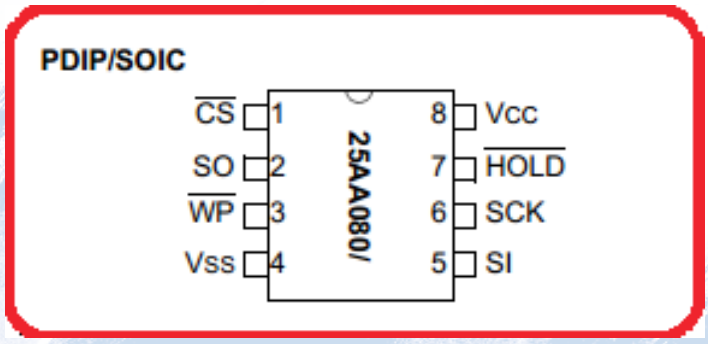


Figura 3.6. Diagrama de la memoria EEPROM 25AA080.
Fuente: 64K I2C™ Serial EEPROM. Data Sheet. Microchip Technology INC.

Las señales de la memoria son:

- **CS:** Selección del chip.
- **SO:** Salida de datos.
- **SI:** Entrada de datos.
- **SCK:** Reloj de entrada.
- **WP:** Protección de escritura.
- **HOLD:** Sostenimiento. Sirve para pausar la comunicación.

CCS ha desarrollado varias librerías para controlar este tipo de memorias. La librería 25080.c se debe incluir para este caso. Revisando el código de la librería se observa que están definidos los pines B0, B1, B2 y B4 para utilizar en el microcontrolador como indica la figura 3.7.

```
#define EEPROM_SELECT PIN_B0
#define EEPROM_CLK    PIN_B1
#define EEPROM_DI     PIN_B2
#define EEPROM_DO     PIN_B4
```

Figura 3.7. Extracto del código del driver 25080.c. Definición del pines del micro.
Elaborado por: Los Autores.

Las funciones que provee la librería son:

- **init_ext_eeprom();** Llama a las otras funciones antes de ser usadas.
- **write_ext_eeprom(address, data);** Escribe el byte data en la dirección address.
- **data = read_ext_eeprom(addrres);** Lee el byte data de la dirección address
- **b = ext_eeprom_ready();** Devuelve TRUE si la memoria EEPROM está lista para recibir los códigos de operación.

Ejercicio 3.5. Comunicación SPI entre PIC18F4550 y memoria EEPROM 25LC080A.

En un vector tipo char vamos a almacenar "Comunicación SPI". Este dato guardamos en la memoria EEPROM. En el LCD indicará que se está escribiendo en la memoria. Luego que se ha finalizado la escritura de la EEPROM, se lee carácter por carácter y se indica en el LCD.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#fuses HS,WDT,NOPROTECT,NOPUT,NOPBADEN,MCLR          //Configuración de fusibles.
#use delay (clock=12000000)             //FOSC = 12 MHz.
#include <25080.C>                      //Incluye librería para manejo de memoria 25080.
#include <lcd.c>                        //Incluye librería para manejar LCD

int dato, n;                            //dato para leer la EEPROM, n contador del número de caracteres.
int address= 0;                         //Dirección inicial de la EEPROM 0.
char mensaje[]= "Comunicacion SPI";     //Datos para grabar en la EEPROM.
int1 p;                                 //Controla parpadeo de escritura.

void main()                             //Función principal main.
{
      init_ext_eeprom();                //Inicializa la memoria externa EEPROM.
      lcd_init();                       //Inicializa el LCD.
      lcd_putc("EEPROM 25LC080");       //Muestra el mensaje en el LCD.

//Escribe los datos desde la dirección 0 hasta la 15.
      for(n= 0; n< 15; n++){
//Escribe el dato que indica "n" en el vector "mensaje".
            write_ext_eeprom(address, mensaje[n]);
            address++;
            lcd_gotoxy(1, 2);
            if(!p){                     //Parpadeo para indicar que está "escribiendo " en la memoria.
                  lcd_putc("Escribiendo...");
                   p= 1;
```

```
            }
            else{
                lcd_putc("            ");  //Borra el mensaje Escritura cada….
                p= 0;
            }
            delay_ms(200);                 //….200 ms.
    }

      //Lectura de datos de la EEPROM:
   lcd_putc("            ");              //Borra el mensaje contenido de la segunda fila.
   lcd_gotoxy(1, 2);                      //Fija el cursor en la columna 1  fila 2.
      //Lee los datos comenzando de la dirección 0 hasta la 15.
  for(address= 0; address< 15; address++){
      //Lee el dato de la dirección address y asigna a la variable "dato".
      dato= read_ext_eeprom(address);
      printf(lcd_putc, "%c", dato);       //Muestra los datos leídos cada…..
      delay_ms(300);                      //……300ms.
      }
}                                         //Fin del main.
```

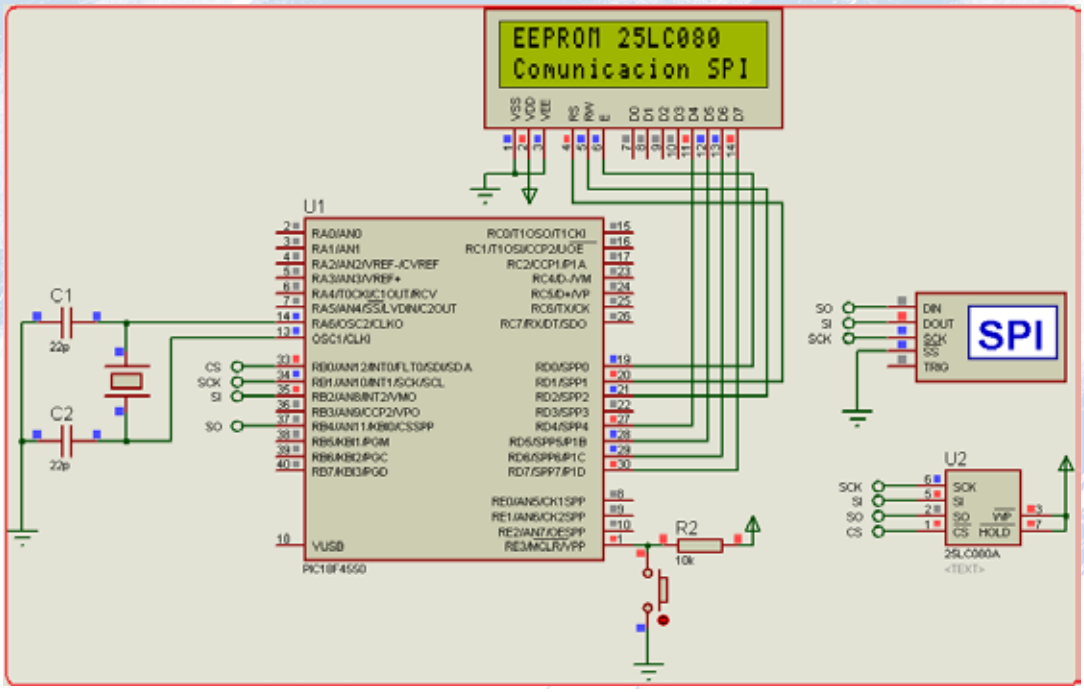


Figura 3.8. Esquema de conexiones para comunicación SPI PIC-EEPROM.
Elaborado por: Los Autores.

Comunicación USB

La comunicación bus universal serial (USB), prácticamente ha venido reemplazar a la mayoría de las formas de comunicación entre un PC y los dispositivos externos.

Los microcontroladores PIC de la gama alta vienen embebidos módulos de comunicación USB, lo que resulta muy útil para el diseño y desarrollo de proyectos electrónicos.

Definiciones del estándar USB

Se dará un vistazo general de algunos términos que son utilizados en la comunicación USB. La mayoría de las definiciones de los términos que se utilizan en el tratamiento del USB, han sido compiladas de los sitios web:

- http://www.aquihayapuntes.com/
- http://picmania.garcia-cuervo.net/
- http://www.sase.com.ar/

HOST

Dispositivo maestro o anfitrión que inicia la comunicación, por ejemplo una computadora.

INTERFAZ FÍSICA

El conector USB está formado por cuatro hilos. Dos para la alimentación 5V (Rojo) y 0V (Negro). Dos conductores para datos D+ (verde) y D- (blanco). Los cables de datos son del tipo trenzado y tienen una impedancia de 90 Ω ± 15%, conducen señales diferenciales de voltaje, el mismo que depende de la velocidad del bus, 3.3v para versiones USB 1.0 y 1.1 y 0.4V para USB 2.0 y 3.0. La corriente de consumo por puerto es 100 mA.

TIPOS DE TRANSFERENCIA

La especificación USB permite cuatro tipos de transferencias: control, bulk, isocrónicas, interrupción.

Control. Configuran y envían comandos. Se usa en la enumeración del dispositivo y para el control de los dispositivos conectados al bus.

Bulk (masivas). Se utilizan para transferencia de datos grandes, es el tipo más rápido de transferencia. Los discos duros, pen drivers, escáneres, impresoras, etc., hacen uso de transferencias tipo bulk. Este método asegura la integridad de datos, pero no la temporización de envíos.

Isócronas: (de igual tiempo). Los dispositivos deben soportar velocidades Full Speed. Provee un método para transferir grandes cantidades de datos,

hasta 1023 bytes, con una temporización de envío asegurada, aunque la integridad de los datos no se asegura. Utilizado en aplicaciones de transmisión continúa (streaming) y donde pequeñas pérdidas de datos no son críticas. Es usada en dispositivos que transmiten señales de audio y de vídeo en tiempo real.

Interrupción: Se utiliza en dispositivos que no requieran mucho ancho de banda como: Teclados, Mouse, Sensores, Pantallas táctiles, etc. Este método asegura la temporización y la integridad de los datos para pequeños bloques de datos.

VELOCIDADES DEL BUS

De acuerdo a la velocidad, hasta la actualidad se han construido cuatro tipos de USB.

Low speed: 1,5 Mbps. Son utilizados por dispositivos como teclados, ratones, etc. que soportan las especificaciones 1.1, 2.0 y 3.0.

Full speed: 12 Mbps. Soportado por USB 1.1, USB 2.0 y USB 3.0. Usados para transmisiones de audio.

High speed: 480 Mbps. Solo USB 2.0 y USB 3.0, para transmisiones de video.

Super speed: 5Gbps. Solo dispositivos USB 3.0.

ENUMERACIÓN DEL DISPOSITIVO

El equipo Host detecta cualquier dispositivo que se conecta al bus. El Host necesita obtener la información sobre el dispositivo. El proceso que realiza el Host en obtener la información se denomina ENUMERACIÓN. Esta información que necesita el Host se encuentra definida en el dispositivo en los llamados descriptores. Los descriptores son datos que se guardan en la memoria no volátil del PIC y contienen la siguiente información: El ID del vendedor (VID) y del producto (PID), consumo de corriente del dispositivo, tipo de transferencia que se va a utilizar, endpoint utilizados, versión USB soportada, clase utilizada, etc.

VID & PID

El **VID** es un número de 16 bits que significa Vendor Identification o código que identifica al fabricante del hardware a conectar. El número 04D8h identifica a Microchip.

El **PID** es un número de 16 bits que significa Product Identification o código que identifica al dispositivo en concreto hardware a conectar. El PIC18 tiene el número de identificación 000Bh.

ENDPOINTS

Puntos terminales. El ENDPOINT es un buffer de memoria RAM que almacena múltiples bytes, para envió y recepción de datos o comandos. Cada ENPOINT son bidireccionales. El proceso de enumeración se realiza a través del ENDPOINT 0.

DRIVER

Es un programa que habilita aplicaciones para poderse comunicar con el dispositivo. Cada dispositivo sobre el bus debe tener un driver, algunos periféricos utilizan los drivers que trae Windows.

PIPE, TUBERÍA

Es una conexión lógica (virtual) entre un ENDPOINT y el software del controlador del host que se produce tras el proceso de enumeración.

CLASE DE COMUNICACIÓN USB

Existen cuatro clases de comunicación USB. Estas son:

Bulk transfers USB. Presenta una transferencia bidireccional masiva de información.

HID (Human Interface Device). Dispositivos de interfaz humana.

CDC (Comunications Device Class). Clase de dispositivo de comunicación. Emula a la comunicación serial RS232.

MSD (Mass Storage Device Class). Dispositivos para almacenamiento masivo.

Funciones relevantes USB del CCS

CCS provee de una gran cantidad de librerías para el manejo de los módulos USB para los microcontroladores que poseen módulos USB como el PIC16F65 o la familia del PIC18FXXX, como el PIC18F4550. Se indica las funciones más importantes basadas en la descripción que ofrece el manual de CCS.

usb_init(). Inicializa el hardware del USB. Espera mientras el periférico USB se conecta al bus, pero eso no significa que se enumera por el PC. Habilitará y usará la interrupción del USB.

usb_init_cs(). Igual a ***usb_init()***, pero no espera a que el dispositivo se conecte al bus. Esto es útil si el dispositivo no está alimentado por bus y puede funcionar sin una conexión USB.

usb_task(). Censa la conexión en el dispositivo y el bus. Cuando el PIC está conectado al BUS, esta función prepara el periférico USB. Cuando el PIC está desconectado del bus, se restablecerá la pila del USB y el periférico. Habilita y deshabilita la interrupción USB. En una aplicación debe definir USB_CON_SENSE_PIN al pin sensor de conexión.

usb_detach(). Remueve al PIC del bus. Se llamará automáticamente por ***usb_task ()*** si se pierde la conexión, pero se puede llamar de forma manual por el usuario.

usb_attach (). Conecta el PIC al bus. Se llamará automáticamente por u***sb_task ()*** si se realiza la conexión, pero se puede llamar de forma manual por el usuario.

usb_attached (). Si se utiliza la conexión de clavijas sentido (USB_CON_SENSE_PIN), devuelve TRUE si ese pin es alto. Else siempre devolverá true.

usb_enumerated (). Devuelve TRUE si el dispositivo ha sido enumerado por la PC. Si el dispositivo ha sido enumerado por la PC, eso significa que está en modo de operación normal y se puede enviar / recibir paquetes.

usb_put_packet (punto final, los datos, len, TGL). Coloca el paquete de datos en el búfer de punto final especificado. Devuelve TRUE si el éxito, FALSE si el buffer está todavía llena con el último paquete.

usb_puts (endpoint, los datos, len, tiempo de espera). Envía los siguientes datos para el endpoint especificado. ***usb_puts ()*** difiere de usb_put_packet () en que va a enviar mensajes en paquetes múltiples, si los datos no caben en un solo paquete.

usb_kbhit (enpoint). Devuelve TRUE si el punto final especificado tiene datos en el buffer de recepción.

usb_get_packet (endpoint, ptr, max). Lee hasta max bytes especificado en el buffer del endpoint y lo guarda en el ptr puntero. Devuelve el número de bytes guardados en puntero ptr.

usb_gets (endpoint, ptr, max, time out). Lee un mensaje desde el punto final especificado. La diferencia entre ***usb_get_packet()*** y ***usb_gets()*** es que ***usb_gets()*** esperará hasta que un mensaje completo ha recibido, que un mensaje puede contener más de un paquete. Devuelve el número de bytes recibidos.

#int_usb. **Interrupción USB.** Un evento de USB ha sucedido y requiere la intervención de la aplicación. La biblioteca USB que proporciona CCS manipula esta interrupción automática.

Archivos más relevantes

pic_usb.h. Controlador a nivel de hardware para el PICmicro PIC16C765 y los microcontroladores de la familia con un periférico USB interno.

pic18_usb.h. Controlador a nivel de hardware para el PIC18F4550 y la familia de los microcontroladores con periférico USB interno.

usb.c. La pila USB, que maneja la interrupción USB y solicitudes de configuración USB del endpoint 0.

Comunicación USB PIC18F4550 – LABVIEW

LabVIEW es un lenguaje de programación de alto nivel, enfocado a la Instrumentación, de tipo gráfico, es decir, a diferencia de los lenguajes tradicionales de alto nivel como FORTRAN, C, PASCAL u otros utiliza iconos gráficos para la confección del programa. Como lenguaje de programación maneja estructuras, por lo cual puede ser utilizado para elaborar algoritmos informáticos, para análisis y procesamiento de la información. Cada programa realizado se denomina un VI.

Es de suponer que el lector tiene conocimientos en el manejo y uso de LabVIEW, por lo que se enfocará directamente a realizar la aplicación de la comunicación USB entre el PIC y LabVIEW.

CONFIGURACIÓN DEL RELOJ DEL SISTEMA

El módulo USB 2 trabaja a 48 MHz, para conseguir esta frecuencia el microcontrolador dispone de un pre-escalar y un post-escalar como se observa en el circuito de la figura 3.9.

Por ejemplo, si el oscilador externo es de 12 MHz, el pre-escalar divide para 3 para tener 4 MHz a la entrada del multiplicador PLL, el cual eleva la frecuencia a 96 MHz. Mediante un divisor de frecuencia para 2, se logra tener los 48 MHz para el USB. Por otro lado mediante el divisor se configura la frecuencia del reloj del CPU. Para este caso tenemos 32 MHz. Nunca la frecuencia del CPU deberá ser mayor a 48 MHz, que es la máxima frecuencia de trabajo para el microcontrolador PIC18F4550.

En la figura 3.9, se indica una parte de la estructura del reloj interno del microcontrolador. Las líneas de color rojo muestran el camino para generar los 48 MHz para el módulo USB y los 32 MHz para el CPU del PIC.

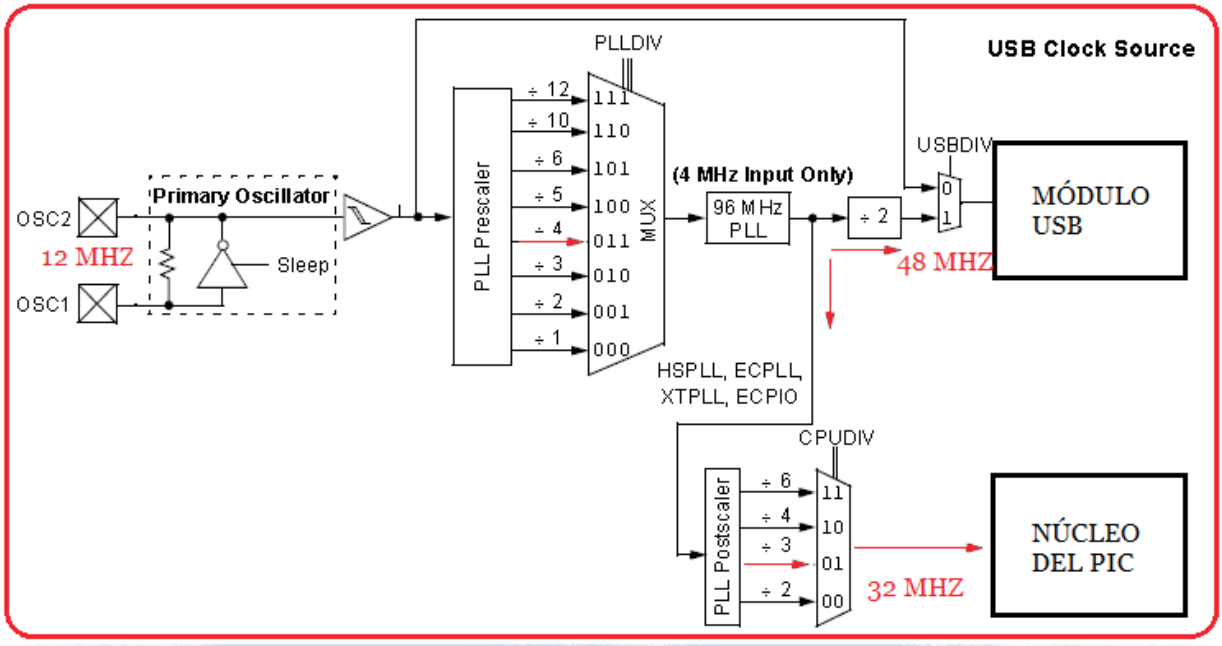


Figura 3.9. Parte de la estructura interna del sistema de reloj del microcontrolador.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Un extracto con las opciones para la configuración del oscilador del USB que el fabricante proporciona se indica en la tabla 3.3.

Input Oscillator Frequency	PLL Division (PLLDIV2:PLLDIV0)	Clock Mode (FOSC3:FOSC0)	MCU Clock Division (CPUDIV1:CPUDIV0)	Microcontroller Clock Frequency
20 MHz	÷5 (100)	HS, EC, ECIO	None (00)	20 MHz
			÷2 (01)	10 MHz
			÷3 (10)	6.67 MHz
			÷4 (11)	5 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
16 MHz	÷4 (011)	HS, EC, ECIO	None (00)	16 MHz
			÷2 (01)	8 MHz
			÷3 (10)	5.33 MHz
			÷4 (11)	4 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz
12 MHz	÷3 (010)	HS, EC, ECIO	None (00)	12 MHz
			÷2 (01)	6 MHz
			÷3 (10)	4 MHz
			÷4 (11)	3 MHz
		HSPLL, ECPLL, ECPIO	÷2 (00)	48 MHz
			÷3 (01)	32 MHz
			÷4 (10)	**24 MHz**
			÷6 (11)	16 MHz

Tabla 3.3. Posibles opciones para configuración del reloj del USB. Extracto tomado del datasheet del PIC184550.
Fuente: PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.

Las opciones para el núcleo del microcontrolador sería 48, 32, 24 y 16 MHZ. Tomaremos una frecuencia de 32 MHz. Mediante los FUSES en CCS se puede configurar la frecuencia requerida. Para el ejemplo la configuración de los fuses para el reloj sería:

#fusesHSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV, PLL3, CPUDIV3,VREGEN, NOMCLR //Configuración de fusibles.

Dónde:

HSPLL, activa el post-escalar para la frecuencia del CPU del micro.

PLL, divisor del reloj del sistema, si el oscilador externo es 12MHz, PLL3, divide para 3, consiguiendo 4MHz.

USBDIV, módulo USB utiliza el reloj directo del sistema *USBDIV=0,* o la frecuencia del PLL *USBDIV=1.* En CCS *USBDIV* asume el valor de 1 y *NOUSBDIV* toma el valor de 0.

CPUDIV, configura la frecuencia para el CPU del Micro, CPUDIV3, obtiene 32 MHz.

VREGEN, activa el regulador interno del USB de 3.3 V.

Instalación del virtual USB DRIVER DE ISIS

PROTEUS es el único programa del mundo que provee de una interfaz gráfica del módulo virtual para USB. A partir de la versión 7 viene incorporado la aplicación VIRTUAL USB DRIVER, que debemos instalar y nos permitirá utilizar el dispositivo en forma virtual.

El VIRTUAL USB DRIVER viene en el grupo de programas de PROTEUS. Si ya está instalado ISIS, basta con ejecutar VIRTUAL USB en el menú de inicio de WINDOWS, como indica la figura 3.10, para encontrar el instalador.

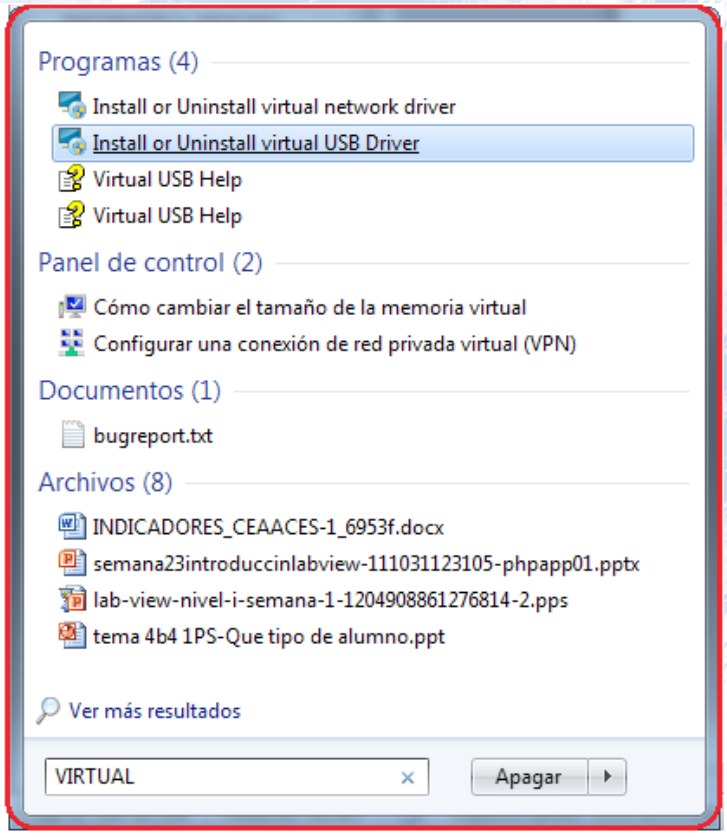


Figura 3.10. Ubicación del instalador del Virtual USB Driver de ISIS.

El Wizard del instalador, nos permite siguiendo las pantallas gráficas, instalar fácilmente el Virtual USB Drivers. A continuación se muestran las capturas de las principales pantallas de la instalación del Virtual USB Drivers.

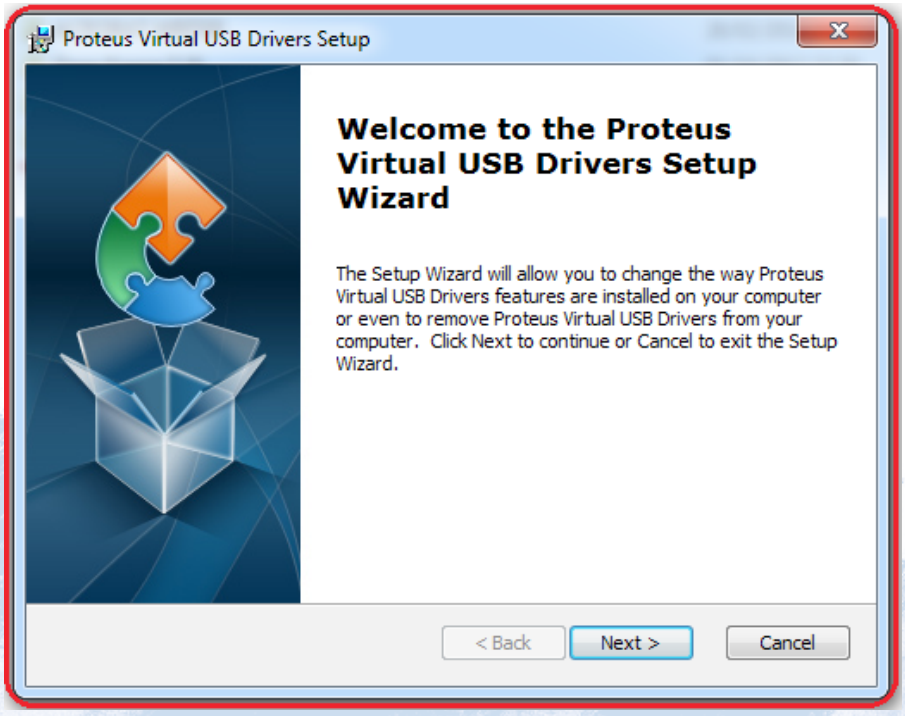


Figura 3.11. Pantalla inicial de instalación del Virtual USB Drivers.
Elaborado por: Los Autores.

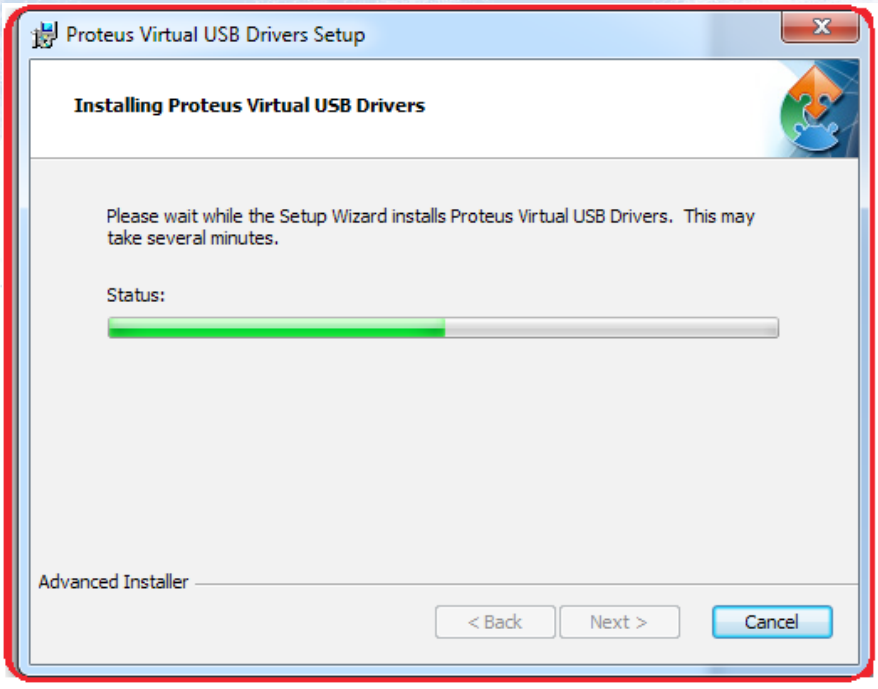


Figura 3.12. Pantalla del progreso de la instalación del Virtual USB Drivers.

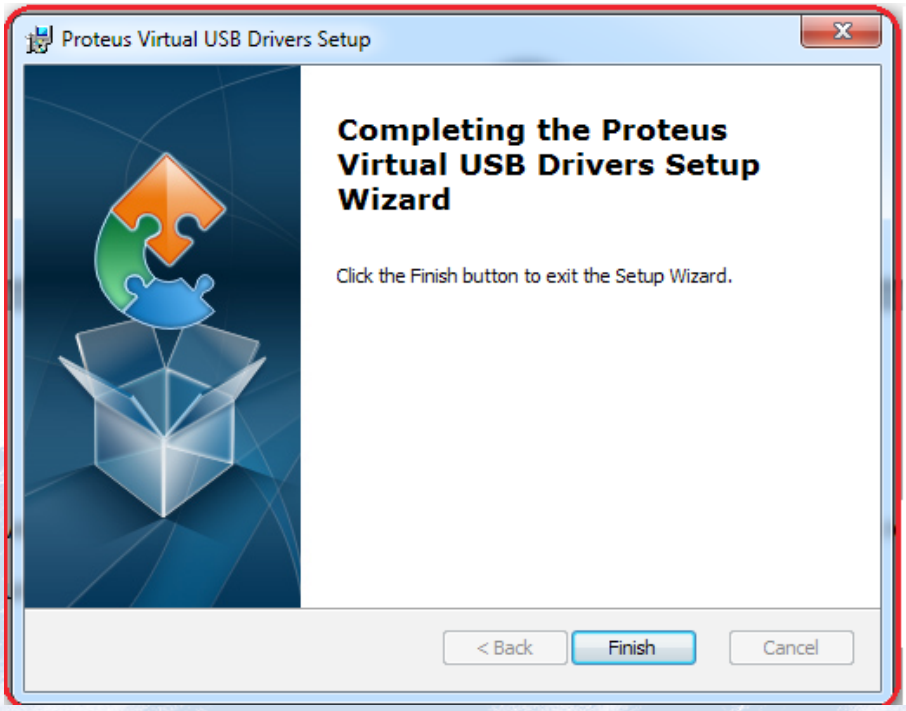


Figura 3.13. Pantalla final de la instalación del Virtual USB Drivers.
Elaborado por: Los Autores.

Una vez que se tiene instalado el driver del USB virtual, al iniciar la simulación en ISIS WINDOWS reconocerá el driver como si se conectará un dispositivo real.

Circuito en ISIS

La aplicación lee los datos de una señal analógica que está conectada al canal AN0 del PIC18F4550. A través de un potenciómetro se ingresa una señal CC de 0 a 5V. Este dato se envía al PC mediante comunicación USB. En LabVIEW se indicará el valor de la lectura del ADC, una variable int8, o sea los valores estarán de 0 a 255 y mediante un indicador numérico se muestra el equivalente al voltaje de la señal de entrada (0 a 5V). Desde LabVIEW se va a controlar los LEDS que están conectados en el puerto B del microcontrolador. Se dispone de dos LEDS de señalización en ISIS para indicar el arranque y la conexión del USB. El circuito utilizado para la simulación en ISIS e indica en la figura 3.14.

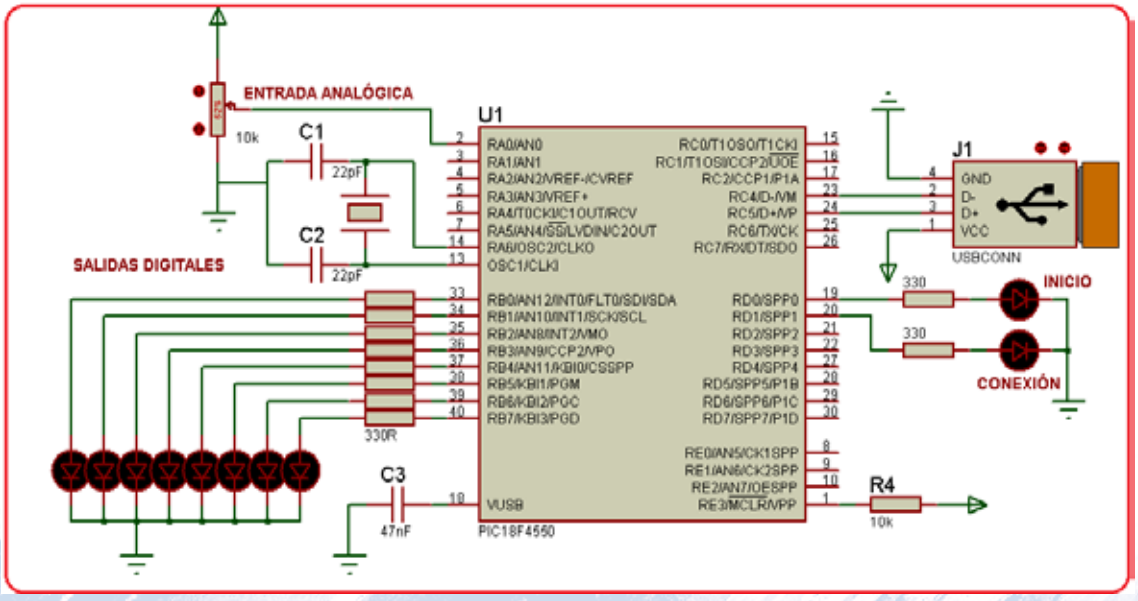


Figura 3.14. Circuito en ISIS para la simulación de la comunicación USB.
Elaborado por: Los Autores.

CODIGO PARA EL MICROCONTROLADOR

El código que se desarrolló para el microcontrolador se indica en su totalidad en el programa siguiente. Las instrucciones utilizadas están comentadas claramente indicando la función de las mismas.

PROGRAMA:

```
#include <18F4550.h>      /                  /Librería para usar el PIC18F4550.
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL3,CPUDIV3,
 VREGEN, NOMCLR          /                  /Configuración de fusibles.
#use delay(clock=12M)     /                  / F_OSC = 12 MHz.
#byte port_b = 0xF81      /                  /Identificador del Puerto B.
#define USB_HID_DEVICE  TRUE          //Tipo de comunicación USB HID verdadera.
                  //Cambia a on EP1 para la transferencia de entrada IN bulk/interrupt.
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE    8
                  //Cambia a on EP1 para la transferencia de salida OUT bulk/interrupt.
#define USB_EP1_RX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE    8
#include <pic18_usb.h>    /          /Funciones de bajo nivel (hardware) que servirán en usb.c
#include <my_usb_desc_hid.h> /   /Descriptores del dispositivo.
#include <usb.c>                   //Librería para el manejo del USB.
#define LED1 PIN_D0                //LED1para la espera de la conexión USB
#define LED2 PIN_D1                //LED2, se enciende cuando el USB está conectado.
int8 Salida[8];                     //Variable de salida para envió de la información entrada analógica.
int8 Entrada[8];                    //Variable para recibir el estado de los interruptores en LABVIEW.
```

```
void main() {       /                   /Función principal.
  set_tris_b(0x00);         /           /Define el Puerto B como salida.
  output_b(0x00);           /           /Define el Puerto B en 0.
  setup_adc_ports(AN0);
  setup_adc(ADC_CLOCK_INTERNAL);  /  /Fija el Clock interno para el ADC.
  set_adc_channel(0);       /           /Fija el canal 0 para el ADC.

  output_high(LED1);        /           /Activa LED1 indicando que se inició la conexión.
  output_low(LED2);         /           /LED2 continúa apagado.
  usb_init();               /           /Inicializa el LCD.
  usb_task();      //Monitorea el estado de la conexión conectándose y desconectándose automáticamente.
  usb_wait_for_enumeration();           //Espera infinitamente hasta que el dispositivo sea.enumerado.
  output_high(LED2);        /           /Apaga LED2 de espera.
  output_low(LED1);                 //Indica que el dispositivo se enumeró y se estableció la comunicación.
  while (TRUE){
    usb_task();    //Monitorea el estado de la conexión conectándose y desconectándose automáticamente.
    if (usb_enumerated()){              //Si el dispositivo está enumerado….?
     Salida[0]=read_adc();              //Lee el valor del ADC.
    /                                   /Envía al bus USB el paquete de datos.
     usb_put_packet(1, Salida, 1, USB_DTS_TOGGLE);
    if (usb_kbhit(1)){          /       /Si hay datos en el buffer del USB….?
      usb_get_packet(1, Entrada, 1); /  /Recibe el dato la variable entrada.
      port_b= Entrada[0];               //Asigna al Puerto B el byte de Entrada.
      }
    }
  }             /                       /Fin del lazo infinito.
}               /                       /Fin del main.
```

Descriptores del dispositivo

Para el ejercicio se utiliza la comunicación HID, debido a que CCS posee las librerías necesarias para realizar esta comunicación y LabVIEW dispone del NI VISA WIZARD para generar el controlador del USB para comunicación con ISIS y el microcontrolador.

Para los descriptores del dispositivo HID utilizaremos la librería ***usb_desc_hid.h.*** Dando un vistazo a este archivo se observa que utiliza los VID 0x0461 y PID 0x0020. Por tanto, debemos modificar este código con los descriptores que utiliza el VIRTUAL USB de ISIS, como lo se hará más adelante. La librería ***usb_desc_hid.h,*** está en la carpeta de la instalación de CCS en el directorio DRIVERS.

```
46
47  #ifndef USB_CONFIG_PID
48      #define USB_CONFIG_PID  0x0020
49  #endif
50
51  #ifndef USB_CONFIG_VID
52      #define  USB_CONFIG_VID 0x0461
53  #endif
```

Figura 3.15. Descriptores usados en CCS.
Elaborado por: Los Autores.

En el administrador de dispositivos de WINDOWS, se puede verificar los dispositivos HID.

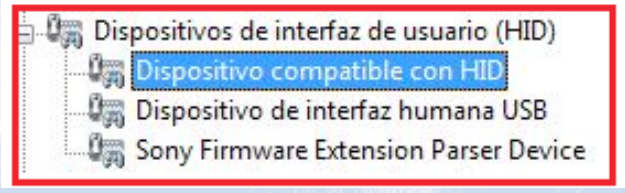


Figura 3.16. Lista de dispositivos HID
Elaborado por: Los Autores.

Luego se puede verificar el VID y PID asignado por WINDOWS al dispositivo HID. Haga doble clic en dispositivo compatible con HID y la pestaña detalles busque el ID del hardware del dispositivo.

Figura 3.17. VID y PID del dispositivo Virtual USB Drivers.
Elaborado por: Los Autores.

Con estos datos se procede a modificar los descriptores de la librería ***usb_desc_hid.h.*** En la presente aplicación se modificó el VID = 0x0000 y PID = 0x0000 y se renombró a la librería por ***my_usb_desc_hid.h,*** para no alterar la original.

```
#ifndef USB_CONFIG_PID
 #define USB_CONFIG_PID  0x0000
#endif

#ifndef USB_CONFIG_VID
 #define  USB_CONFIG_VID 0x0000
```

Figura 3.18. Descriptores utilizados para la enumeración del dispositivo.
Elaborado por: Los Autores.

Driver de control e instrumento virtual en LabVIEW

Para que el dispositivo USB pueda ser controlado desde LabVIEW es necesario que este lo reconozca como propio, con un driver USB desarrollado en LabVIEW. Para ello LabVIEW brinda una utilidad llamada "NI VISA Driver Wizard". Esta da la posibilidad de desarrollar un driver para dispositivos PCI y USB. NI VISA se puede descargar gratuitamente desde el link ***http://www.ni.com/download/ni-visa-5.2/3337/en/***

Es necesario verificar la compatibilidad para los sistemas operativos y la versión de LabVIEW. En nuestro caso se desarrolló en LabVIEW 2012 y NI VISA 5.41. Una vez instalado el NI VISA, se sigue los siguientes pasos para la generación del DRIVER.

Al iniciar el NI-VISA Wizard, se debe seleccionar el tipo de bus que en nuestro caso es USB.

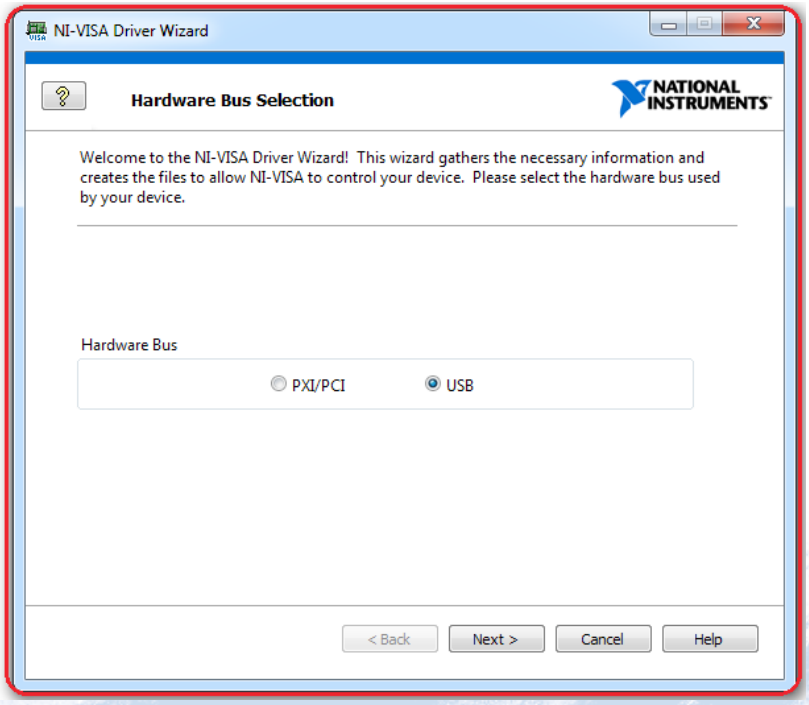


Figura 3.19. Selección del tipo de bus para la comunicación.
Elaborado por: Los Autores.

En la siguiente ventana se observa la lista de dispositivos disponibles y debe estar el que utiliza ISIS. Si no aparece haga clic en el botón refresh. En nuestro caso está disponible el USB\VID_0&PID_0.

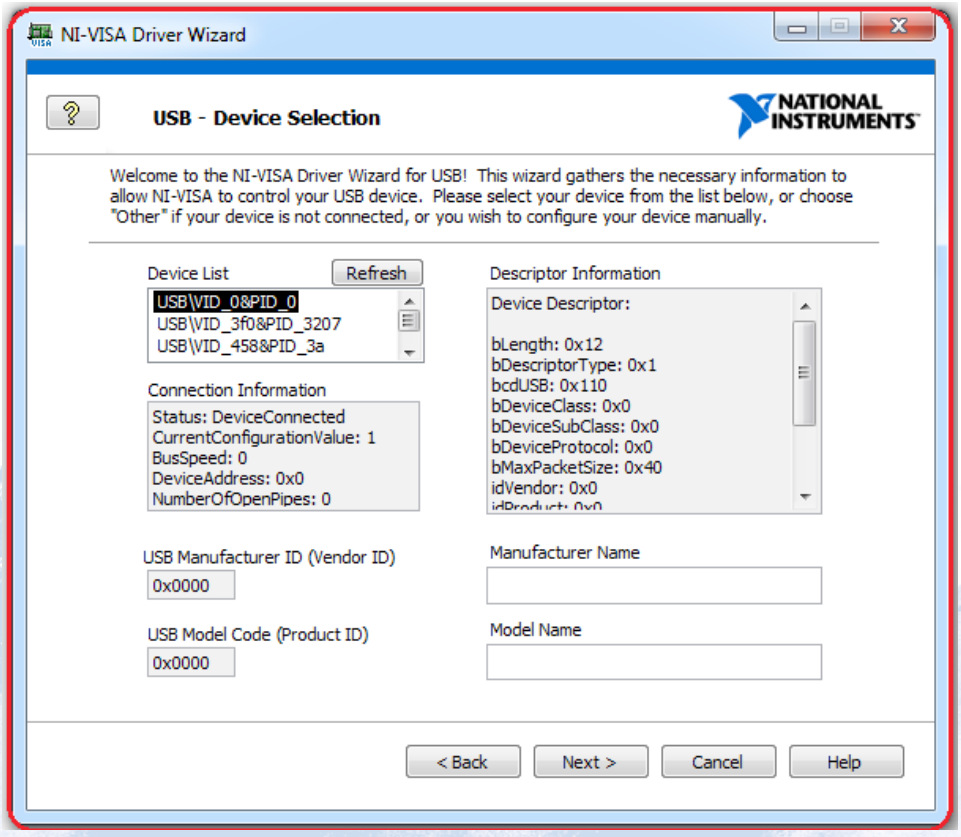


Figura 3.20. Selección de dispositivos USB.
Elaborado por: Los Autores.

Se selecciona la opción ***other.***

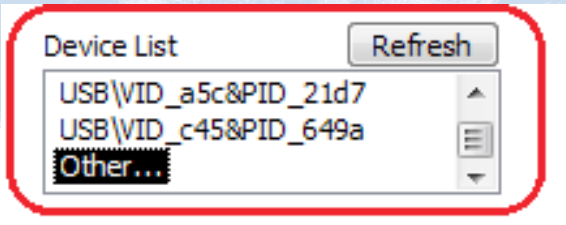


Figura 3.21. Lista de dispositivos disponibles.
Elaborado por: Los Autores.

En esta caja de texto se procede a completar los datos solicitados, cuidando indicar el VID y PID correcto, es decir los que están utilizando el driver de ISIS y el programa del microcontrolador elaborado en CCS. Los campos de manufactura y el modelo del dispositivo se designan al azar, para el ejemplo, estos campos se han designado con NI-VISA y NI VISA.

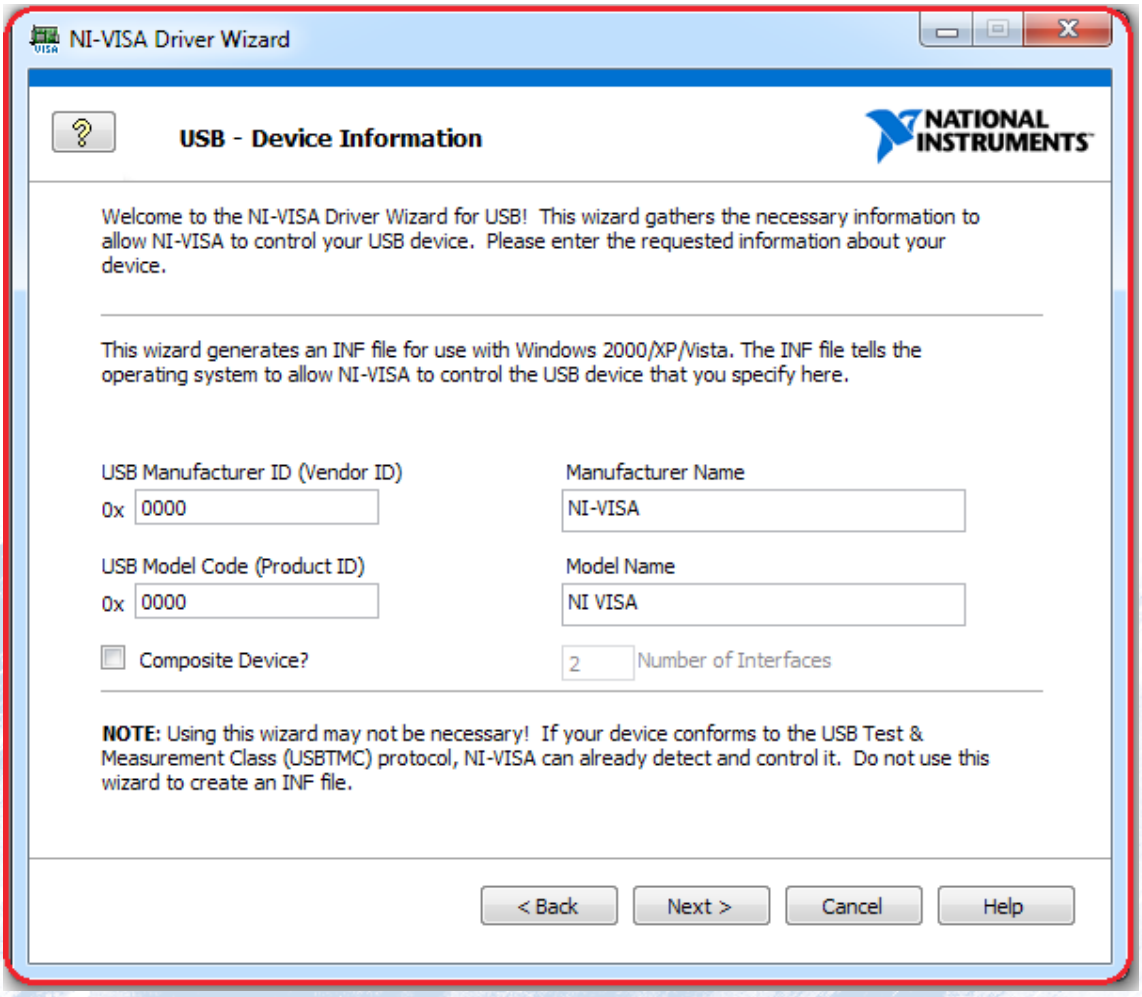


Figura 3.22. Información el dispositivo USB.
Elaborado por: Los Autores.

A continuación se nombre del archivo ***Inf,*** y se guarda en una carpeta de acuerdo a las necesidades del usuario o en mismo directorio del NI VISA.

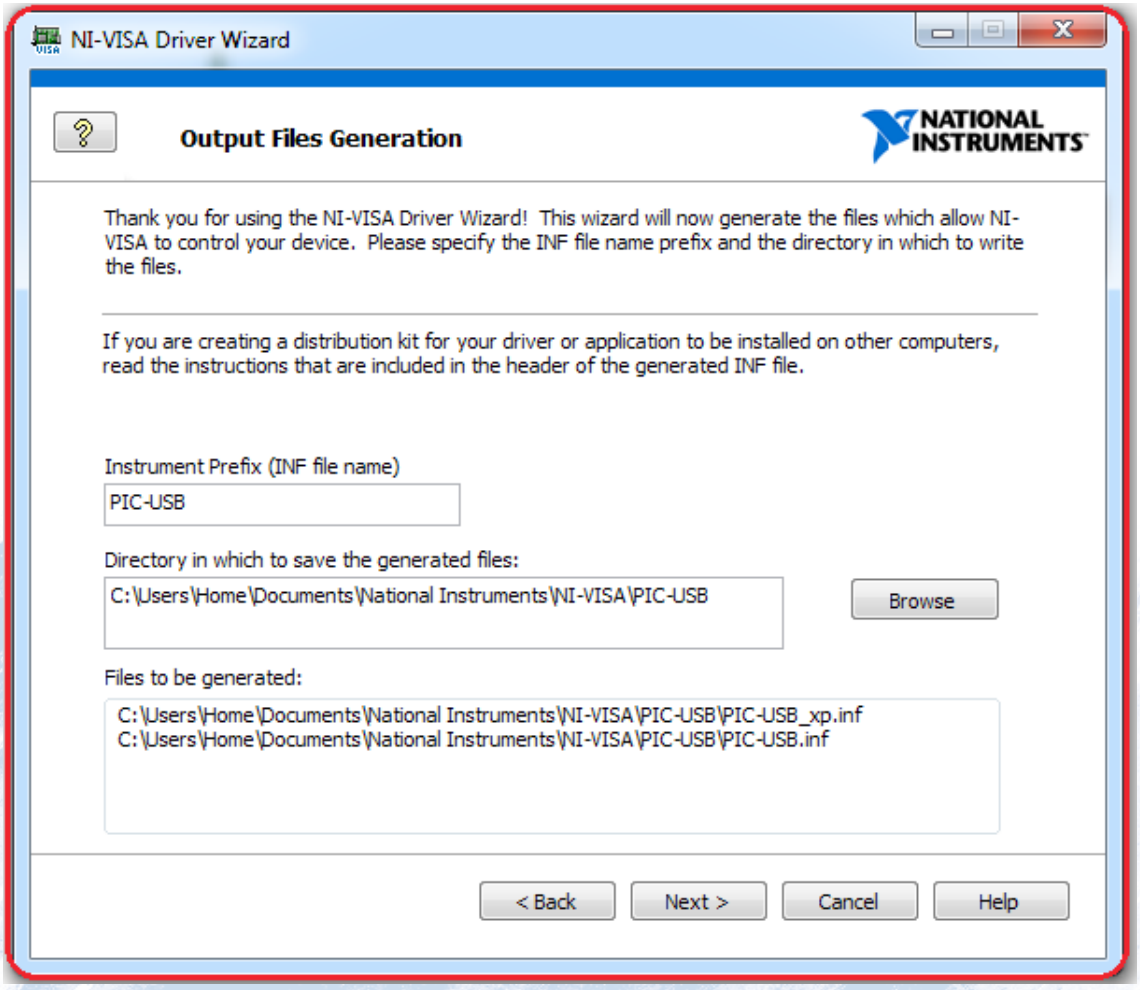


Figura 3.23. Nombre el archivo INF y directorio a guardar.
Elaborado por: Los Autores.

Finalmente se opta por la primera opción, es decir, instalamos el driver inf, quedando listo para utilizar.

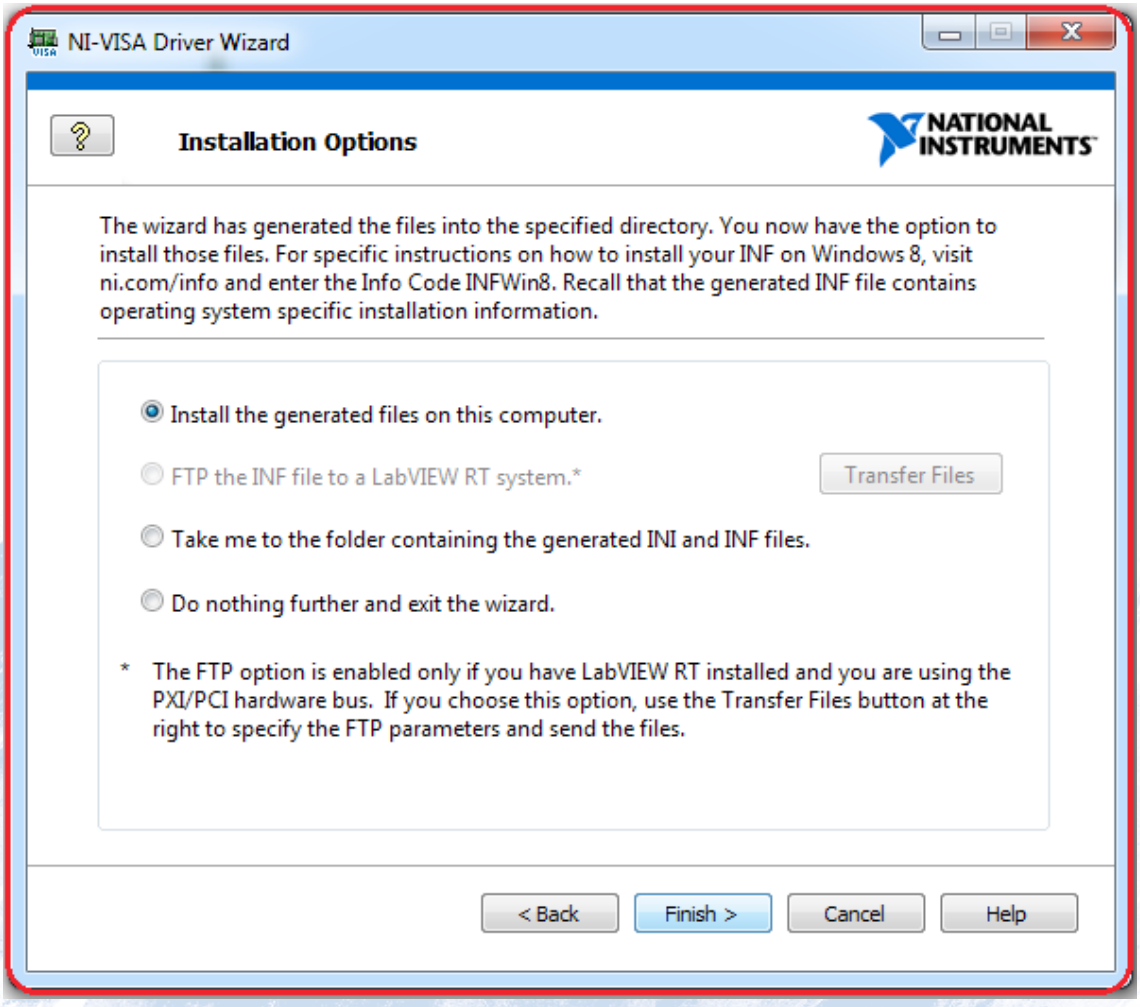


Figura 3.24. Pantalla final de instalación y generación del archivo INF del driver NI VISA.
Elaborado por: Los Autores.

Finalmente en el administrador de dispositivos de WINDOWS se puede verificar que se crea el NI VISA USB.

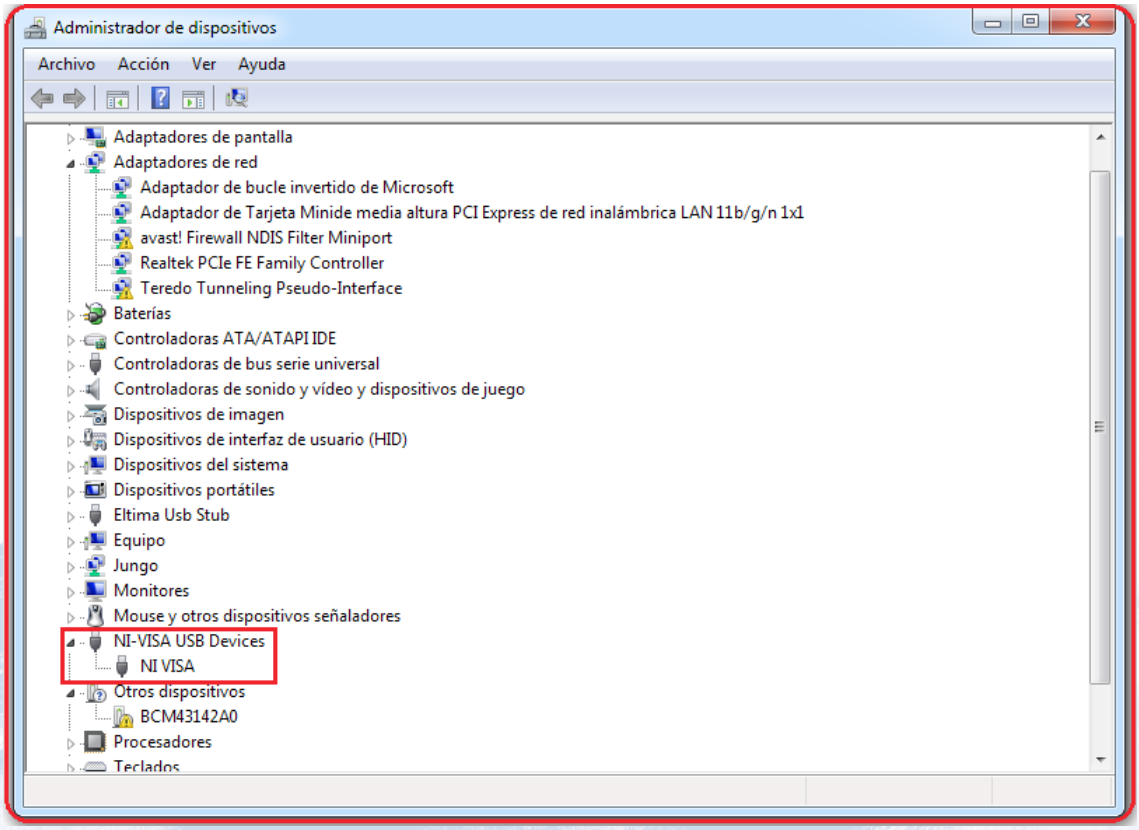


Figura 3.25. Administrador de dispositivos indicando el controlador NI VISA.
Elaborado por: Los Autores.

PROGRAMA EN LABVIEW

PANEL FRONTAL

Ocho interruptores con sus indicadores se utilizan para activar o desactivar el puerto B. Mediante el *VISA resource name* se detecta el driver generado por el NI VISA que sirve para la comunicación con ISIS o con el dispositivo físico.

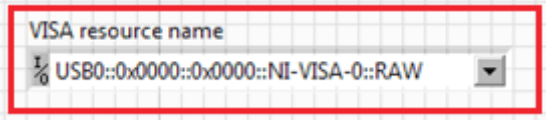


Figura 3.26. VISA RESOURSE indicando el controlador creado con NI VISA.
Elaborado por: Los Autores.

Dos indicadores numéricos muestran la lectura directa del conversor ADC y su equivalente en voltios.

La figura 3.27, indica la interfaz de usuario o panel frontal con todos los componentes utilizados en LabVIEW.

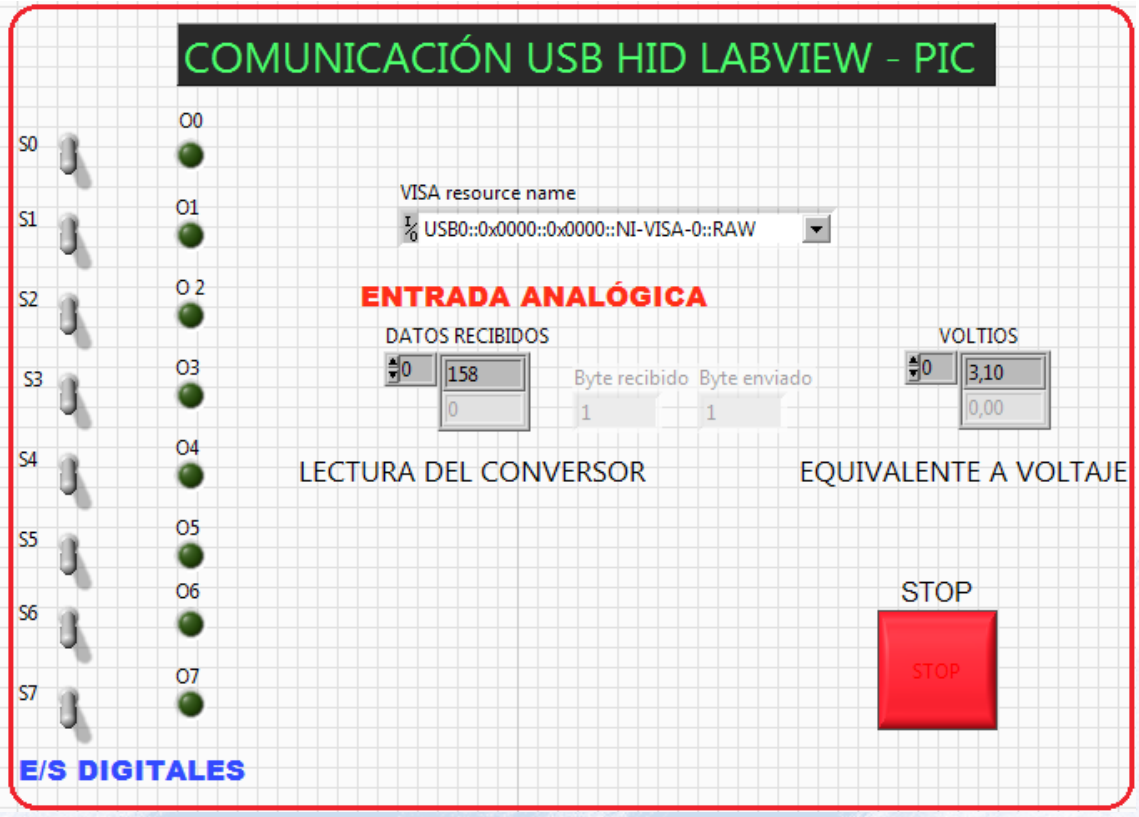


Figura 3.27. Panel frontal. Interfaz del usuario.
Elaborado por: Los Autores.

Diagrama de bloques

Para escribir en el bus USB se utiliza el VISA WRITER que trabaja con datos tipo string. Por tanto, cualquier dato numérico debemos convertir a string.

El proceso para convertir de un dato numérico individual a un string es el siguiente:

Agrupar los bits individuales de los pulsadores (BUNDLE).

Convertir el grupo de elementos del mismo tipo de datos en array 1D de elementos del mismo tipo de datos (CLUSTER TO ARRAY).

Convertir el array booleano en un entero o un número de punto fijo mediante la interpretación del array como la representación binaria del número (BOOLEAN ARRAY TO NUMBER).

Concatenar los arrays elementos en un array n-dimensional (BUILD ARRAY).

Convertir el byte array sin signo que representa los caracteres ASCII a una cadena (BYTE ARRAY TO STRING).

Concatenar las cadenas de entrada y matrices 1D en una única cadena de salida (CONCATENATE STRINGS).

La figura 3.28, indica las funciones utilizadas en el proceso de conversión.

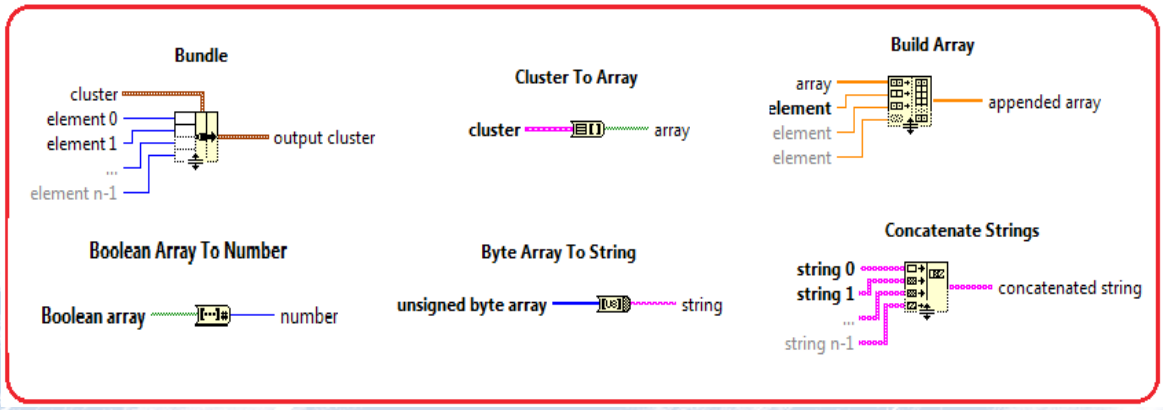


Figura 3.28. Funciones utilizadas en la conversión de bits individuales a un string.
Elaborado por: Los Autores.

La cadena de datos se ingresa al write buffer del VISA WRITE quedando el dato listo en el buffer del USB.

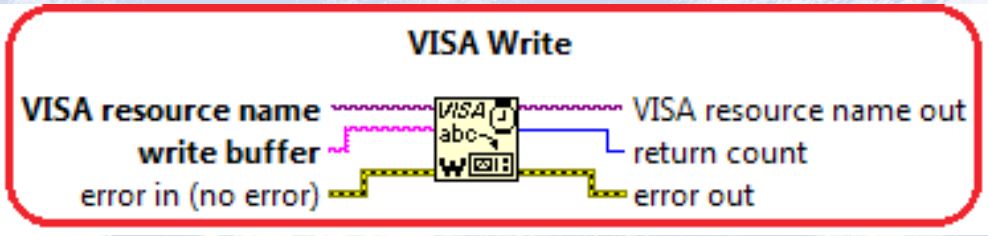


Figura 3.29. VISA Write.
Elaborado por: Los Autores.

El diagrama de bloques de la figura 3.30, detalla las funciones utilizadas en todo el proceso de lectura y escritura de los datos. El VISA resource name detecta el controlador NI-VISA, se abre el VISA y mediante eventos de interrupción del USB, se escribe o se lee los datos. Siempre se debe cerrar el VISA. Existe también un pequeño bloque de conversión matemática a voltaje y los indicadores numéricos respectivos.

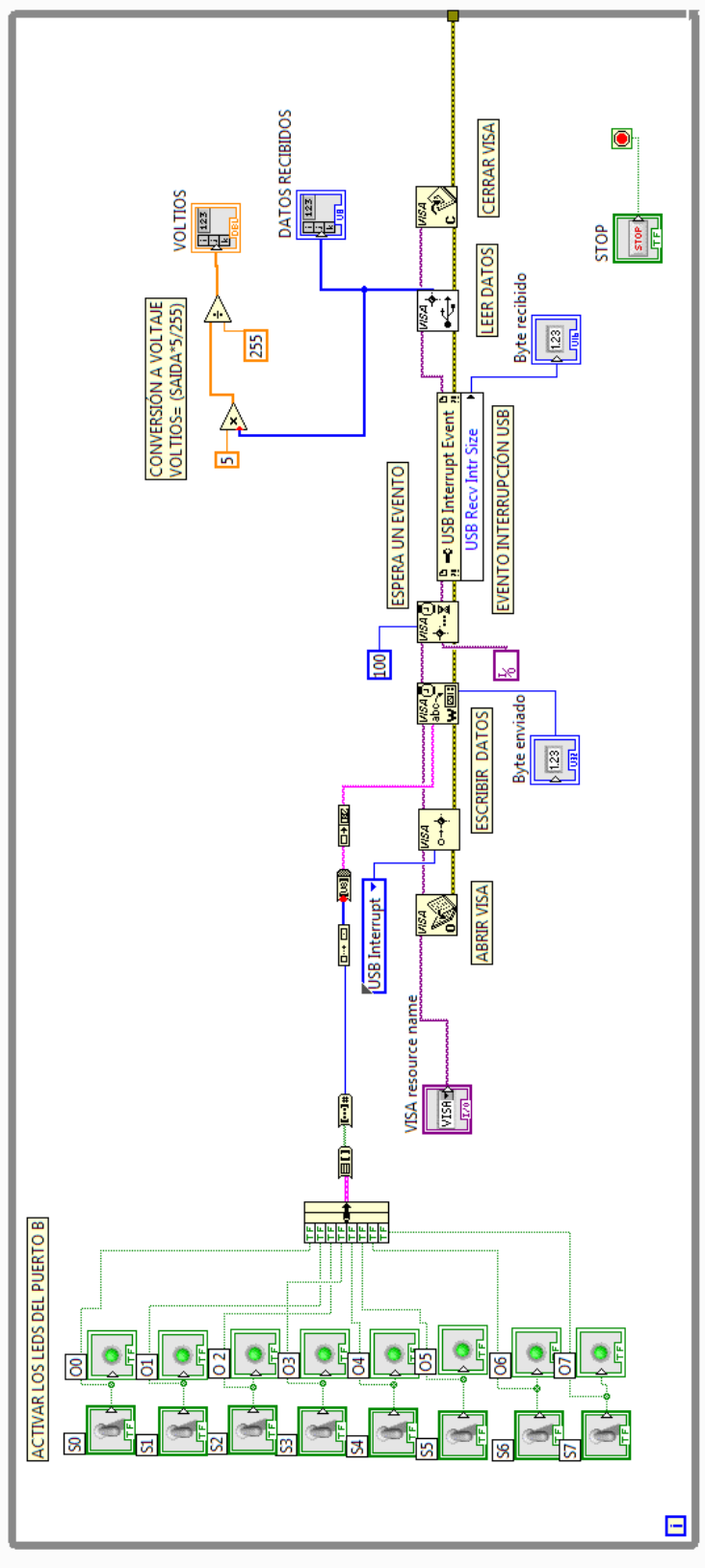
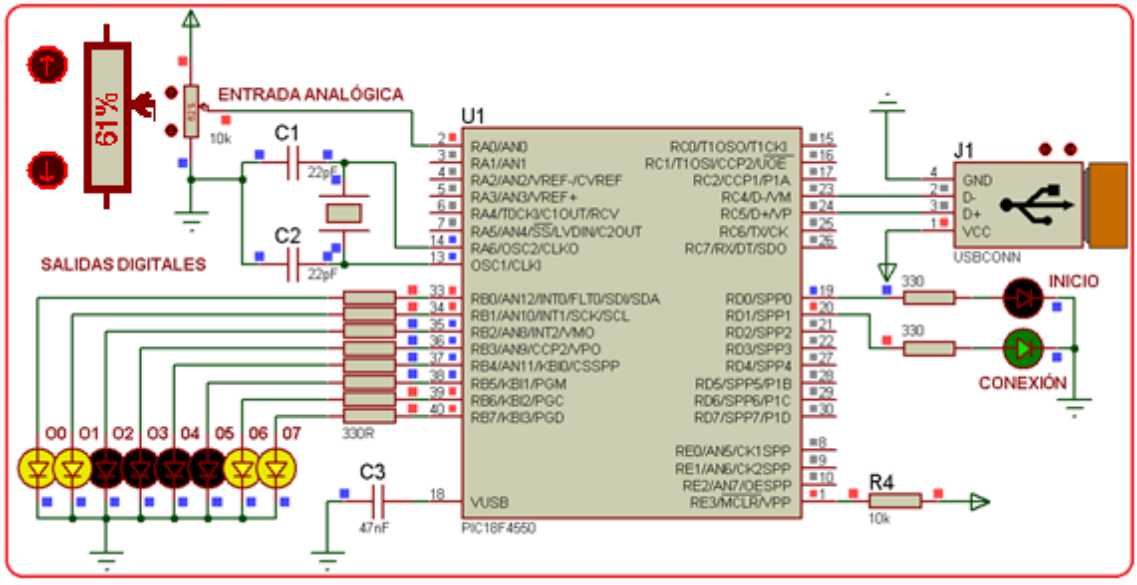


Figura 3.30. Diagrama de bloques.
Elaborado por: Los Autores.

a. Panel frontal.



b. Circuito en ISIS.

Figura 3.31. Captura de la simulación en ISIS con LabVIEW.
Elaborado por: Los Autores.

La figura 3.31, muestra la captura de la pantalla del panel frontal de LabVIEW. Se puede observar que los pulsadores S0, S1, S6 y S7, están activados y los indicadores numéricos de la entrada analógica indican 158 y 3.10 Voltios.

En ISIS, las salidas O0, O1, O6 y O7, están activados en respuesta a las señales de los pulsadores y el potenciómetro está en posición del 61% que corresponde a 158 de la lectura del ADC y su equivalente en voltaje de 3.1V.

Comunicación serial RS232 PIC18F4550 – LABVIEW

A pesar que en la mayoría de las computadoras modernas ya no existe el puerto serial, a nivel industrial el protocolo de comunicación RS232 sigue vigente, por lo que realizaremos una aplicación de comunicación utilizando este protocolo.

Puerto Serial DB9

La descripción de la función de pines del conector serial DB-9 usado para la comunicación serial se indica en la tabla 3.4.

PIN	FUNCIÓN	
1	DCD	Data Carrier Detect
2	RXD	Received Data
3	TXD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Tabla 3.4. Pines del puerto serial DB9.
Elaborado por: Los Autores.

De estos pines nos interesa el terminal 2 para la recepción de datos RX, el terminal 3 TX para la transmisión y el terminal 5 de tierra común.

El estándar RS232 de EIA (Electronics Industry Association) establece los parámetros de las especificaciones eléctricas del puerto serial, algunas de estas son:

1. Un “Espacio” (0 lógico) está entre +3 y +25 voltios.
2. Una “Marca” (1 lógico) está entre -3 y- 25 voltios.
3. La región entre +3 y -3 voltios es indefinida.
4. Un circuito abierto nunca debe exceder 25 Volts (con referencia a tierra).
5. La corriente de corto circuito no debe exceder los 500mA.
6. El estándar RS232C especifica una tasa de transferencia de máximo de 20,000 bps (bits por segundo).

En el caso de los PCs, las señales del puerto señal son +13V para el 0 lógico y -13V para recepción y transmisión de datos y a la inversa en las señales de control.

Debido a que el microcontrolador trabaja con niveles de voltaje TTL, para comunicarse con el puerto serial del computador debemos realizar un circuito de interface, para adaptar a los voltajes del puerto del PC y del microcontrolador. El circuito MAX232 es utilizado universalmente para esta conversión. La figura 3.32, indica la estructura básica de la interface.

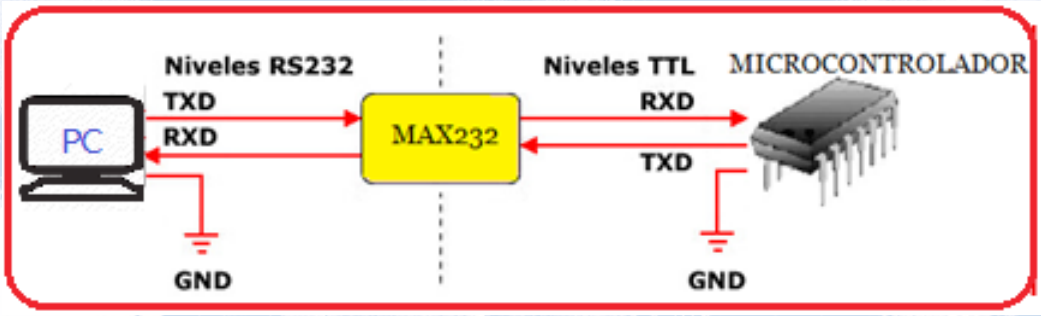


Figura 3.32. Conexión de PC con Microcontrolador con circuito MAX232.
Elaborado por: Los Autores.

Para la conexión física entre el microcontrolador y el puerto del PC si es necesario utilizar el circuito integrado MAX232.

EL CIRCUITO MAX232

El MAX232 es un circuito integrado fabricado por National Instrument dispone internamente de 4 conversores de niveles TTL al estándar RS232 y viceversa, para comunicación serie. La figura 3.33, muestra la distribución de pines del circuito MAX232.

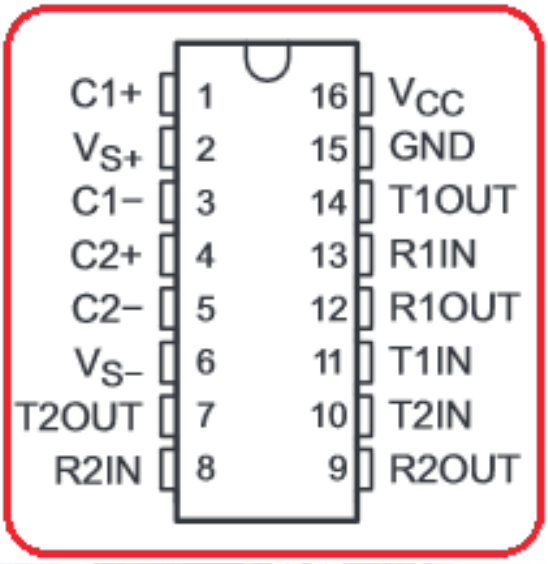


Figura 3.33. Circuito Integrado MAX232 de National Instrument.
Fuente: MAX232x Dual EIA-232 Drivers/Receivers. Texas Instrument.

Mediante la conexión externa de capacitores polarizados en al MAX232 se consigue la adaptación de voltajes a niveles TTL para el microcontrolador y en nivel RS232 para el puerto del PC. La figura muestra el diagrama de la interface entre el puerto del PC y el microcontrolador con un MAX232.

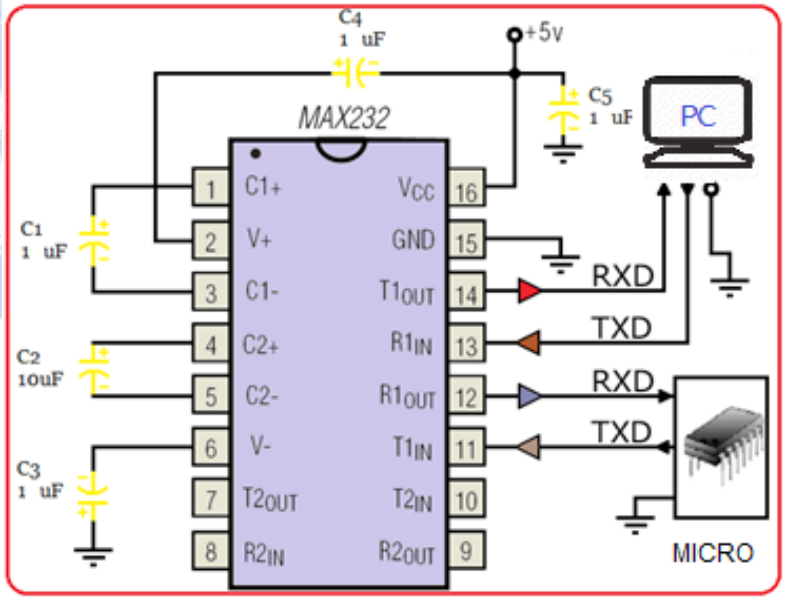


Figura 3.34. Conexiones del MAX232 para comunicación entre PC y PIC.
Elaborado por: Los Autores.

Aplicación. Termómetro y control de LEDS

Se realiza una comunicación dúplex entre el PIC y LabVIEW. La información analógica del sensor de temperatura LM35 será leída en el canal AN0 del Microcontrolador y mostrada en LabVIEW en un termómetro. Además desde LabVIEW se tiene ocho interruptores para controlar los LEDS que están conectados al puerto D del microcontrolador. Como el simulador Proteus posee el puerto serial virtual se efectuará la conexión entre LabVIEW y Proteus para ver los resultados inmediatamente.

PROGRAMA DEL MICROCONTROLADOR

Para el desarrollo del programa se tomó en cuenta las consideraciones:

- Uso del reloj interno INTRC_IO (8 MHz). Los demás fusibles son los mismos utilizados en los programas anteriores.
- Los parámetros para la transmisión se fijaron en 9600 bauds, el pin de recepción RC7, el pin de transmisión C6, 8 bits de datos y sin bit de paridad.
- Los datos se receptan y transmiten por interrupciones. Cuando hay un dato de entrada se produce la interrupción serial ***#int_rda*** y cuando hay una variación en el conversor AD se envía el dato mediante la activación de la interrupción ***#int_ad***. El orden de prioridad para la ejecución de las interrupciones se ha definido primero la serial y luego la del conversor: ***#priority int_rda,int_ad.***
- El dato del conversor se almacena en la variable ***Salida*** y es de tipo numérico (int16). El RS232 envía caracteres o datos tipo string por lo que necesitamos realizar una conversión de numérico a cadena. CCS tiene varias funciones para realizar esta conversión como son ***atoi, atof*** y ***sprintf***. Aunque la función ***sprintf*** ocupa más memoria RAM, es la más fácil de utilizar. Por esta razón analizamos los argumentos de esta función:

sprintf( )

Según el manual de CCS, esta función opera como el ***printf,*** excepto que la salida se coloca en la cadena especificada. La cadena de salida se dará por terminado con un nulo.

Sintaxis:

sprintf(string, cstring, values...);
num=sprintf(string, cstring, values...)

Donde, ***string*** es un array de caracteres. ***cstring*** es una cadena constante o un array de caracteres terminados en null. ***Values*** son una lista de variables separadas por comas.

Num retorna el número de bytes escrito en la cadena (string).

Por tanto, en el programa la instrucción ***sprintf(Texto,"%Lu",Salida);*** el valor numérico de la variable *Salida*, convierte en una cadena almacenando en la variable ***Texto***. El formato ***"%Lu",*** es el mismo utilizado en la función ***printf,*** es decir, el número es un entro largo sin signo.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550.
#device adc=10                          //Define el ADC en 10 bits.
#include <stdlib.h>                     //Incluye la librería stdlib.h.
                                        //Configuración de fusibles.
#fuses INTRC_IO,NOWDT,NOPROTECT,NOPUT, NOPBADEN, NOMCLR  /  /Definición de fusibles.
#use delay (clock=8000000)              //Define el OSC en 8 MHz.
                                        //Definición de parámetros del estándar RS232.
#use rs232(baud=9600,xmit=pin_c6,rcv=pin_c7,bits=8,parity=N)
#priority int_rda,int_ad                //Prioridad de ejecución de las interrupciones.
#use standard_io(b)                     //Usa librería estándar para el puerto B.
#use standard_io(d)                     //Usa librería estándar para el puerto D.
#byte port_b = 0xF81                    //Identificador del Puerto B.
#byte port_d = 0xF83                    //Identificador del Puerto D.

int16 Salida;                           //Variable para envió de datos.
int16 Entrada;                          //Variable para recibir datos.
char Texto[30];                         //Variable para conversión de dato numérico a string.

#int_rda                                //Interrupción serial de recepción de datos.
void rda_isr()
{
   Entrada= getc();                     //Asigna a Entrada el dato recibido.
}

#int_ad                                  //Interrupción del ADC.
void ad_interrupt(){
  sprintf(Texto,"%Lu",Salida);          //Convierte el dato numérico (Salida) a un String (Texto).
  printf("%s",Texto);                   //Envía el dato (Texto)
}
```

```
void main(void){                        //Función principal void.
  setup_adc_ports(AN0);                 //Activa canal AN0.
  setup_adc(ADC_CLOCK_INTERNAL);        //Usa CLOCK interno para el ADC.
  set_adc_channel(0);                   //Fija el canal 0 para el ADC.
  enable_interrupts(int_rda);           //Habilita la interrupción serial.
  enable_interrupts(int_ad);            //Habilita la interrupción del conversor AD.
  enable_interrupts(global);            //Habilita las interrupciones globales.

  set_tris_d(0x00);                     //Fija el puerto D como salida.
  output_high(pin_d0);                  //Activa pin D0.
  delay_ms(1000);                       //Retardo de 1000 ms.
  output_low(pin_d0);                   //Apaga pin D0.

 while( TRUE ){
   port_d= Entrada;                     //Asigne el dato de entrada al puerto D,
   Salida=read_adc();                   //Lee el valor de conversor AD y asigna a salida.
   delay_ms(100);                       //Retardo de 100 ms.
}
```

Circuito en PROTEUS

La figura 3.35, ilustra los componentes y conexiones utilizados para la comunicación entre LabVIEW y Proteus mediante puerto serial. Los datos del sensor de temperatura LM35 se ingresan al canal AN0 del microcontrolador. Mediante el puerto serial virtual COMPIN se puede simular la comunicación entre el PIC y LabVIEW. Como se observa, en este caso, no es necesario realizar la interface con el MAX232. Se han conectado 8 LEDS en el puerto D para prender o apagar desde LabVIEW.

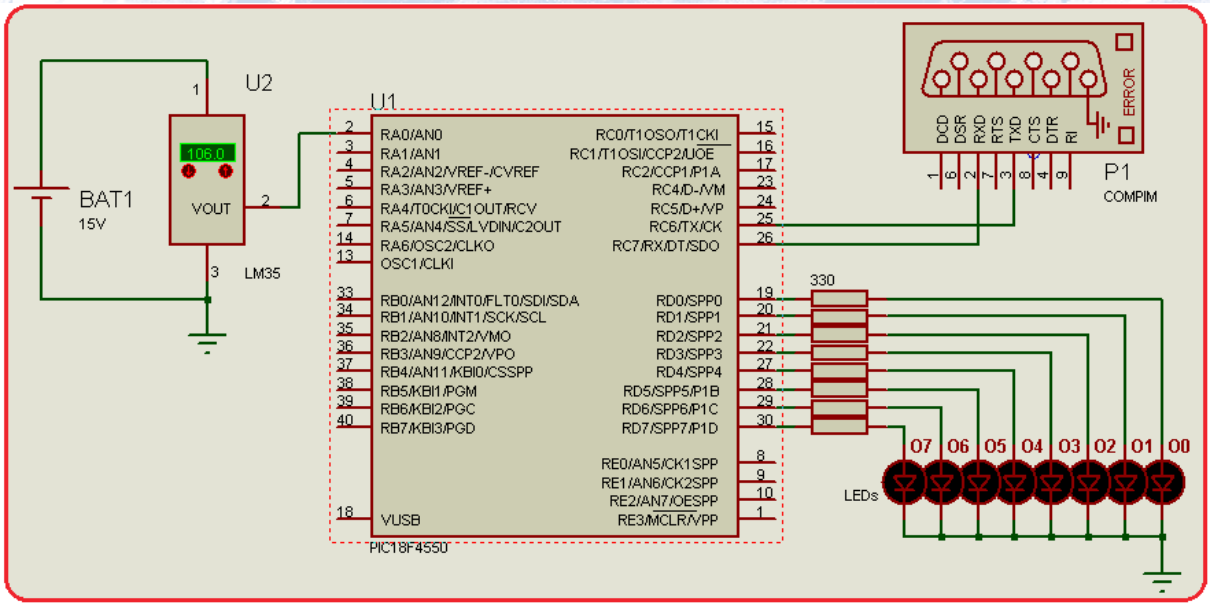


Figura 3.35. Circuito en Proteus para comunicación serial.
Elaborado por: Los Autores.

PROGRAMA EN LABVIEW

La figura 3.36, muestra el panel frontal del programa realizado en LabVIEW. Se pueden apreciar que se tienen ocho interruptores (S0 a S7) que sirven para activar los LEDs que están conectados en el puerto D del microcontrolador. Los LEDs (O0 a 07) del panel se activan según se accione el interruptor.

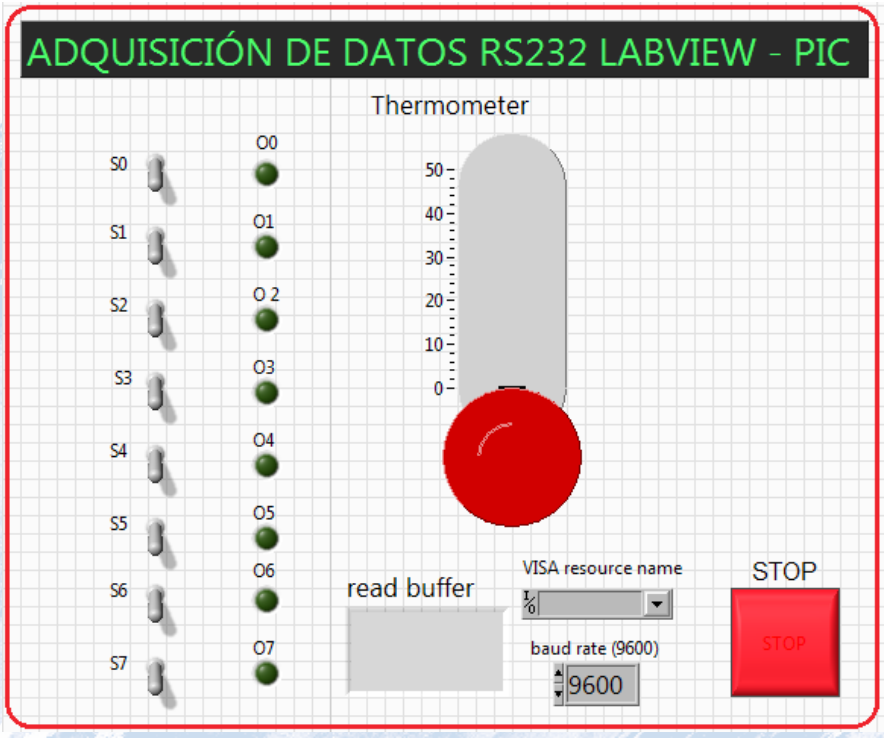


Figura 3.36. Panel Frontal.
Elaborado por: Los Autores.

La figura 3.37, indica el diagrama de bloques completo del programa en LabVIEW. En el proceso de comunicación se distinguen las etapas de apertura del puerto, escritura de los datos, lectura de datos y el cierre del puerto.

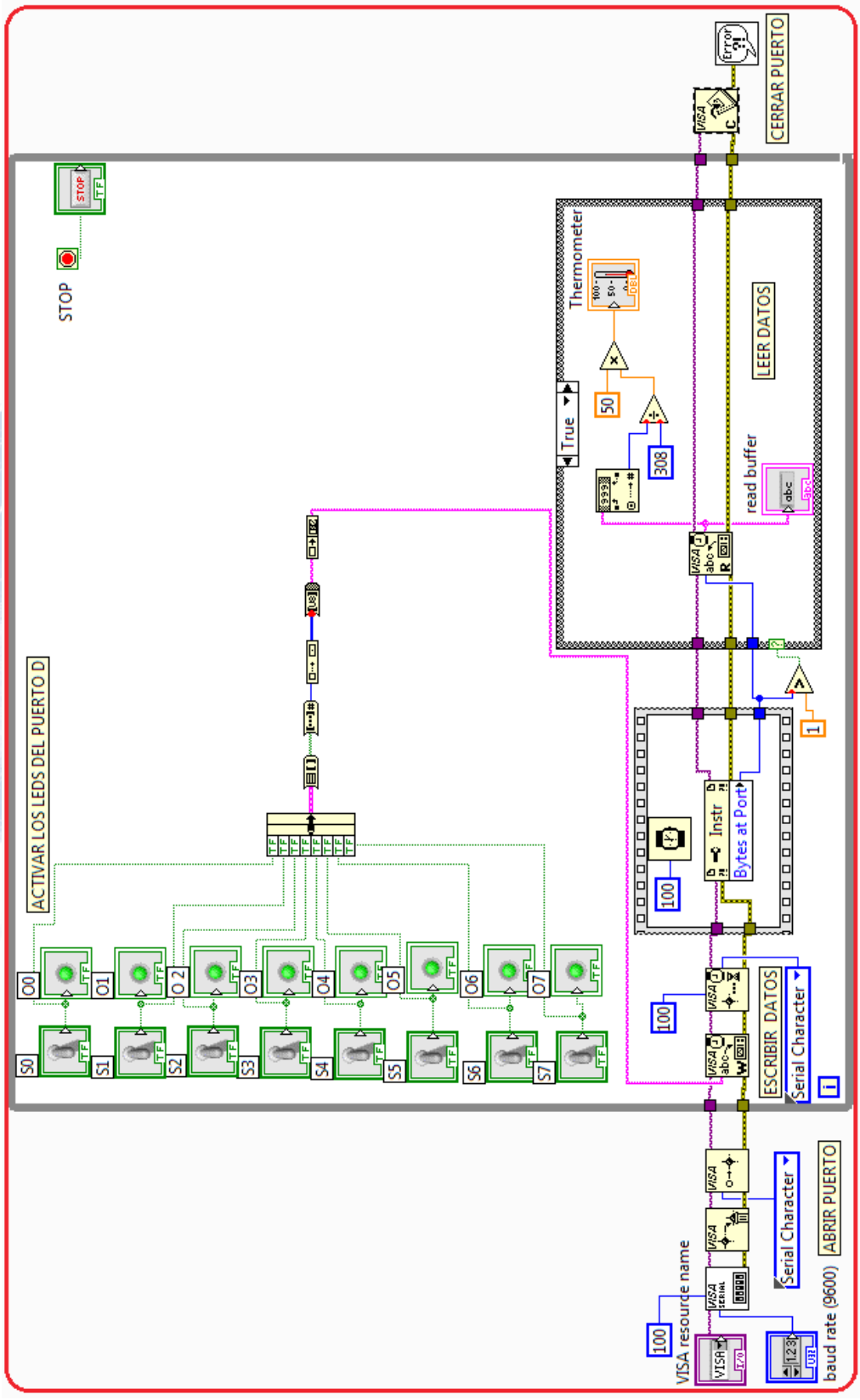


Figura 3.37. Diagrama de bloques del circuito en LabVIEW para la comunicación serial RS232.

Abrir el puerto. Para usar el puerto serial se utiliza el VISA SERIAL, como indica la figura 3.38.

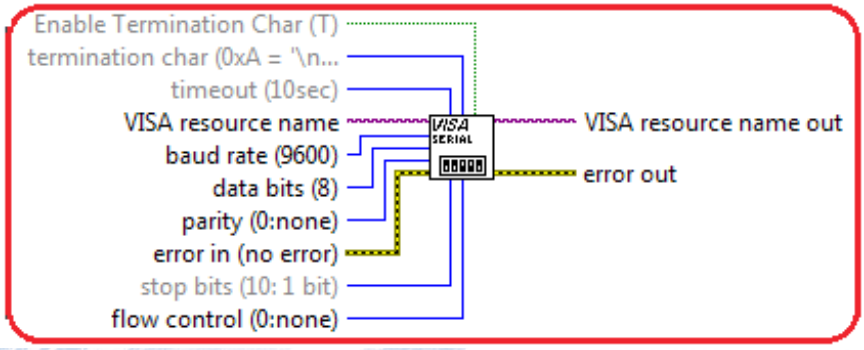


Figura 3.38. VISA SERIAL.
Elaborado por: Los Autores.

Para inicializar el puerto serial creamos en el **VISA resource name** un control con lo cual, en el panel frontal se genera el control para abrir y usar los recursos existentes, como muestra la figura 3.39.

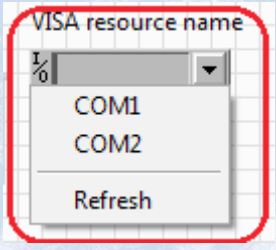


Figura 3.39. VISA resource name.
Elaborado por: Los Autores.

En el control VISA SERIAL, también se debe indicar el BAUD RATE que por lo general es 9600 y el TIMEOUT del VISA que en el ejemplo se utiliza 100 (ms). Los otros controles (número de bits del dato, paridad, etc.) se pueden especificar de acuerdo a las necesidades.

Seguidamente se han descartado todas las ocurrencias o eventos pendientes mediante el **VISA Discard Events** y luego se habilita el puerto mediante el **VISA Enable Event.** Es importante indicar el tipo de evento con el cual responderá el VISA en la presente aplicación como recibirá un carácter serial se utiliza este evento (Serial Character). La figura 3.40, muestra los controles citados.

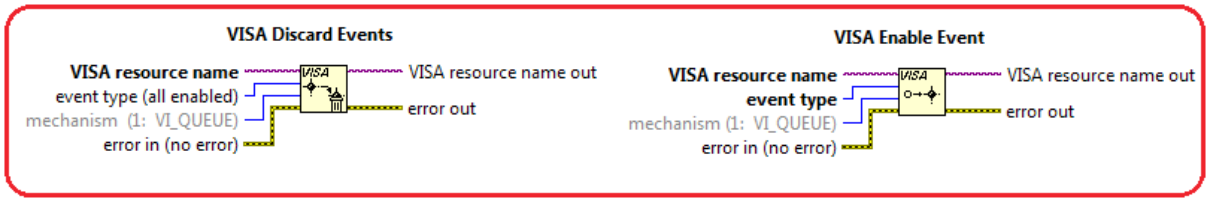


Figura 3.40. VISA Discard Events y VISA Enable Event.
Elaborado por: Los Autores.

Dependiendo de la aplicación y si no se tiene eventos previos se pueden omitir estos dos controles del programa.

Escritura del dato. Para escribir los datos del **Write buffer** para el dispositivo o interface especificado por el **VISA resource name**, utilizamos el **VISA WRITE** y en conjunte con este control utilizamos el **VISA Wait Event** que suspende la ejecución de cualquier aplicación y espera por el tipo de evento especificado durante el periodo **timeout** que en la aplicación es de 100 (ms). La figura 3.41, indica los controles mencionados.

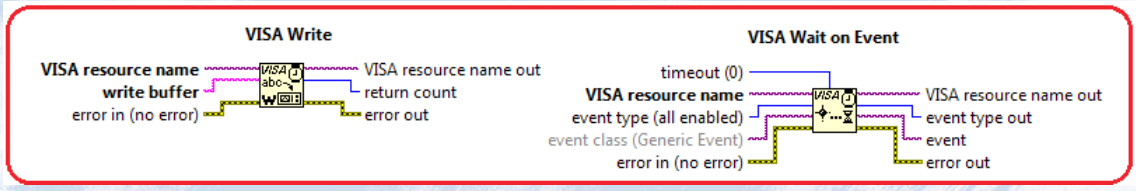


Figura 3.41. VISA Write y VISA Wait Event.
Elaborado por: Los Autores.

Para el control de los LEDs que se encuentran conectados en el Puerto D, el proceso de conversión de datos booleanos a un string, es el mismo explicado en la sección de la comunicación USB.

Para obtener las propiedades del bloque escritura el flujo de datos pasamos por un **Property Node,** como detalla la figura 3.42. Es importante este bloque para acceder y pasar el número de bytes de lectura.

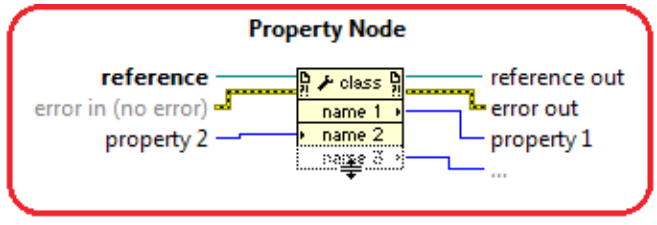


Figura 3.42. Property Node.
Elaborado por: Los Autores.

Lectura del dato. En esta etapa se utiliza un **VISA Read.** Se debe especificar el número de bytes del dispositivo o interface indicado en el **VISA resource name** y retorna el dato en el **read buffer.** La figura 3.43, muestra todas las opciones del **VISA Read**.

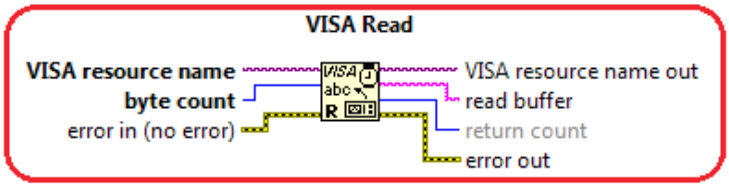


Figura 3.43. VISA Read.
Elaborado por: Los Autores.

El proceso de lectura se realizará siempre que la condición es **True** (verdadera) por lo que el proceso de lectura se realiza en un **Case Structure.** En caso de la condición **False** (Falso) el flujo de datos será directo hasta el **VISA Close**. La figura 3.44, muestra el alambrado en el caso **False.**

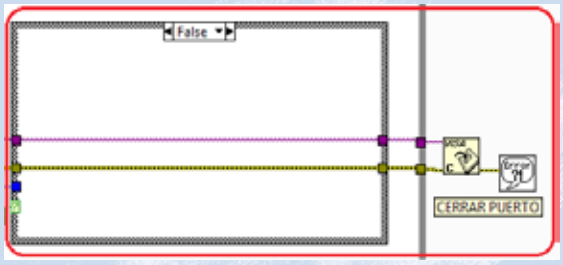


Figura 3.44. Conexiones del Case Structure caso False.
Elaborado por: Los Autores.

El número de bytes leídos (byte count) no es constante, por ejemplo si el rango es 0 - 9 el **byte count** deberá ser 1, de 10 – 99, 2 y así sucesivamente por lo no podemos poner una constante o un control manual. El circuito que controla automáticamente el número de bytes a leer se indica en la figura 3.45. El lector puede interpretar el funcionamiento de esta parte del circuito.

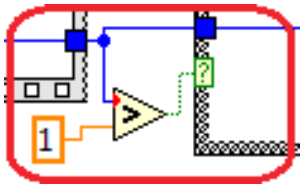


Figura 3.45. Circuito de control del **byte count.**
Elaborado por: Los Autores.

El dato del **read buffer** es un string por lo que se necesita un conversor a dato numérico y a su vez adaptar el valor a la temperatura requerida. En la aplicación se ha medido que el valor máximo que envía el microcontrolador es 308 que equivale a 50 °C. La función **Decimal String To Number,** realiza la conversión y el circuito de adaptación incluido el termómetro se muestra en la figura 3.46.

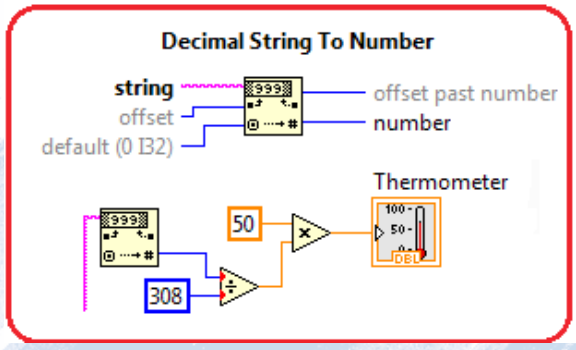


Figura 3.46. Conversión de string a número y adaptación a temperatura.
Elaborado por: Los Autores.

Cierre del puerto. Finalmente, se procede al cierre del puerto para lo cual se utiliza el **VISA Close**, como presenta la figura 3.47.

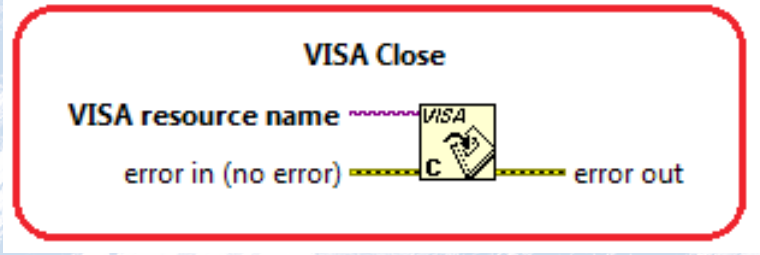


Figura 3.47. VISA Close.
Elaborado por: Los Autores.

Para lograr la comunicación entre LabVIEW y Proteus, debemos crear los puertos seriales virtuales, para esto se utilizó la versión DEMO del VIRTUAL SERIAL PORT DRIVER (VSPD) de ELTIMA Software. Este programa crea los puertos seriales virtuales. La Figura 3.48, muestra los puertos virtuales COM1 y COM2 creados por VSPD.

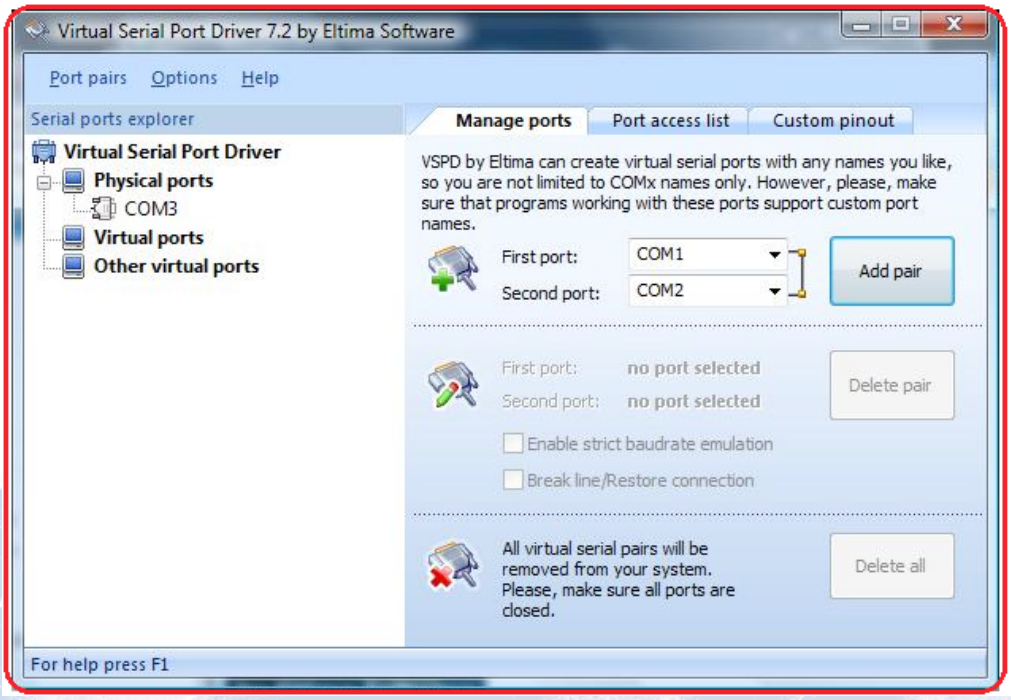


Figura 3.48. Puertos seriales virtuales.
Elaborado por: Los Autores.

Una vez creados los puertos virtuales, en ISIS debemos configurar el puerto serial virtual COMPIM como COM1 y con las especificaciones que indica la figura 3.49. Por tanto, el VISA RESOURCE NAME de LabVIEW se configura como COM2 y los dos programas para una transmisión recepción de 9600 baudios.

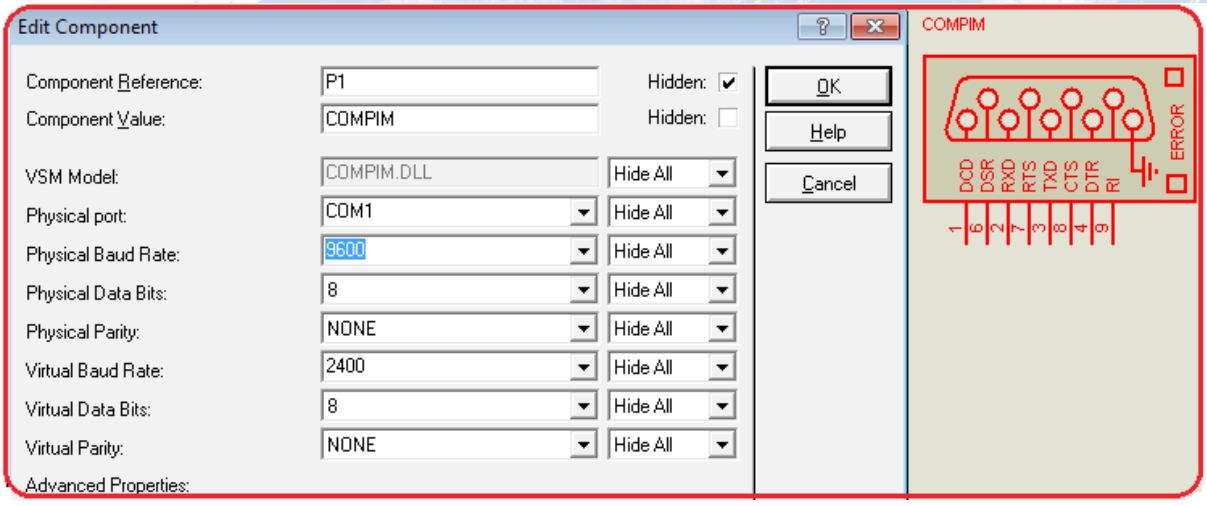
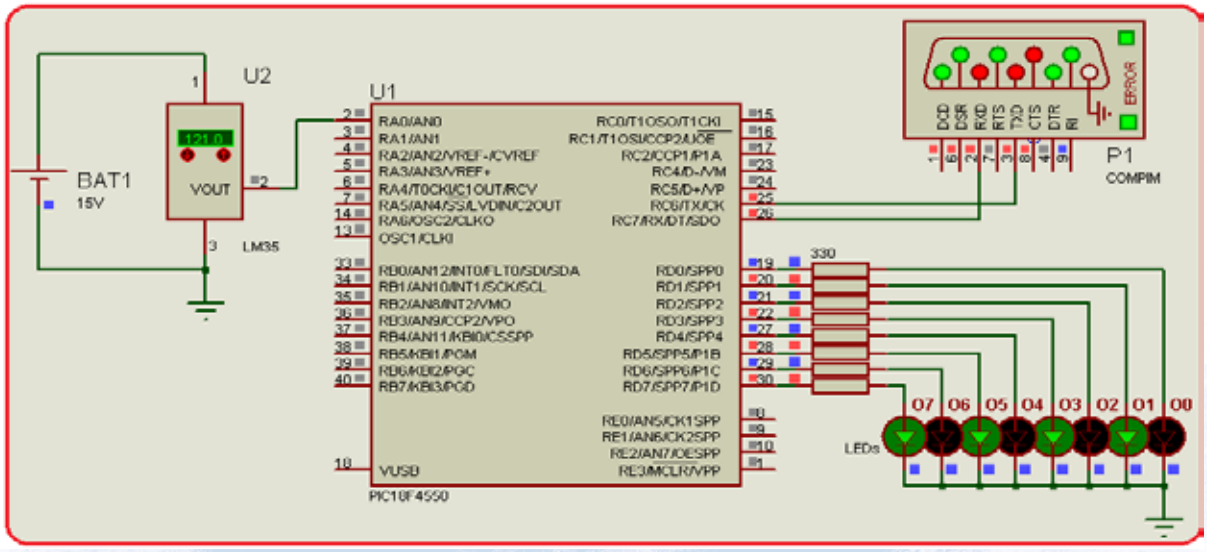
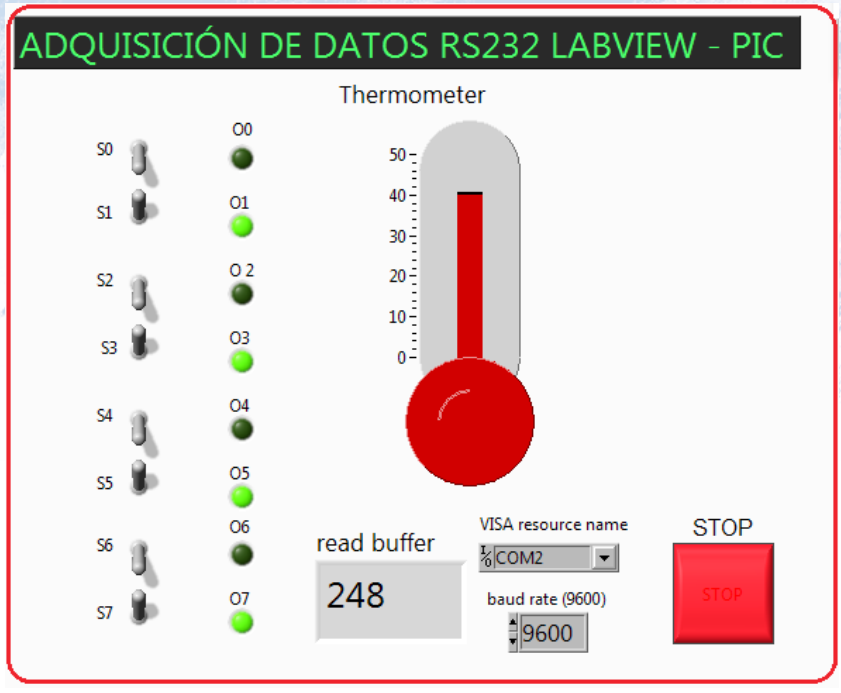


Figura 3.49. Configuración del puerto virtual serial en ISIS.
Elaborado por: Los Autores.

La figura 3.50, muestra la captura de la simulación en ISIS y en LabVIEW. Observe que están activadas las salidas O1, O3, 05 y 07 en ISIS y en LabVIEW. El sensor de temperatura muestra la cantidad de 121 que corresponde a la conversión y dato recibido en LabVIEW a 248 equivalente a una temperatura de 40.25 °C.



a. Captura del circuito simulado en ISIS.



b. Captura del panel frontal.

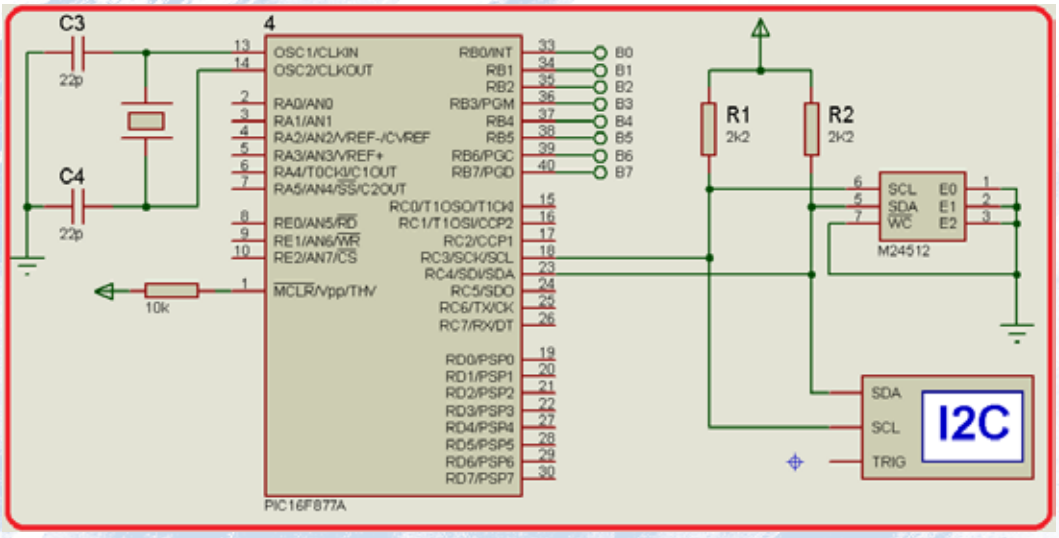
Figura 3.50. Captura de la simulación comunicación serial ISIS – LabVIEW.
Elaborado por: Los Autores.

Ejercicios propuestos

Diseñe un contador Módulo 100 en un microcontrolador 1, los datos envié a un microcontrolador 2, utilizando comunicación UART para visualizarlos en un LCD.

Utilice comunicación I2C y realice el ejercicio 3.1.

Ingresar los datos por teclado para leer y grabarlos en una EEPROM mediante comunicación I2C con PIC. Se debe ingresar la dirección de la memoria para leer o grabar el dato, para luego todo el proceso. La figura 3.51, muestra el hardware necesario.



a). Conexión del PIC y los dispositivos exteriores.

b). Conexiones de los dispositivos de entrada y salida.

Figura 3.51. Comunicación I2C EEPROM – PIC (Ejercicio 3.3).

Elaborado por: Los Autores.

Diseñar un reloj digital, utilizando un RTC (Real Time Clock), como por ejemplo el DS1397, realice la comunicación con un PIC. Muestre los datos en un LCD. Integre la mayor cantidad de funciones del micro para tener un reloj digital completo, reseteo, igualación, etc.

Modifique el programa de LabVIEW (figura 3.27) para observar el resultado del voltaje en un indicador numérico (medidor GAUGE).

Elabore una aplicación para leer dos señales analógicas y visualizarlas en LabVIEW.

Realice una aplicación para un medidor de temperatura. Utilice la interfaz de LabVIEW para observar la temperatura medida.

Elabore un programa para visualizar en LabVIEW el estado del puerto D. Utilice una interface de LEDs, como muestra en la figura 3.52, que activarán si el puerto es 1 o se apagan si el puerto es 0.

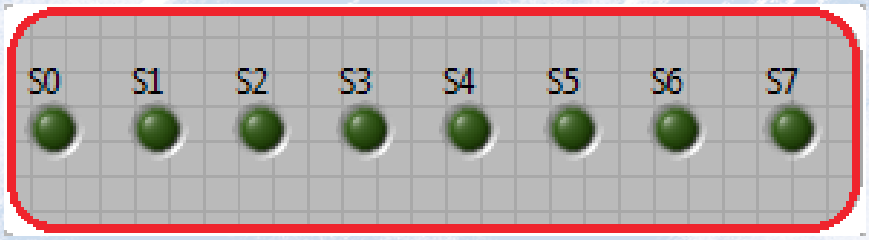
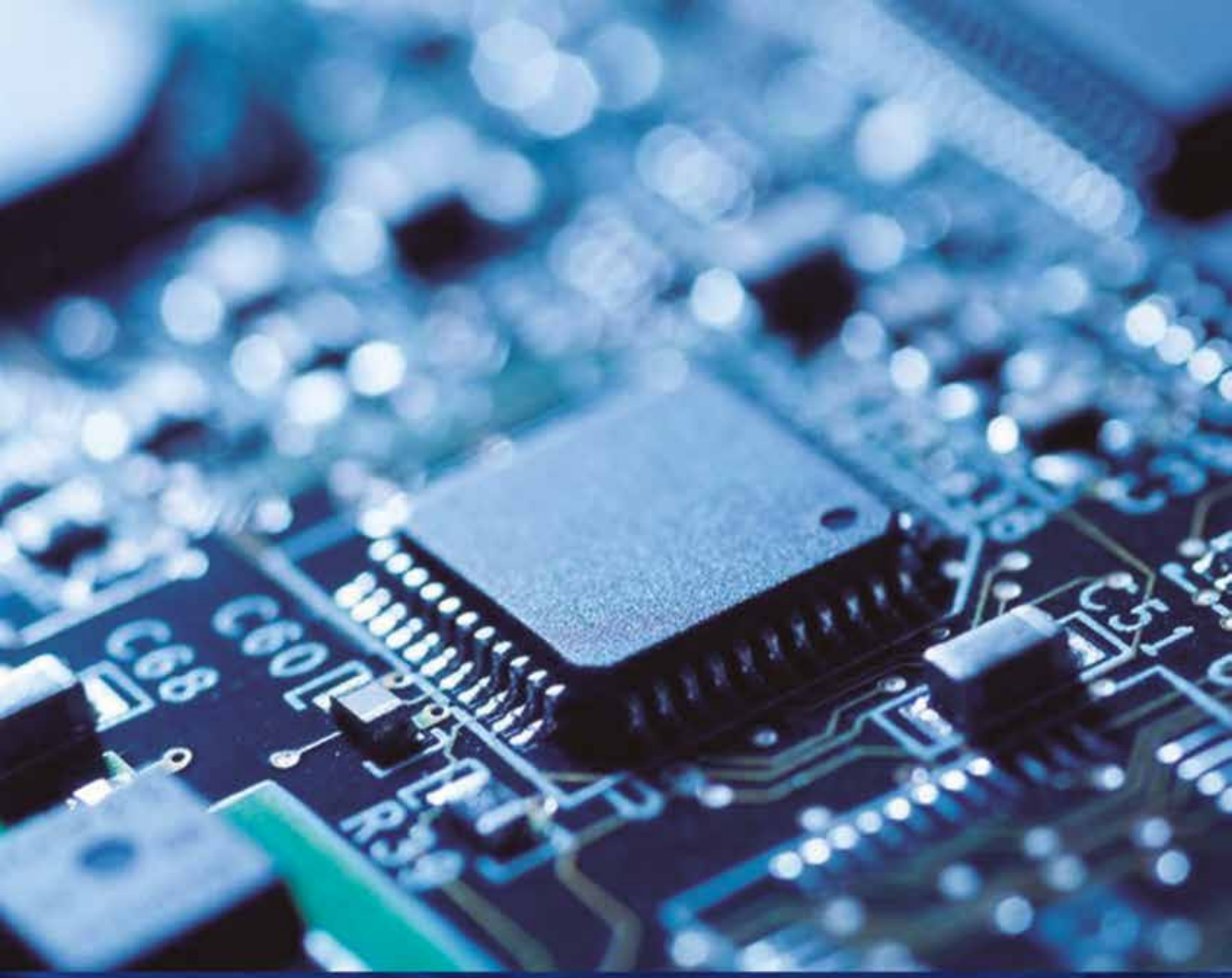


Figura 3.52. Interfaz de salida en LabVIEW para monitorear el estado de las entradas.
Elaborado por: Los Autores.



CAPÍTULO 4

CONTROL DE PEQUEÑOS MOTORES ELÉCTRICOS

En la mayoría de aplicaciones de robótica, juguetes, servomecanismos, limpiaparabrisas, posicionamiento de espejos, faros de vehículos, etc., tienen como elemento que realiza el trabajo mecánico un pequeño motor eléctrico.

Para generar las señales de control es muy común utilizar un microcontrolador y con un adecuado circuito de interfaz, se tiene una gran flexibilidad al momento de implementar los circuitos de control y potencia de los motores. Los motores de mayor uso en este tipo de aplicaciones son los de corriente continua, los servomotores y los motores paso a paso.

Control de motor DC

Los motores DC son los más económicos y comunes. Existen de varios tipos, pero su principio de funcionamiento es el mismo. Presentan las siguientes características:

- Cuando el motor DC recibe su tensión nominal el motor alcanza su máximo torque y velocidad.
- La velocidad del motor DC varía en forma proporcional con la tensión de trabajo.
- Al invertir la polaridad de la tensión que ingresa al motor se invierte el giro.

Mando directo de un motor DC

Una de las aplicaciones más comunes es el mando directo de un motor DC (encendido – apagado). El programa es el mismo que se utiliza para prender y apagar un LED. El circuito de interface entre el microcontrolador y motor es lo que difiere dependiendo el tipo de motor o carga que se maneje. Se podría manejar la carga mediante un optoacoplador, transistor, tiristor o relé. Para el ejercicio se propone manejar un motor DC de 12V, para lo cual hemos utilizado un transistor TIP122, como se indica en el circuito de la figura 4.1.

Los pulsadores *PRENDER* y *APAGAR*, servirán para activar o desactivar el motor. Es necesario también colocar en paralelo con el motor un diodo en polarización inversa, para cortocircuitar las corrientes de autoinducción del motor que generan al momento de prender o pagar el motor y que podrían afectar al transistor.

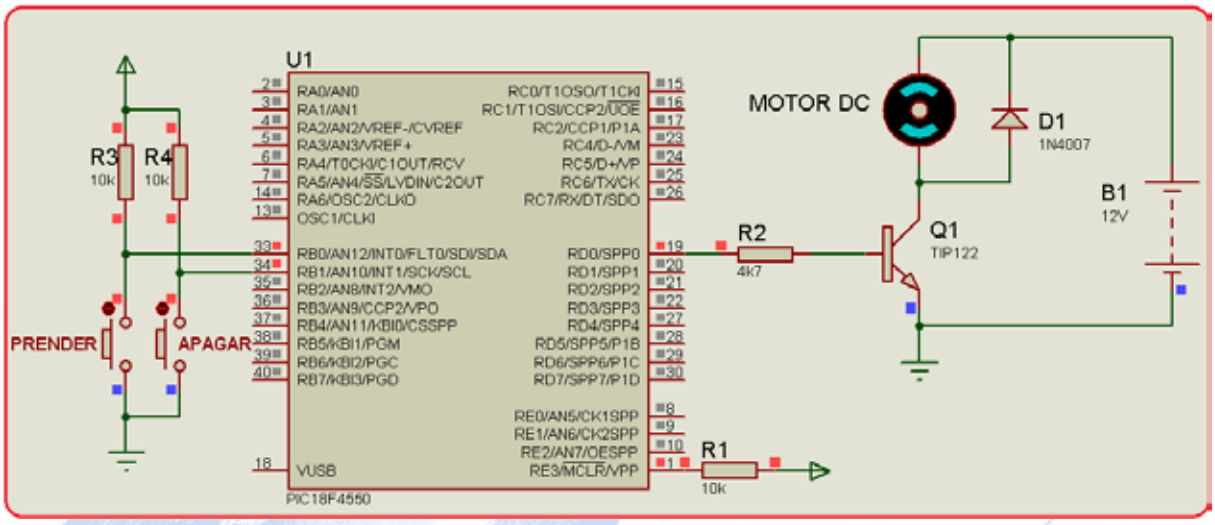


Figura 4.1. Mando de motor DC.
Elaborado por: Los Autores.

Ejercicio 4.1. Mando directo de motor DC.

Si se activa el pulsador PRENDER (pin RB0), se coloca el pin RD0 en alto, con lo cual el transistor se satura conectando el motor a la fuente de 12V. Al accionar el pulsador APAGAR (pin RB1), la base del transistor recibe 0 V y el transistor se abre, apagando al motor. La resistencia en la base del transistor R_B podemos calcular así:

$$R_B = (Vp - V_{BE})/IB_{SAT}$$

Donde,

Vp, es el voltaje del puerto (5V).

V_{BE}, el voltaje de base emisor del transistor (0,7V).

$IB_{SAT,}$ la corriente de base de saturación del transistor.

Se supone que se utiliza un motor de 12 V_{CC} (el modelo S330014, por ejemplo) con una corriente de 800 mA a plena carga. Como el TIP122 tiene una ganancia de 1000, la corriente de base de saturación es:

$$IB_{SAT} = I_C/\beta_{CC} = 800\ mA/1000 = 0.8\ mA.$$

Por tanto, la resistencia de base es igual:

$$R_B = (Vp - V_{BE})/IB_{SAT}$$

$$R_B = (5 - 0.7V)/0.8mA$$

$$RB = 5.375K\Omega$$

Se elige una resistencia comercial de 4.7KΩ, con lo cual se garantiza que transistor entregue la corriente necesaria al motor.

PROGRAMA:

```
#include <18F4550.h>                    //Librería para usar el PIC18F4550.
#fuses INTRC_IO, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD      //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#use standard_io(b)                     //Usa la librería estándar B.
#use standard_io(d)                     //Usa la librería estándar D.

void main(){                            //Función main.

 while(TRUE){                           //Bucle infinito.
  if (input(PIN_B0) ==0){               //Si el PIN B0 = 0....
    output_high(PIN_D0);                //Activa salida D0, para prender el motor.
  }
  if (input(PIN_B1) == 0){              //Si el Pin B1= 0.....
    output_low(PIN_D0);                 //Desactiva salida D0, para apagar el motor.
  }
  }                                     //Fin del bucle infinito.
}                                       //Fin del main.
```

Inversión de giro de un motor DC

Para invertir el giro de un motor DC, se debe invertir la polaridad de la tensión que alimenta al motor. La forma muy común de realizar esta inversión es mediante un puente H, construido por transistores o directamente utilizando el circuito integrado L293D.

El circuito integrado L293D

El circuito integrado L293D de Texas Instruments, es muy útil para el control de motores DC y de paso bipolares. Es semejante a un circuito puente H realizado por transistores. La figura 4.2, indica la estructura interna del L293D, en el cuál se observa cuatro drivers que manejan hasta 600 mA con tensiones que van desde los 4.5V hasta los 36V.

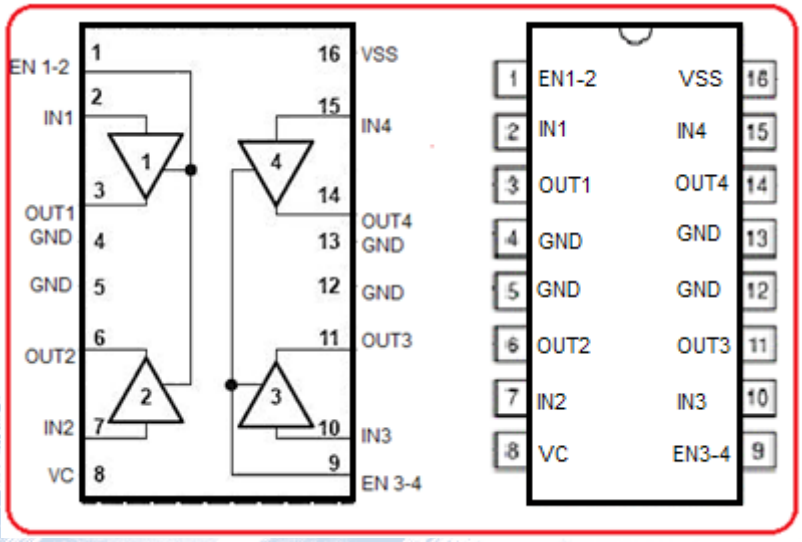


Figura 4.2. Estructura interna y diagrama de conexiones del CI L293D.
Fuente: L293x Quadruple Half-H Drivers. Texas Instruments. Internet.

La tabla 4.1, indica los niveles lógicos para el funcionamiento del L293D.

ENTRADAS		**SALIDA**
IN	**EN**	**OUT**
H	H	H
L	H	L
X	L	Z
H: Nivel alto. L: Nivel Bajo. X: Nivel alto o bajo. Z: Alta impedancia.		

Tabla 4.1. Funcionamiento del L293D.
Elaborado por: Los Autores.

Ejercicio 4.2. Control del giro de un motor DC.

La figura 4.3, muestra el diagrama de conexiones del L293D con el motor DC y el microcontrolador.

El motor puede girar en cualquier sentido si está apagado. La variable M censa el estado del motor: M=0, motor apagado y M=1, motor prendido. El pulsador conectado en el puerto B0 activa la salida D0 y el motor girará

en un sentido siempre que M=0. Con el pulsador conectado en el puerto B1, se activa la salida D1, para que el motor gire en el otro sentido de giro. Para evitar cambiar el sentido de giro cuando está el motor funcionando, la variable M cambia a 1. El motor se apaga al accionar el pulsador que está conectado en el puerto RB2. Nuevamente toma el valor 0 y queda listo para accionar al motor.

PROGRAMA:

```
#include <18F4550.h>                    //Librería para usar el PIC18F4550.
#fuses INTRC_IO, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD, NOMCLR    //Configuración de fusibles.
#use delay (clock=4000000)              //F_OSC = 4 MHz.
#use standard_io(b)                     //Usa la librería estándar B.
#use standard_io(d)                     //Usa la librería estándar D.
int1 M=0;                               //Variable M determina detecta si el motor está apagado.

void main(){                            //Función main.

 while(TRUE){                           //Bucle infinito.
  if (input(PIN_B0)==0 && M==0){        //Si se acciona el pulsador de B0 y M=0, el motor….
    output_high(PIN_D0);                //…. Se prende en sentido derecho, por ejemplo.
    M=1;                                //M=1, para que no se pueda accionar  el otro sentido de giro.
  }
  if (input(PIN_B1)==0&&M==0){          //Si se acciona el pulsador de B1 y M=0, el motor….
    output_high(PIN_D1);                //…. Se prende en sentido izquierdo, por ejemplo.
    M=1;                                //M=1, para que no se pueda accionar  el otro sentido de giro.
  }

  if (input(PIN_B2) == 0){              //Si se acciona el pulsador de B2, el motor….
    output_low(PIN_D0);                 //….se apaga en cualquier sentido que este activado.
    output_low(PIN_D1);
    M=0;                                //M=0, listo para prender el motor nuevamente.
  }
 }                                      //Fin del bucle infinito.
}                                       //Fin del main.
```

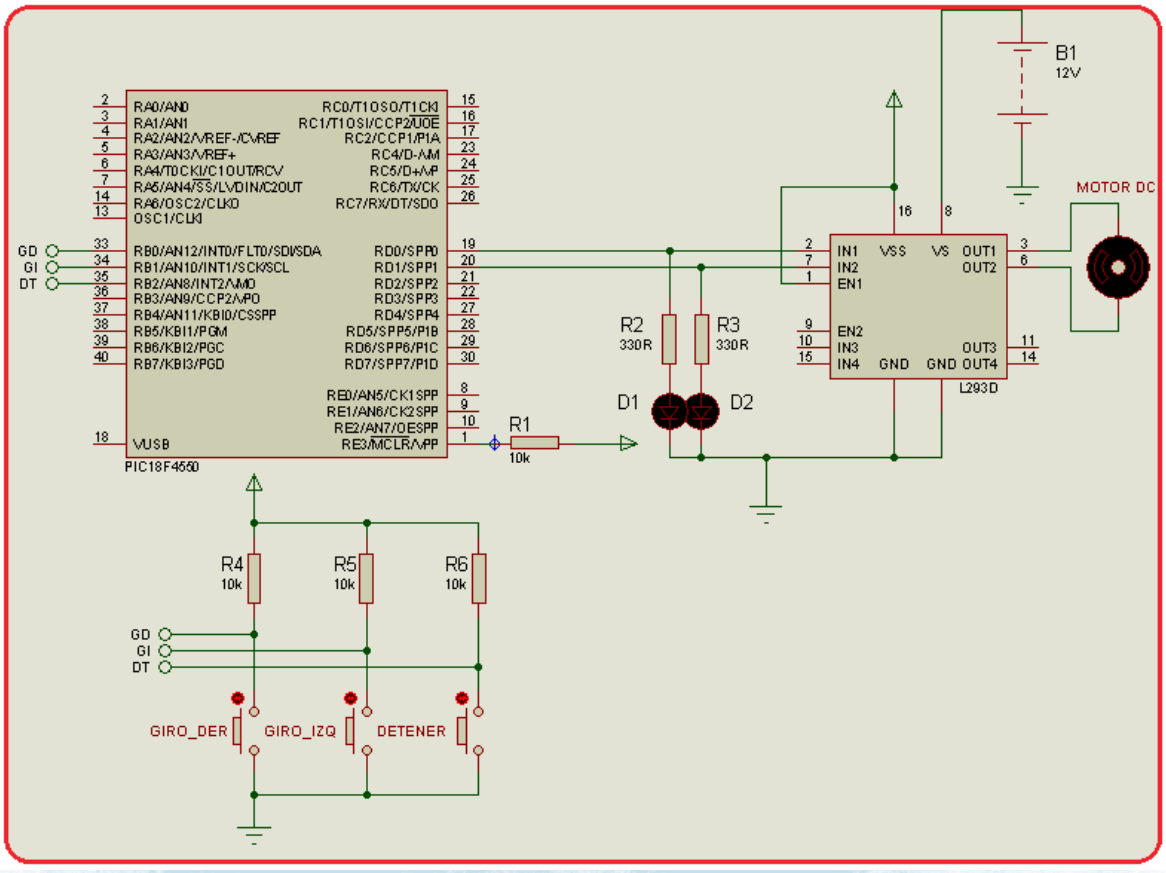


Figura 4.3. Inversión de giro de un motor DC.
Elaborado por: Los Autores.

Control de un motor DC

La forma más sencilla de controlar la velocidad de un motor DC es disminuyendo o aumentado la tensión que ingresa al motor. Para realizar esto podemos controlar el ancho del pulso que ingresa al motor mediante señal PWM.

Ejercicio 4.3. Control de velocidad de un motor DC.

El programa consiste en obtener una señal PWM variable que para controlar la velocidad de un motor DC. Mediante un pulsador se arranca el motor aplicando un PWM a un 50% de DC. Se dispone de dos pulsadores uno para incrementar y otro para disminuir el ancho del pulso en aproximadamente en el 2% del DC (Duty cycle).

En este ejercicio se asigna un **PR2= 255** para tener una resolución de 10 bits, por tanto, el **DC** inicial será de **512**, para obtener un DC del 50%.

Recuerde que el DC con 10 bits varia de 0 a 1023 (1024 valores). Con esto se tiene un T_{PWM} de:

$T_{PWM} = (1/F_{OSC})*4*t2div*(PR2+1)$, *para t2div= 16 (máximo periodo).*

$T_{PWM} = (1/12MHz)*4*16*(256)$

$\boldsymbol{T_{PWM} = 1{,}36\ ms}$.

El incremento o decremento del ancho del pulso es del 2%, o sea:

$\Delta DC = 0.02DC = 0.02*512$

$\Delta DC = 10.24$.

Tomaremos un $\boldsymbol{\Delta DC = 10}$.

El DC inicial lo se asigna a una variable **valor:**

valor= 512

El ancho de pulso se incrementa mediante la relación:

valor= valor +10;

No debe sobrepasar los 1023, para no exceder el límite del DC.

El ancho de pulso se disminuye mediante la relación:

valor= valor - 10;

No debe llegar a 0 para no tener un periodo nulo.

Con estas consideraciones se desarrolla el programa.

PROGRAMA:

```
#include <18f4550.h>                    //Librería para usar el PIC18F4550,
#fuses XT,WDT,NOPROTECT,NOPUT, NOPBADEN          //Configuración de fusibles.
#use delay (clock=12000000)              //F_OSC = 12 MHz.
#include <stdlib.h>                     //Librería stdlib.h.
#BYTE port_b= 0xF81                     //Identificador para el puerto b en  la localidad 0xF81.
#BYTE port_c= 0xF82                     //Identificador para el puerto c en  la localidad 0xF82.
#BYTE port_d= 0xF83                     //Identificador para el puerto d en  la localidad 0xF83.
#define prender bit_test(port_d,0)      //Define al puerto RD0 como prender.
#define apagar bit_test(port_d,1)       //Define al puerto RD0 como apagar.
#define INC bit_test(port_d,2)          //Define al puerto RD0 como INC.
#define dec bit_test(port_d,3)          //Define al puerto RD0 como dec.
short x=0;                              //Bandera para detectar el cambio de estado del pulsador de inicio.
long int valor=0;                       //Variable para almacenar el Duty de la señal PWM
```

```
void main(void) {                       //Función principal main.
  set_tris_d(0xff);                     //Puerto d como entrada.
  set_tris_b(0x00);                     //Puerto b como salida.
  port_b=0;                             //Borra todo el puerto b.

 while(true){                           //Bucle infinito.
  if (prender==0 )                       //Inicia la generación del PWM si se acciona pulsador prender.
  {
            /          / x se fija en 1 para mantener activo al circuito.
                      // (bandera de memoria, recuerda la orden de inicio).
    x =1;
    valor=512;                          //La señal PWM empieza a generar con un duty próximo al 50%.
    setup_ccp1(CCP_PWM);                //Configura CCP1 como PWM.
    setup_timer_2(T2_DIV_BY_16, 255, 1);//Fija la frecuencia a:1365.33 us o 0.73 KHz.
    bit_set(port_b,0);                  //Indica inicio del circuito.
    while(prender==0){}                 //No hace nada mientras el pulsador está activado
  }
  while(x==1)                           //Repite mientras x es 1, orden de inicio del pulsador prender.
  {
    set_pwm1_duty(valor);               //PWM con ancho de pulso equivalente a la variable valor.
   if (INC==0)                          //Si hay pulso para incrementar…
    {
      if (valor <= 1020)                //… valor es menor o igual a 1020
      {
       valor = valor +10;               //Incrementa valor en 10.
       while (INC==0){ }                //Pulsador activo, no realiza nada (anti rebote).
      }
    }
   if (dec==0)                          //Si hay pulso para decrementar…
    {
      if (valor >= 20)                  //… valor es mayor o igual a 20
      {
       valor = valor -10;               //Decrementa valor en 10.
       while (dec==0){ }                //Pulsador activo, no realiza nada (anti rebote).
      }
    }
  if (apagar == 0 )                     //Pulsador apagar accionado…
    {
      bit_clear(port_b,0);              //Apaga LED que indica el funcionamiento del circuito.
      valor =0;                         //valor = 0
      set_pwm1_duty(valor);             //Reinicia el módulo PWM salida 0.
      x=0;                              //Bandera x= 0, sin acción el circuito.
      break;                            //Sale del lazo.
    }
  }
}                                       //Fin del bucle infinito.
```

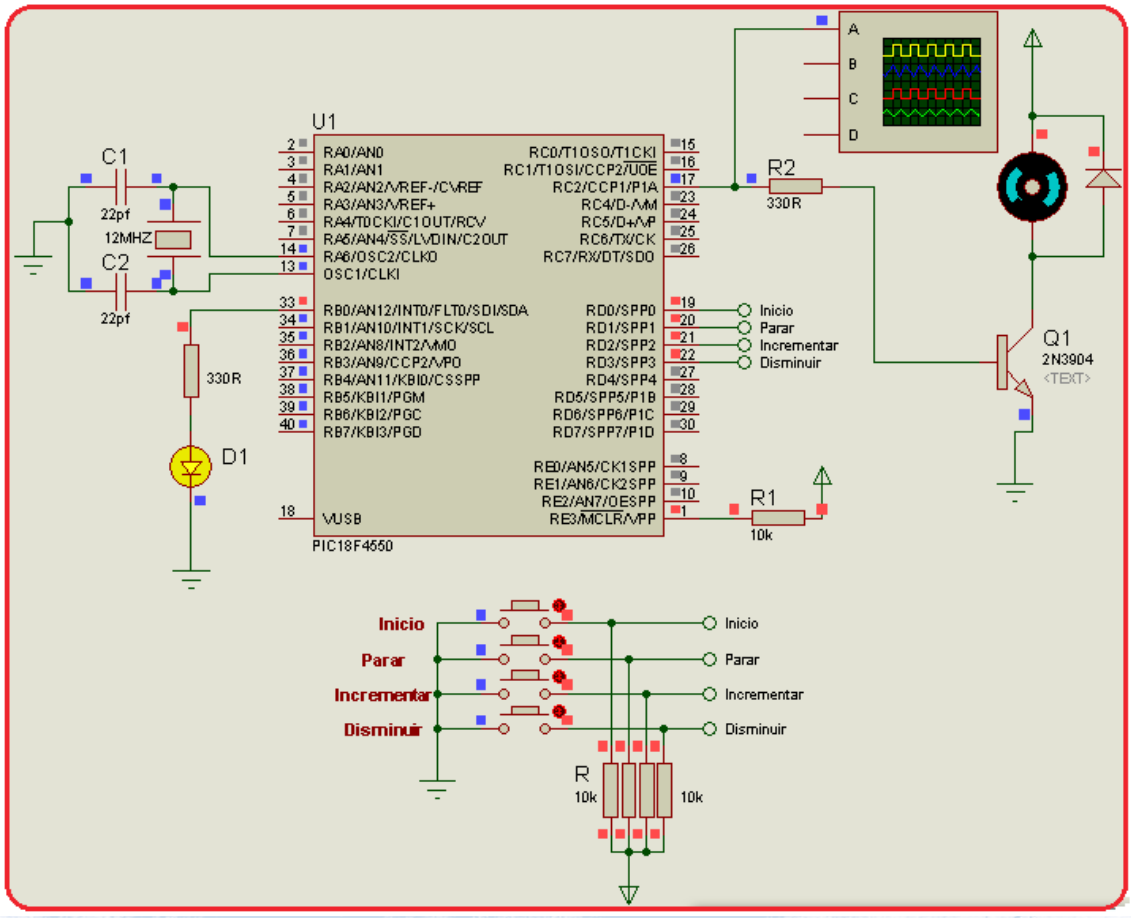


Figura 4.4. Control de velocidad de un motor DC.
Elaborado por: Los Autores.

Control de motor paso a paso unipolar

El motor paso a paso unipolar está formado por dos bobinas A y B. Las bobinas están divididas por un terminal central que forman un punto común. Este tipo de motores pueden tener 5 o 6 cables terminales. Los terminales comunes de A y B se unen para de esta forma facilitar el control del motor.

El factor más importante para la elección de un motor paso a paso es el número de grados que girará el rotor por cada pulso que se aplique. Los grados por paso se calculan dividiendo 360 (una vuelta completa) por la cantidad de pasos que da el motor. Es muy común tener los grados por paso desde: 0,72°, 1,8°, 3,6°, 7,5°, 15° y hasta 90°. Los valores de grado por paso se conocen como resolución de giro del motor. La figura 4.5, indica la estructura simplificada de un motor paso a paso unipolar. Los colores indicados son los más comunes que usan los fabricantes para identificar las bobinas.

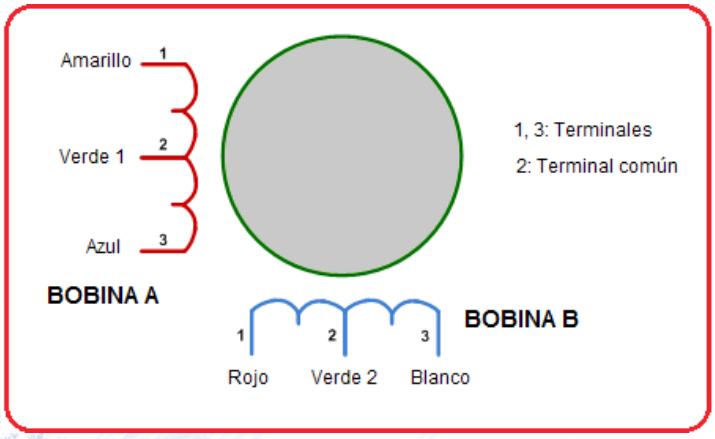


Figura 4.5. Esquema de un motor unipolar paso a paso.
Elaborado por: Los Autores.

Para conectar las bobinas del motor unipolar con el microcontrolador se utiliza el circuito integrado ULN2803 o mediante transistores.

El circuito integrado ULN2803a

El CI ULN2803A de Texas Instruments, contiene un array de 8 transistores tipo Darlington, capaces de manejar cargas de hasta 500 mA. El esquema simplificado del CI ULN2803A se indica en la figura 4.6.

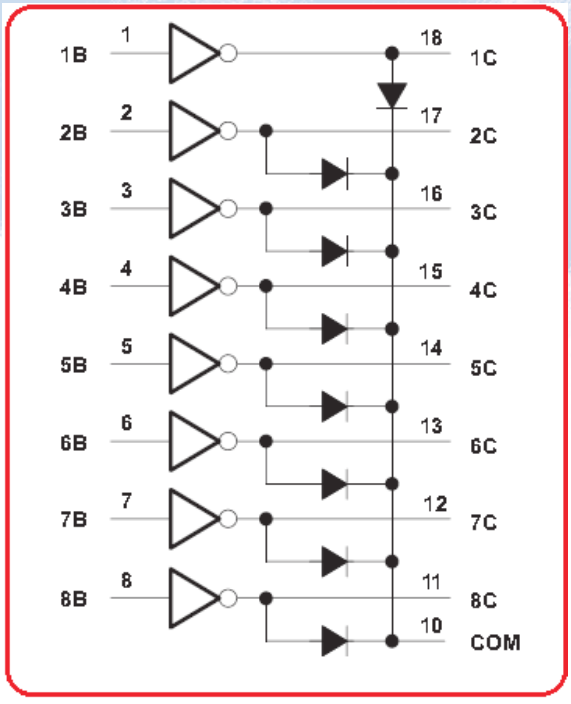


Figura 4.6. Esquema simplificado del CI ULN2803A.
Fuente: ULN2803A Darlington Transistor Arrays. Texas Instruments.

El diagrama de conexiones del circuito de control del motor unipolar utilizando como interface el CI ULN2803 y el microcontrolador se muestra en la figura 4.7.

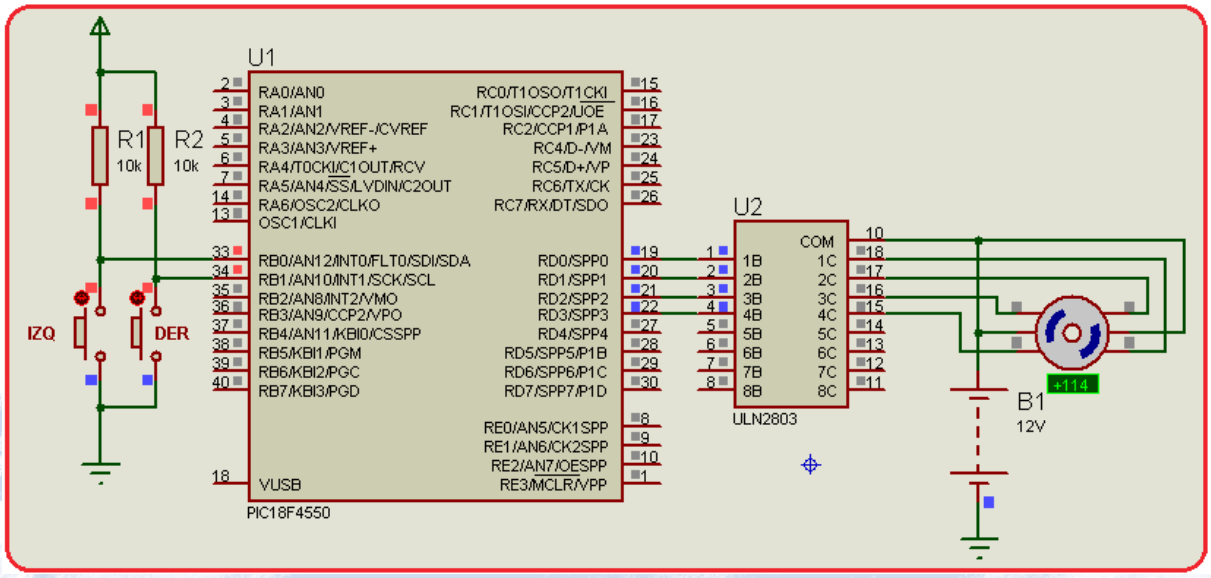


Figura 4.7. Circuito para manejar el motor unipolar.
Elaborado por: Los Autores.

Existen tres métodos de control para los motores paso a paso unipolares. Estos se denominan:

- Paso simple.
- Medio paso.
- Paso doble.

A continuación se realiza el control utilizando estos métodos.

Paso Simple

La figura 4.8, indica la distribución interna de las bobinas en cuatro polos. Los polos A y C forma la primera bobina y los polos B y D forman la segunda.

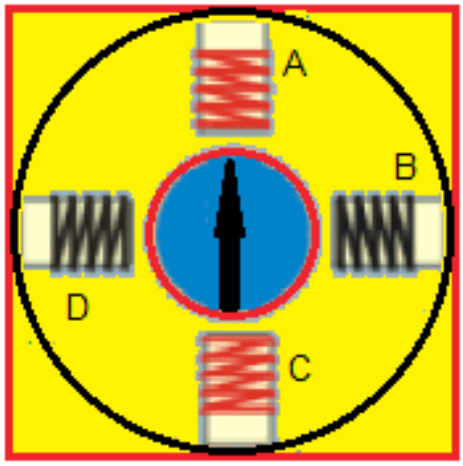


Figura 4.8. Distribución interna de los bobinados del motor paso a paso unipolar.
Elaborado por: Los Autores.

En el paso simple las bobinas reciben los pulsos en secuencia una a la vez, empezando desde la bobina A hasta llegar a la D. La tabla 4.2, muestra la secuencia de pulsos para un paso simple. Dado que en cada paso las bobinas no se energizan, el torque del motor es muy pobre.

PASO	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Tabla 4.2. Secuencia paso simple.
Elaborado por: Los Autores.

Para invertir el giro del motor paso a paso unipolar la secuencia de pulsos se invierte, es decir, se empieza en D y se termina en A como se indica en la tabla 4.3.

PASO	A	B	C	D
1	0	0	0	1
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0

Tabla 4.3. Secuencia de pulsos para invertir el giro del motor paso a paso unipolar.
Elaborado por: Los Autores.

Ejercicio 4.4. Mando de motor paso a paso unipolar, secuencia paso simple.

```
#include <18F4550.h>                          //Librería para usar el PIC18F4550.
#fuses INTRC, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD, NOMCLR /       /Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC= 4 MHz.
#use standard_io(b)                           //Usa la librería estándar B.
#use standard_io(d)                           //Usa la librería estándar D.
int i=1;                                      //Controla el número de pulsos aplicados al motor.

void time(){
  delay_ms (60);                              //Controla el tiempo del pulso.
  return;
}

void time1(){
  delay_ms (60);                              //Controla el tiempo del pulso.
  return;
}
void main(){                                  //Función principal main.

 while(TRUE){                                 //Bucle infinito.
                                               //Giro a la izquierda.
   if (input(PIN_B0) ==0){                     //Si el pin B0 es 0.....
      for (i=1; i<=4;++i){                     //Aplica 4 pulsos  a las bobinas.
        output_high(PIN_D0);                   //Energiza la bobina A.
        time();                                //……..
        output_low(PIN_D0);            .
        time1();
        output_high(PIN_D1);
        time();
        output_low(PIN_D1);
         time1();
        output_high(PIN_D2);
        time();
        output_low(PIN_D2);
        time1();
        output_high(PIN_D3);
        time();
        output_low(PIN_D3);
        time1();
     }
  }
 if (input(PIN_B1) ==0){                       //Giro a la derecha.
      for (i=0; i<=4;++i){
        output_high(PIN_D3);
        time();
        output_low(PIN_D3);
        time1();
        output_high(PIN_D2);
        time();
        output_low(PIN_D2);
        time1();
        output_high(PIN_D1);
        time();
        output_low(PIN_D1);
        time1();
        output_high(PIN_D0);
        time();
        output_low(PIN_D0);
        time1();
```

Medio Paso

La secuencia medio paso significa energizar las bobinas en secuencia primero una luego dos y así sucesivamente. Con esta forma de aplicación de pulsos siempre estarán energizadas al menos una bobina con lo cual el torque del se mejora notablemente. La tabla 4.4, presenta la secuencia de energización de las bobinas para medio paso.

PASO	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Tabla 4.4. Secuencia medio paso para control de motor PAP unipolar.
Elaborado por: Los Autores.

La aplicación inversa de los pulsos en las bobinas (empezando desde la bobina D), hará que el motor invierta el giro.

Ejercicio 4.5. Control de motor paso a paso unipolar utilizando secuencia medio paso.

PROGRAMA:

```
#include <18F4550.h>                          //Librería para usar el PIC18F4550.
#fuses INTRC, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD, NOMCLR /      /Configuración de fusibles.
#use delay (clock=4000000)                    //F_OSC= 4 MHz.
#use standard_io(b)                           //Usa la librería estándar B.
#use standard_io(d)                           //Usa la librería estándar D.
int i=1;                                      //Controla el número de pulsos aplicados al motor.

void time(){
  delay_ms (60);                              //Controla el tiempo del pulso.
  return;
}
```

```
void time1(){
  delay_ms (60);                          //Controla el tiempo del pulso.
  return;
}

void main(){                              //Función principal main.

 while(TRUE){                             //Bucle infinito.


  if (input(PIN_B0) ==0){                 //Giro a la izquierda.
     for (i=0; i<=4;++i){                 //……….
       output_high(PIN_D0);
       output_low(PIN_D3);
       time();
       output_high(PIN_D1);
       time();
       output_low(PIN_D0);
       time();
       output_high(PIN_D2);
       time();
       output_low(PIN_D1);
       time();
       output_high(PIN_D3);
       time();
       output_low(PIN_D2);
       time();
       output_high(PIN_D0);
       time();
    }
  }
   //Giro a la derecha.
  if (input(PIN_B1) ==0){
     for (i=0; i<=4;++i){
       output_high(PIN_D0);
       output_high(PIN_D3);
       time();
       output_low(PIN_D0);
       time();
       output_high(PIN_D2);
       time();
       output_low(PIN_D3);
       time();
       output_high(PIN_D1);
       time();
       output_low(PIN_D2);
       time();
       output_high(PIN_D0);
```

Paso Doble

Con el paso doble se activan las bobinas de dos en dos con lo que el motor genera una campo magnético más potente que atraerá con más fuerza y retendrá el rotor del motor en el sitio, mejorando el torque del motor. Los pasos en esta secuencia, también son algo más bruscos en relación a las secuencias anteriores.

PASO	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Tabla 4.5. Secuencia paso doble para control de motor PAP unipolar.
Elaborado por: Los Autores.

Igual que en los casos anteriores, la aplicación inversa de los pulsos en las bobinas (empezando desde la bobina D), hará que el motor gire en el otro sentido.

Ejercicio 4.6. Control de motor paso a paso unipolar utilizando secuencia paso doble.

PROGRAMA:

```
#include <18F4550.h>                          //Librería para usar el PIC18F4550.
#fuses INTRC, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD, NOMCLR      //Configuración de fusibles.
#use delay (clock=4000000)                    //FOSC= 4 MHz.
#use standard_io(b)                           //Usa la librería estándar B.
#use standard_io(d)                           //Usa la librería estándar D.
int i=1;                                      //Controla el número de pulsos aplicados al motor.

void time(){
  delay_ms (60);                              //Controla el tiempo del pulso.
  return;
}

void time1(){
  delay_ms (60);                              //Controla el tiempo del pulso.
  return;
}

void main(){                                  //Función principal main.
```

```
while(TRUE){                                    //Bucle infinito.

 if (input(PIN_B0) ==0){                        //Giro a la izquierda.
    for (i=0; i<=4;++i){                        //……….
      output_low(PIN_D3);
      output_high(PIN_D0);
      output_high(PIN_D1);
      time();
      output_low(PIN_D0);
      output_high(PIN_D2);
      time();
      output_low(PIN_D1);
      output_high(PIN_D3);
      time();
      output_low(PIN_D2);
      output_high(PIN_D0);
      time();
   }
 }

 if (input(PIN_B1) ==0){                        //Giro a la derecha.
     for (i=0; i<=4;++i){
      output_low(PIN_D2);
      output_high(PIN_D0);
      time();

      output_low(PIN_D1);
      output_high(PIN_D3);
      time();

      output_low(PIN_D0);
      output_high(PIN_D2);
      time();

      output_low(PIN_D3);
      output_high(PIN_D0);
      output_high(PIN_D1);
      time();
   }
 }
}                                               //Fin del bucle infinito.
```

Control de motor paso a paso bipolar

El motor paso a paso bipolar tiene cuatro terminales que salen del motor que corresponde a las dos bobinas que contiene el dispositivo.

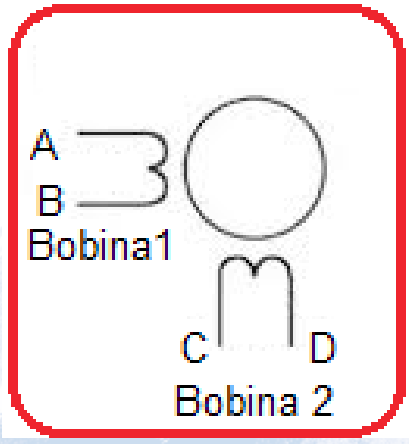


Figura 4.9. Esquema simplificado de un motor bipolar.
Elaborado por: Los Autores.

Los motores bipolares necesitan la inversión de la corriente que circula en sus bobinas en una determinada secuencia. El movimiento del eje del motor bipolar se produce por cada inversión de la polaridad en las bobinas. El sentido de giro está determinado por la secuencia que se aplique. La tabla 4.6, indica la secuencia necesaria para energizar las bobinas del motor bipolar. El ancho de duración del pulso depende del motor. Este dato se puede obtener de las hojas técnicas que el fabricante proporciona.

PASO	A	B	C	D
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla 4.6. Secuencia de energización de las bobinas para un motor bipolar.
Elaborado por: Los Autores.

Cambiando el sentido de energización de las bobinas se cambia el sentido de giro del motor bipolar.

PASO	D	C	B	A
1	+V	-V	+V	-V
2	+V	-V	-V	+V
3	-V	+V	-V	+V
4	-V	+V	+V	-V

Tabla 4.7. Secuencia de energización de las bobinas para un motor bipolar para invertir el giro del motor.
Elaborado por: Los Autores.

El diagrama de conexiones entre el motor bipolar el microcontrolador se presenta en la figura 4.10. El CI L293D, se utiliza para manejar la corriente de las bobinas del motor.

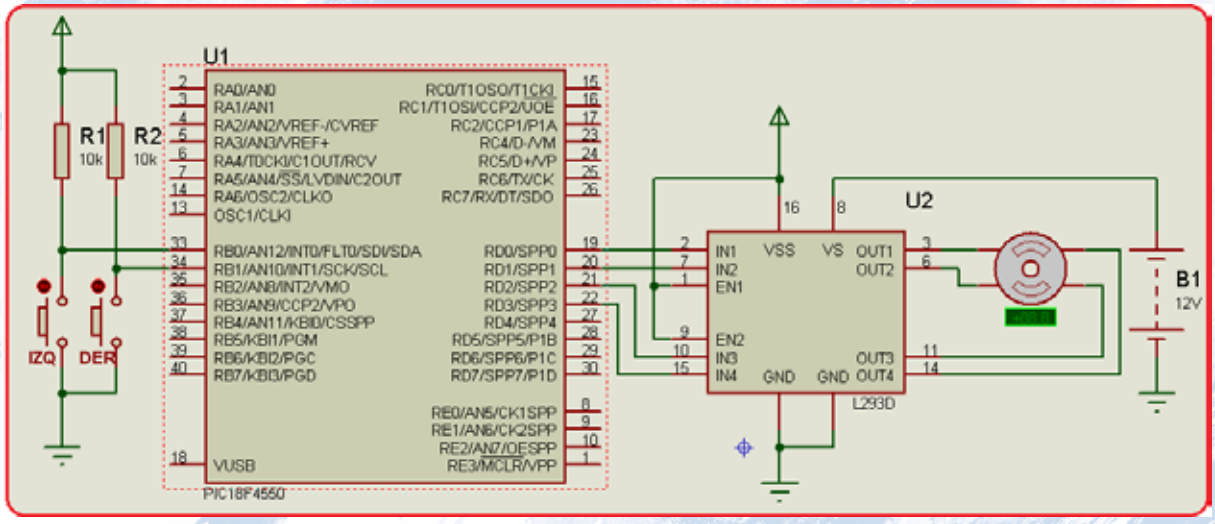


Figura 4.10. Circuito de control para control del motor bipolar.
Elaborado por: Los Autores.

Ejercicio 4.7. Mando de motor paso a paso bipolar.

PROGRAMA:

```
#include <18F4550.h>                    //Librería para usar el PIC18F4550.
#fuses INTRC, NOWRT, NOPUT, NOWDT, NOLVP, NOCPD, NOMCLR     //Configuración de fusibles.
#use delay (clock=4000000)      /       /F_OSC= 4 MHz.
#use standard_io(b)             /       /Usa la librería estándar B.
#use standard_io(d)             /       /Usa la librería estándar D.
#BYTE port_b= 0xF81             /       /Identifica al puerto B en la dirección 0xF1.
int i=1;        /                       /Controla el número de pulsos aplicados al motor.

void time(){
  delay_ms (300);               /       /Controla el tiempo del pulso.
  return;
}

void time1(){
  delay_ms (500);               /       /Controla el tiempo del pulso.
  return;
}

void main(){            /               /Función principal main.
        port_b= 0;              /       /Puerto B en 0.

 while(TRUE){           /               /Inicio bucle infinito.
            /                           /Giro a la izquierda.
  if (input(PIN_B0) ==0){               //……..
     for (i=0; i<=4;++i){
      output_low(PIN_D0);
      output_high(PIN_D1);
      output_high(PIN_D2);
      output_low(PIN_D3);
      time();
      output_low(PIN_D0);
      output_high(PIN_D1);
      output_low(PIN_D2);
      output_high(PIN_D3);
      time();
      output_high(PIN_D0);
      output_low(PIN_D1);
      output_low(PIN_D2);
      output_high(PIN_D3);
      time();
      output_high(PIN_D0);
      output_low(PIN_D1);
      output_high(PIN_D2);
      output_low(PIN_D3);
      time();
    }
  }
            /                                   /Giro a la derecha.
```

```
if (input(PIN_B1) ==0){
    for (i=0; i<=4;++i){
      output_high(PIN_D0);
      output_low(PIN_D1);
      output_high(PIN_D2);
      output_low(PIN_D3);
      time();
      output_high(PIN_D0);
      output_low(PIN_D1);
      output_low(PIN_D2);
      output_high(PIN_D3);
      time();
      output_low(PIN_D0);
      output_high(PIN_D1);
      output_low(PIN_D2);
      output_high(PIN_D3);
      time();
      output_low(PIN_D0);
      output_high(PIN_D1);
      output_high(PIN_D2);
      output_low(PIN_D3);
      time();
    }
  }
 }          /                    /Fin del bucle infinito.
}           /                    /Fin del bucle main.
```

Control de servomotor

Los servomotores son un tipo especial de motor de c.c. Están formados por un amplificador, un motor cc, un sistema reductor constituido por ruedas dentadas y un circuito de realimentación (control), todo en una misma caja de pequeñas dimensiones. La operación normal del servomotor es de 0 a 180º. El servomotor se posiciona de forma inmediata en cualquier posición dentro del rango de operación. La posición del servomotor se realiza por la aplicación de pulsos de una determinada duración al terminal de control. Por cada pulso el servomotor girará un cierto ángulo (depende de las características del servo).

Cada servomotor, dependiendo de la marca y modelo, tiene sus propios márgenes de operación. Por ejemplo, el servomotor Hitec HS 311, cuando se aplica un pulso de 1.5 ms se posiciona el eje a 90°, con 1 ms en 0° y 2 ms en 180°. Para el servomotor Futaba S3003, los valores posibles de la señal en alto están entre 0,3 y 2,1 ms, que posicionan al eje del servomotor en ambos extremos de giro (0° y 180°, respectivamente). El valor 1,2 ms envía al eje a la

posición central, y otros valores de duración del pulso dejarían al servomotor en la posición proporcional a dicha duración.

Para el caso analizado, se puede aplicar la relación $t = 0{,}3 + \theta/100$, donde t está dado en milisegundos y θ en grados.

Duración del nivel alto [ms]	Ángulo [grados]
0,3	0
1,2	90
2,1	180
0,75	45

Tabla 4.8. Valores usados en un servomotor Futaba S3003.
Fuente: Servomotor: http://www.ecured.cu/index.php/Servomotor

La frecuencia de aplicación de los pulsos para la mayoría de servomotores es de 50 Hz, hay ciertos modelos que trabajan con 40 Hz. La tabla 4.8, presenta un resumen de los tiempos del ancho del pulso para obtener ángulos de 0, 90 y 180º para diferentes marcas de servomotores, así como los colores característicos de los conductores.

FABRICANTE	DURACIÓN DEL PULSO (ms)			FRECUENCIA	COLOR DE LOS CABLES		
	MÍNIMA (0°)	NEUTRAL (90°)	MÁXIMA(180°)	(Hz)	POSITIVO	NEGATIVO	CONTROL
Futaba	0.9	1.5	2.1	50	Rojo	Negro	Blanco
Hitech	0.9	1.5	2.1	50	Rojo	Negro	Amarillo
Gruapner/Jr	0.8	1.5	2.1	50	Rojo	Negro	Naranja
Multiplex	1.05	1.6	2.15	40	Rojo	Negro	Amarillo
Robbe	0.65	1.3	1.95	50	Rojo	Negro	Blanco
Simprop	1.2	1.7	2.2	50	Rojo	Azul	Negro

Tabla 4.9. Ancho de pulsos para control de servomotores de diferentes marcas.
Fuente: Servomotor: http://www.ecured.cu/index.php/Servomotor

En la tabla 4.10, se muestra la relación de progreso entre el ancho del pulso y su correspondiente ángulo para un servomotor Futaba.

Tiempo (ms)	0,500	0,667	0,833	1,00	1,167	1,333	1,500	1,667	1,833	2,00	2,167	2,333	2,50
Ángulo	0	15	30	45	60	75	90	105	120	135	150	165	180

Tabla 4.10. Relación entre el ángulo del servomotor y el tiempo del pulso PWM.
Fuente: Servomotor: http://www.ecured.cu/index.php/Servomotor

Ejercicio 4.7. Control un servomotor con señal PWM.

Uno de los servomotores de mayor uso es el HITEC HS-311, el cual para posicionarse en un extremo (0^O) necesita de pulsos de 1 ms y para el otro extremo (180^O) pulsos de 2 ms.

Se utiliza dos pulsadores para ordenar al microcontrolador que genere las señales de 1 y 2 ms, respectivamente.

Dado que el servomotor necesita pulsos cada 50 Hz, el periodo del T_{PWM} es:

T_{PWM} = 1/50 Hz= 20 ms.

EL valor del PR2 es:

PR2= T_{PWM}/ (T_{OSC}*4*T2_DIV) -1

Debido a que el T_{PWM} a generar es alto, se utilizará el oscilador interno de 500kHz del PIC.

T_{OSC}= 1/500kHz= 0.002 ms.

PR2= 20ms/(0.002ms*4*16) - 1 = 155.25, tomaremos

PR2= 155.

Para 1 ms el duty cycle es el 5% y para 2 ms el 10%, por tanto, el valor de DC es;

value= 4(PR2 + 1)*DC

value= 4(156)*0.5= 31.2

value= 31 para 1ms y

value= 63 para 2 ms

Para aumentar el ancho del ancho pulso a partir de value=31 incrementamos la variable hasta 63 y el servomotor cambia de posición y desde este valor, se decrementa para retornar a la posición inicial. El pulsador INC sirve para incrementar el ancho del pulso y el pulsador DEC decrementa el ancho del pulso en una unidad respectivamente. El circuito utilizado para el control del servomotor se observa en la figura 4.11.

PROGRAMA:

```
#include <18F4550.h>       /                    /Librería para usar el PIC18F4550.
#fuses  NOWRT, NOPUT, NOWDT, NOLVP, NOCPD  /        /Configuración de fusibles.
#use delay(internal=500KHZ)                  //F_OSC = 500 kHz.
long value = 31;    /                        /Valor del DC 5% o sea 1ms.

void main(){          /                      /Función principal main.
setup_ccp1(CCP_PWM);   /                     /Fija el módulo CCP1 como PWM.
setup_timer_2(T2_DIV_BY_16, 155, 1); /       /Fija el TMR2.

 while(TRUE) {      /                        /Bucle infinito.
  if (input(PIN_B0) == 0 && value<=63){          //Al accionar el pulsador del puerto B0…
    set_pwm1_duty (value);                    //5% duty cycle inicial.
    value++;         /                        /incrementa value hasta 63 que corresponde a 2ms.
    delay_ms (200);          /               /el incremento de value es cada 200 ms.
   }

  if (input(PIN_B1) == 0 && value>=31){          //Al accionar el pulsador del puerto B1…
    set_pwm1_duty (value);
    value--;          /                       /decrementa value hasta 31 que corresponde a 1ms.
    delay_ms (200);          /               /el incremento de value es cada 200 ms.
   }
  }          /                                /Fin del bucle infinito.
}            /                                /Fin del main.
```

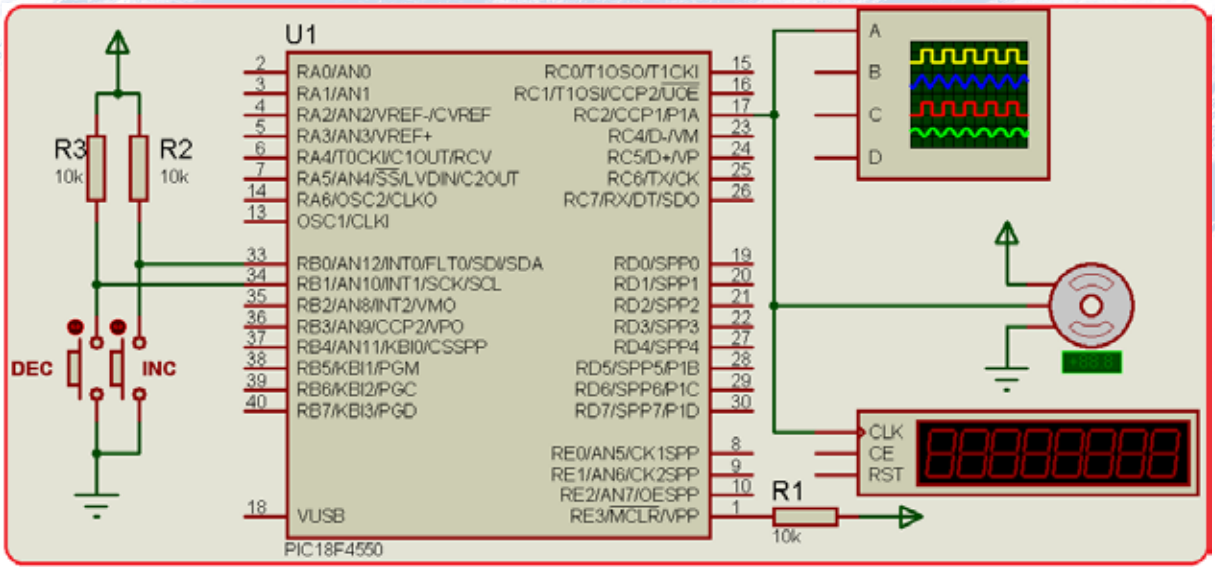


Figura 4.11. Circuito para generar y comprobar las señales PWM para el servomotor.
Fuente: Servomotor: http://www.ecured.cu/index.php/Servomotor

EJERCICIOS PROPUESTOS

Se tiene 3 pulsadores P1, P2 y P3, para accionar a un motor DC. Al accionar P1 el motor funciona al 100% de su velocidad nominal, al accionar el pulsador P2, el motor funciona al 75% de su velocidad nominal y al pulsar P3 el motor funciona al 25% de su velocidad nominal. Si el motor arranca al 25%, deberá primero pasar por el 75% y luego al 100%, de igual forma si el motor arranca al 100%, el motor deberá primero pasar por el 75% y luego al 25%. Realice el programa para que cumpla con lo indicado y utilice señales PWM para el control.

Al accionar un pulsador P1 un motor PAP unipolar inicia el giro en sentido horario y queda funcionando indefinidamente. Se dispone de un pulsador P2, para apagar el motor.

Realice un programa para control de giro de un motor PAP bipolar que funciona de la siguiente forma:

- El motor puede girar en forma indefinida en cualquier sentido siempre que el motor esté apagado, para esto se dispone de dos pulsadores P1 y P2.
- Se dispone de un pulsador P3 para parar el motor.

Busque las características técnicas del servo HITEC HS-322HD y diseñe el circuito y el programa para controlar el servo utilizando una señal PWM generada por el microcontrolador. Utilice los componentes externos que sean necesarios.

BIBLIOGRAFÍA

1. Eduardo García Breijo. Compilador C CSS y Simulador PROTEUS para Microcontroladores PIC. MARCOMBO, S.A. Barcelona - España. 2008.

2. PIC18F2455/2550/4455/4550 Data Sheet. Microchip Technology INC.: Disponible en: www.microchip.com.

3. Home Automation Using the PIC16F877A. Microchip Technology INC.: Disponible en: www.microchip.com.

4. CCS C Compiler Manual PCD. Disponible en: https://www.ccsinfo.com/downloads/ccs_c_manual.pdf

5. Software CCS. Disponible en: http://www.ccsinfo.com/ccsfreedemo.php.

6. Manual de usuario del Compilador PCW de CCS. http://www.cursos.ucv.cl/eie48700/referencias/CCS_C_Manual.pdf

7. Memorias I2C. 64K I2C™ Serial EEPROM. Disponible en Internet: http://ww1.microchip.com/downloads/en/DeviceDoc/21189f.pdf

8. L293x Quadruple Half-H Drivers. Texas Instruments. Disponible en Internet: http://www.ti.com/lit/ds/symlink/l293.pdf

9. ULN2803A Darlington Transistor Arrays. Texas Instruments. Disponible en Internet: http://www.ti.com/lit/ds/symlink/uln2803a.pdf

10. MAX232x Dual EIA-232 Drivers/Receivers. Texas Instruments. Disponible en Internet: http://www.ti.com/lit/ds/symlink/max232.pdf

11. Electrónica, Microcontroladores y Psicología. Melodías con PIC. Disponible en Internet: https://picmind.es.tl/Melod%EDas-con-PIC.htm

12. Profesor Rangel. Microcontroladores PIC Gama Alta. Disponible en: http://profesor-rangel.blogspot.com/p/manejo-de-interrupciones.html

13. Tutoriales Programación PIC en C: http://www.aquihayapuntes.com/

14. Aprendiendo Electrónica. Comunicación Serial. Disponible en Internet: http://aprendiendoelectronicafacil.blogspot.com

15. El Rincón de CCS C: Proyectos y experimentos: http://picmania.garcia-cuervo.net/

16. Simposio Argentino de Sistemas Embebidos: http://www.sase.com.ar/

17. Servomotores:
http://www.info-ab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm

18. Eduardo Carletti. Servos, características básicas.
http://robots-argentina.com.ar/MotorServo_basico.htm

19. Eduardo Carletti. Motores Paso a Paso, características básicas. http://robots-argentina.com.ar/MotorPP_basico.htm

20. Servomotor: http://www.ecured.cu/index.php/Servomotor

21. Foros de electrónica de diferentes tópicos: http://www.todopic.com.ar/foros/index.php

22. Foros, proyectos, tutoriales: http://www.ucontrol.com.ar/

23. Comunicación serial, proyectos, componentes: http://robotypic.blogspot.com/

24. PICs, Electrónica y Robótica: http://picrobot.blogspot.com/

25. Protocolo USB. Disponible en Internet en: http://www.ing.unlp.edu.ar/electrotecnia/cdm/9%20Protocolo%20USB.pdf

Publicaciones Científicas

ISBN: 978-9942-765-37-6

ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA