2011

Universidad Nacional de Ingeniería

Luis Alberto Vargas Tijerino

MANUAL DE LABORATORIO PARA ELECTRONICA DIGITAL 2 EN BASE A PIC, CCS Y PROTEUS

En este manual se describe la realización de 6 prácticas de laboratorio.

Contenido

Laboratorio 1: Introducción a CCS y Proteus.

1.1 Objetivo.

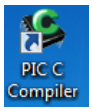
Familiarizarse con el compilador CCS y el simulador de circuitos Proteus mediante la implementación práctica del ejemplo Parpadeo de un LED.

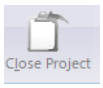
1.2 Introducción.

A continuación se mostrarán los pasos necesarios para crear un nuevo proyecto en CCS y luego se mostrarán los pasos necesarios para simular nuestro proyecto en Proteus.

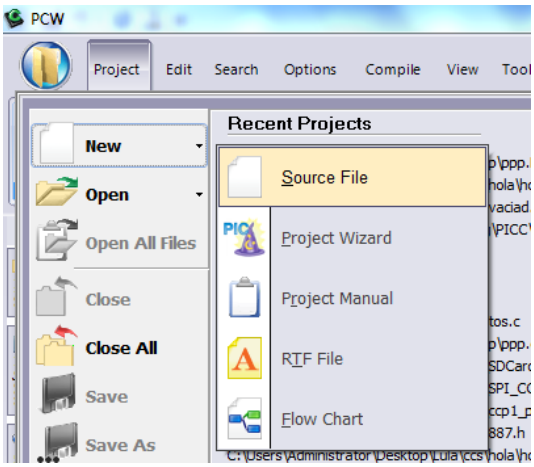
1.3 Procedimiento.

1.3.1 Crear el Proyecto en CCS

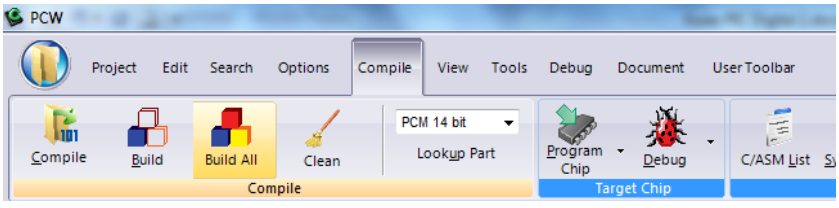
1. Abrir el Compilador PIC C de CCS 

2. Presionar el botón **close Project** 

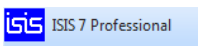
para cerrar todos los archivos anteriormente abiertos.
3. Dar click en el botón **inicio**. 
4. Crear un nuevo archivo dando click en **New → Source File** a como se muestra en la siguiente figura



5. Crear una Nueva carpeta llamada Lab1 en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
6. Abrir la carpeta Lab1 y guardar el proyecto con el nombre de *Led_Blink1*.
7. Copiar el **Código Led_Blink1.¡Error! No se encuentra el origen de la referencia.**
8. Dar clic en el botón Build All de la pestaña Compile o presionar la tecla F9.

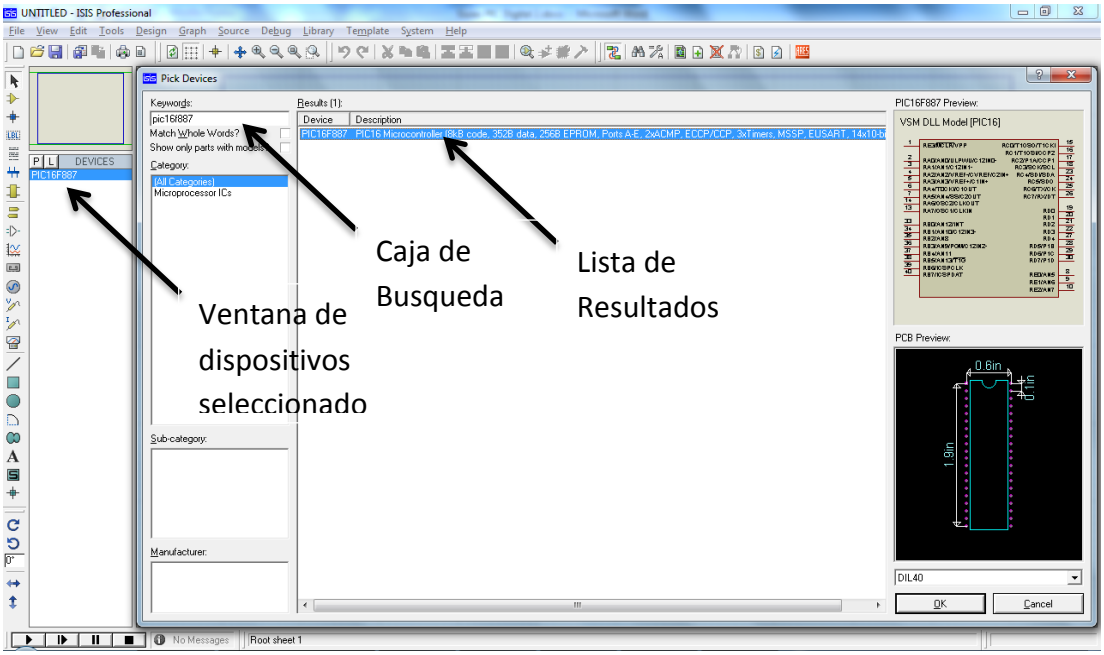


1.3.2 Crear el circuito en Proteus

1. Abrir el simulador ISIS 7 de Proteus 

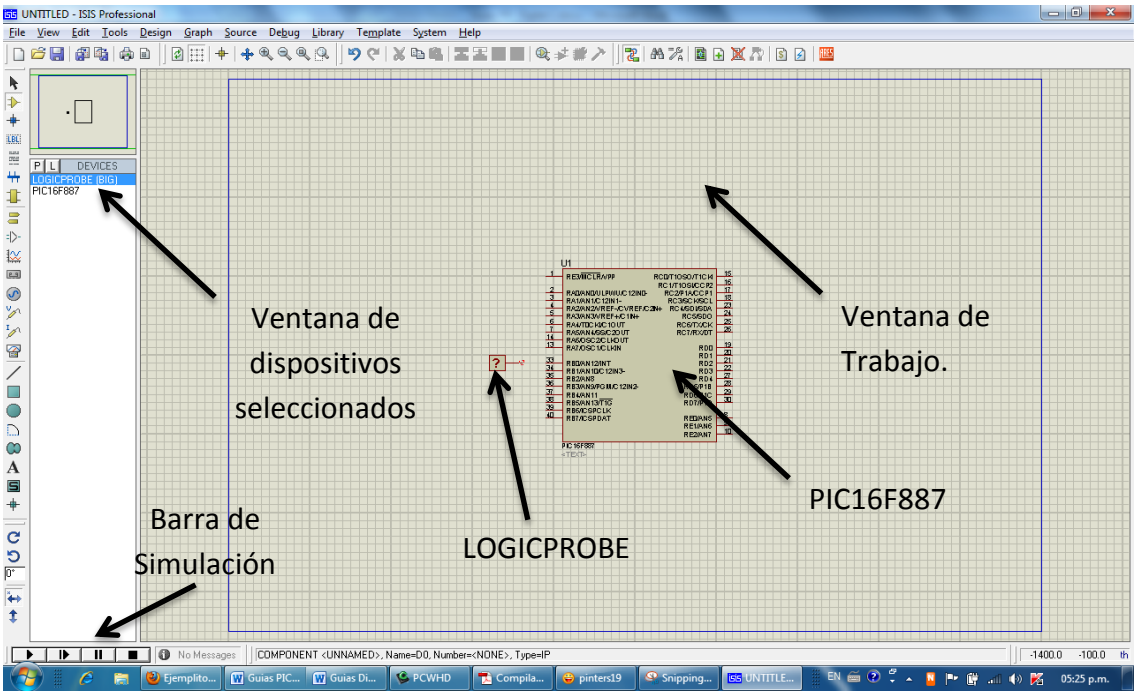
.

2. Presionar la tecla “P” del teclado.
3. Escribir PIC16F887 en la caja de búsqueda y darle doble clic al dispositivo en la lista de resultados para que aparezca en la ventana de dispositivos seleccionados.



4. Escribir LOGICPROBE en la caja de búsqueda y darle doble clic al dispositivo en la lista de resultados para que aparezca en la ventana de dispositivos seleccionados.
5. Escribir LOGICTOGGLE en la caja de búsqueda y darle doble clic al dispositivo en la lista de resultados para que aparezca en la ventana de dispositivos seleccionados.
6. Cerrar la ventana de Pick Devices.
7. En la ventana de dispositivos seleccionados darle un clic al dispositivo y luego dar un clic en la ventana de trabajo para que aparezca el dispositivo y finalmente ubicar los dispositivos como aparece en la siguiente figura.
    a. Si se desea rotar el dispositivo en sentido horario se debe presionar la tecla “+” del teclado numérico.
    b. Si se desea rotar el dispositivo en sentido anti-horario presionar la tecla “-“ del teclado numérico.

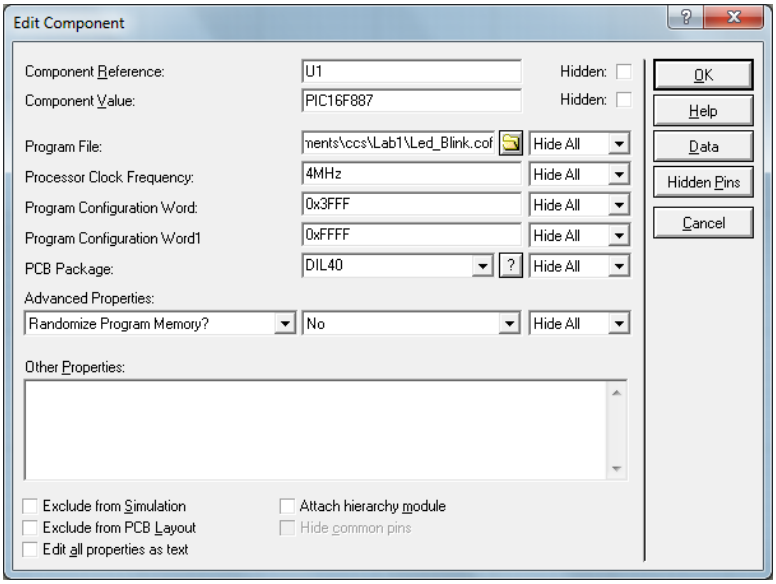
c. Si se desea rotar el dispositivo en espejo se debe presionar la combinación de teclas “ctrl+m”.



8. Conectar el Pin del LOGICPROBE con el pin RB0 del PIC16F887.

1.3.3 Cargar el Archivo fuente

1. Darle doble clic al PIC16F887 ubicado en la ventana de trabajo
2. Darle clic al botón de la caja de texto Program File y seleccionar el archivo fuente Led_Blink1.cof
3. Cambiar la frecuencia de reloj a 8MHz.

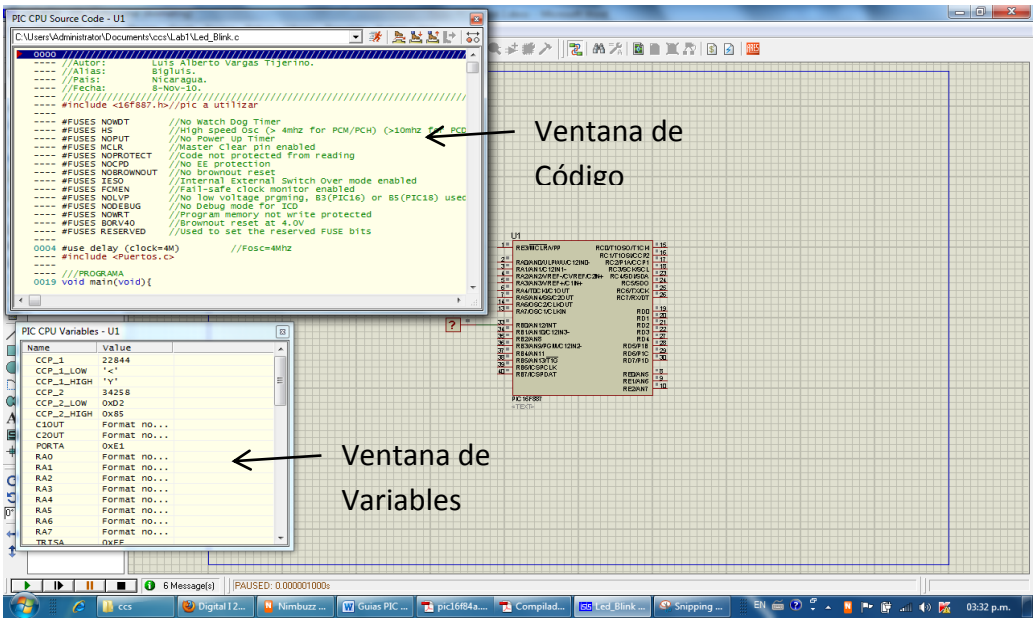


4. Luego presionar el botón Ok.

1.3.4 Realizar la simulación

1. Dar clic en el botón Play  de la barra de simulación  y observar la simulación.
2. Dar clic en el botón stop  de la barra de simulación.
3. Dar clic en el botón step  de la barra de simulación para iniciar la simulación paso a paso o presionar la combinación de teclas Ctrl+F12.

4. A continuación se mostraran la ventana del código fuente y la ventana de variables.



5. En la ventana de Código aparecen 4 botones  .

   a. **Step over**  **o tecla F10:** La simulación se realiza paso a paso pero al encontrar la llamada a una función el depurador la realiza toda la función de una sola vez sin entrar a ella.

   b. **Step into**  **o tecla F11:** La simulación se realiza paso a paso y al encontrar la llamada a una función el depurador la realiza entra a la función realizándola paso a paso.

   c. **Step out**  **o combinación Ctrl+F11:** Si el depurador se encuentra en el código interno de una función al presionar este botón saldrá del código interno de la función y se mostrará la línea siguiente a la llamada a la función.

   d. **Run to Source Line**  **o combinación Ctrl+F10:** Ubica al depurador en la línea seleccionada por el usuario.

6. Una vez que se presione uno de los botones anteriores se puede observar el cambio en la ventana de código y en la ventana de variables.

1.4 Códigos.

1.4.1 Código Led_Blink1.

El código siguiente sirve para hacer parpadear el LED conectado al PIN_B0 cada 0.5 segundos utilizando las funciones propias de CCS para manejar los Puertos.

```
////////////////////////////////////////////////////////////////////////////
//Autor: Luis Alberto Vargas Tijerino.
//Alias: Bigluis.
//Pais:       Nicaragua.
//Fecha: 8-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT        //No Watch Dog Timer
#FUSES HS           //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT        //No Power Up Timer
#FUSES MCLR         //Master Clear pin enabled
#FUSES NOPROTECT    //Code not protected from reading
```

```
#FUSES NOCPD       //No EE protection
#FUSES NOBROWNOUT  //No brownout reset
#FUSES IESO        //Internal External Switch Over mode enabled
#FUSES FCMEN       //Fail-safe clock monitor enabled
#FUSES NOLVP       //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG     //No Debug mode for ICD
#FUSES NOWRT       //Program memory not write protected
#FUSES BORV40      //Brownout reset at 4.0V
#FUSES RESERVED    //Used to set the reserved FUSE bits

#use delay (clock=8M)         //Fosc=8Mhz

///PROGRAMA
void main(void){
   setup_adc_ports(NO_ANALOGS|VSS_VDD); //Todas las entradas del PIC Seran Digitales.
   setup_comparator(NC_NC_NC_NC);       //Los comparadores Analogicos estaran apagados.
   while(TRUE){           //bucle infinito
      output_low(PIN_B0);               //led off
      delay_ms(500);  //Retardo de 500ms = 0.5s
      output_high(PIN_B0);              //led on
      delay_ms(500);  //Retardo de 500ms = 0.5s
   }
}
```

Laboratorio 2: Puertos de Entrada y Salida.

2.1 Objetivo.

Familiarizarse con los Puertos de Entrada y Salida mediante la implementación práctica del ejemplo *LED y Botones* y el ejemplo *Pantalla LCD.*

2.2 Trabajo Previo.

Contestar las siguientes Preguntas:

1. ¿Cuantos puertos de entrada tiene el PIC16F887?
2. ¿Cuáles son los pines necesarios para conectar el LCD?
3.

2.3 Introducción.

En casi todos los proyectos es necesario leer alguna entrada de tipo digital conectada a pulsadores, interruptores, sensores digitales o similares; también es necesario escribir datos por medio de una salida de tipo digital conectada a LED, pantallas LCD, display de siete segmentos o similares. Este laboratorio trata de explicar cómo realizar la configuración de los puertos del PIC para utilizarlos como entradas o salidas digitales.

2.4 Puertos de Entradas/Salidas.

Los Microcontroladores *PIC* tienen terminales de entrada/salida (**I/O**, *Input/Output*) divididos en puertos, que se encuentras nombrados alfabéticamente A, B, C, D, etc. Cada puerto puede tener hasta 8 terminales que se pueden comportar como una I/O digital.

El PIC16F887 tiene hasta 35 I/O digitales de propósito general, y en dependencia de la configuración de sus periféricos pueden estar disponibles como I/O de propósito general. Por lo tanto, si un periférico es habilitado, el pin asociado no será utilizado como I/O de propósito general.

2.4.1 Lectura del puerto.

La siguiente función configura el puerto como entrada y realiza su lectura:

input_x( )

Sintaxis	**value = input_x()**
Parametros	***value*** es el valor booleano que se lee de la entrada ***x***.

2.4.2 Lectura del pin de Entrada

La siguiente función configura el PIN como entrada y realiza su lectura:

input( )

Sintaxis	**value = input(*pin*)**
Parametros	***value*** es el valor que se lee del ***pin x***.

2.4.3 Escritura del puerto.

La siguiente función configura el puerto como salida y realiza su escritura:

output_x( )

Sintaxis	**output_a (*value*)**
Parametros	***value*** es el valor de 8 bits que se escribe en la salida ***x***.

2.4.4 Escritura de un pin.

Las siguientes funciones configuran el pin como salida y realizan su escritura:

output_low( ), **output_high( )** y **output_bit( )**

Sintaxis	**output_low(*pin*)**	Pone a 0 el pin
	output_high(*pin*)	Pone a 1 el pin
	output_bit(*pin,value*)	Le da el valor de **value** al **pin**.

2.4.5 Configuración rápida de los pines I/O.

Una configuración rápida genera código más eficiente, ya que el compilador asume que los pines de I/O serán cambiados solo si es especificado por el usuario. A continuación se muestra la función para decidir si los pines de un puerto son de entrada o salida:

set_tris_x( )

Sintaxis	**set_tris_a(*value*)**
Parametros	***value*** es un valor de 8 bits, cada bit determina si el pin correspondiente es de entrada o salida. Por ejemplo: si el bit 0 es igual a 0 entonces el pin 0 será salida y si el bit 4 es 1 entonces el bit 4 será una entrada.

Al utilizar la función anterior siempre es necesario utilizar la siguiente directiva:

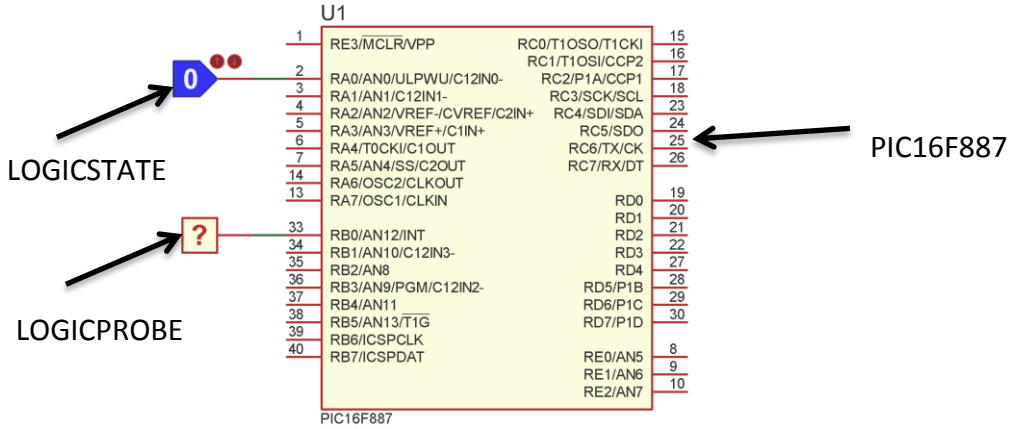
#USE FAST_IO

Sintaxis	**#use fast_io (*port*)**	
Parametros	***port*** puede ser:	
	• A	solo el puerto A
	• B	solo el puerto B
	• C	solo el puerto C
	• D	solo el puerto D
	• E	solo el puerto E
	• F	solo el puerto F
	• G	solo el puerto G
	• H	solo el puerto H
	• I	solo el puerto I
	• J	solo el puerto J
	• ALL.	Todos los puertos

2.5 Procedimiento.

2.5.1 LED y Botones.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**
2. Crear una Nueva carpeta llamada Lab2 y una subcarpeta llamada LED_Botones en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
3. Abrir la carpeta LED_Botones y guardar el proyecto con el nombre de *LED_Botones*.
4. Copiar el **Código LED_Botones**.
5. Compilar el proyecto.
6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana PICK DEVICE el PIC16F887, el LOGICSTATE y el LOGICPROBE.

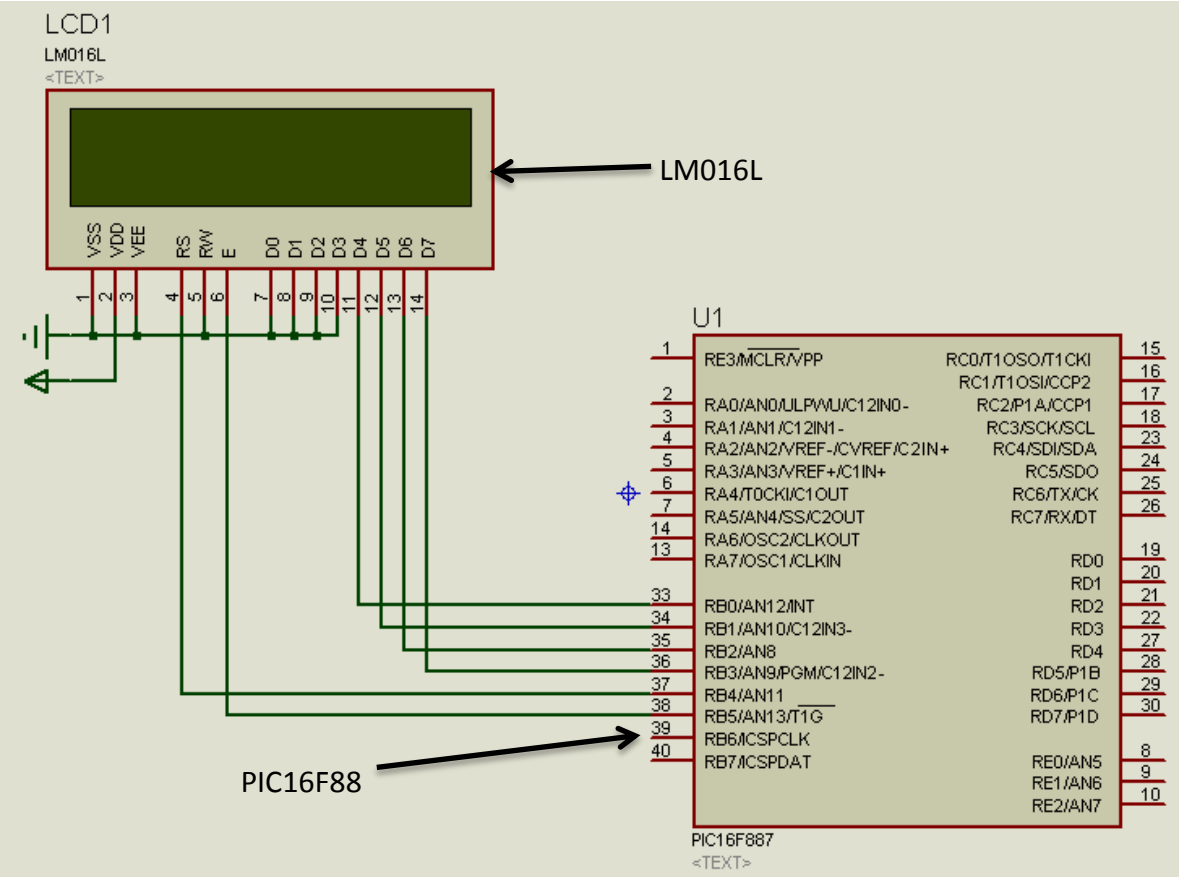


7. **Cargar el Archivo fuente** *Led_Botones.cof* y **Realizar la simulación.**

2.5.2 Pantalla LCD.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**
2. Crear una Nueva carpeta llamada LCD_HolaMundo dentro de la carpeta Lab2.
3. Abrir la carpeta LCD_HolaMundo y guardar el proyecto con el nombre de *LCD_HolaMundo*.
4. Copiar el **Código LCD_HolaMundo.**
5. Compilar el proyecto.

6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana PICK DEVICE el *PIC16F887* y el *LM016L*.



7. **Cargar el Archivo fuente** *LCD_HolaMundo.cof* y **Realizar la simulación.**

2.6 Códigos.

2.6.1 Código LED_Botones.

A continuación se muestra un código que hace que el LED conectado al pin RB0 se encienda cuando se presiones el botón conectado a RA0.

```
////////////////////////////////////////////////////////////////////////////
//Autor: Luis Alberto Vargas Tijerino.
//Alias: Bigluis.
//Pais:       Nicaragua.
//Fecha: 8-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT        //No Watch Dog Timer
#FUSES HS           //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT        //No Power Up Timer
#FUSES MCLR         //Master Clear pin enabled
#FUSES NOPROTECT    //Code not protected from reading
#FUSES NOCPD        //No EE protection
#FUSES NOBROWNOUT   //No brownout reset
#FUSES IESO         //Internal External Switch Over mode enabled
#FUSES FCMEN        //Fail-safe clock monitor enabled
#FUSES NOLVP        //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG      //No Debug mode for ICD
#FUSES NOWRT        //Program memory not write protected
#FUSES BORV40       //Brownout reset at 4.0V
#FUSES RESERVED     //Used to set the reserved FUSE bits

#use delay (clock=4M)          //Fosc=4Mhz
#include <Puertos.c>
```

```
///PROGRAMA
void main(void){
   TRISB=0;  //Todos los pines de PORB como salida
   TRISA=255;
   setup adc ports(NO ANALOGS|VSS VDD); //Todas las entradas del PIC Seran Digitales.
   setup_comparator(NC_NC_NC_NC);       //Los comparadores Analogicos estaran apagados.
   while(TRUE){       //bucle infinito
      if(RA0)         //SI se activa el boton conectado a RA0.
         RB0=1;       //El led conectado a RB0 se activara.
      else            //De lo contrario.
         RB0=0;       //El led conectado a RB0 se apagara.
   }
}
```

2.6.2 Código LCD_HolaMundo.

La función del código siguiente es hacer que el PIC muestre la cadena “Hola Mundo” y luego de 1 segundo mostrara en la siguiente línea “UNI FEC” mediante la pantalla LCD de 16x2 caracteres.

```
////////////////////////////////////////////////////////////////////////////
//Autor:   Luis Alberto Vargas Tijerino.
//Alias:   Bigluis.
//Pais:    Nicaragua.
//Fecha:   12-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT       //No Watch Dog Timer
#FUSES HS          //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT       //No Power Up Timer
#FUSES MCLR        //Master Clear pin enabled
#FUSES NOPROTECT   //Code not protected from reading
#FUSES NOCPD       //No EE protection
#FUSES NOBROWNOUT  //No brownout reset
#FUSES IESO        //Internal External Switch Over mode enabled
#FUSES FCMEN       //Fail-safe clock monitor enabled
#FUSES NOLVP       //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG     //No Debug mode for ICD
#FUSES NOWRT       //Program memory not write protected
#FUSES BORV40      //Brownout reset at 4.0V
#FUSES RESERVED    //Used to set the reserved FUSE bits

#use delay (clock=8M)         //Fosc=4Mhz

//Definir Pines de conexion al LCD
#define LCD ENABLE PIN  PIN B5
#define LCD_RS_PIN      PIN_B4
#define LCD_RW_PIN      PIN_B6
#define LCD_DATA4       PIN_B0
#define LCD_DATA5       PIN_B1
#define LCD DATA6       PIN B2
#define LCD_DATA7       PIN_B3

#include <lcd.c>  //Llamada a la libreria lcd.c

void main(){
   setup adc ports(NO ANALOGS|VSS VDD);   //Todas las entradas del PIC Seran Digitales.
   setup comparator(NC NC NC NC);         //Los comparadores Analogicos estaran apagados.
   lcd_init();                            //Inicializamos el LCD.
   lcd_putc("Hola Mundo");                //Se escribe en el LCD la cadena Hola mundo.
   delay ms(1000);
   lcd_putc("\nUNI FEC");
}
```

Laboratorio 3: Convertidor Analógico a Digital.

3.1 Objetivo.

Familiarizarse con el módulo de conversión analógico digital mediante la implementación práctica del ejemplo *Leer_ADC.*

3.2 Trabajo Previo.

Contestar las siguientes Preguntas:

1. ¿Cuántas entradas analógicas tiene el PIC16F887?
2. ¿De cuántos bits es el módulo ADC interno del PIC16F887 y cómo podemos modificarlos?
3. ¿Qué pines del PIC podemos utilizar como Voltajes de referencia?
4. ¿Cuánto es el Tiempo de adquisición del PIC16F887?
5. ¿Cuánto es el Tiempo de conversión AD del PIC16F887?
6. Si en la entrada del convertidor analógico se le conecta una fuente de 1.5 volts cual es el valor que se obtendrá en el momento de la lectura.

3.3 Introducción.

Existe una gran cantidad de proyectos en las que es necesario que nuestros dispositivos digitales realicen operaciones de control de variables analógicas tales como temperatura, presión, flujo, caudal, peso, distancia, velocidad o similares. Este laboratorio trata de explicar cómo realizar la configuración del módulo ADC del PIC para realizar las mediciones antes mencionadas.

3.4 Conversor Analógico Digital.

Un **conversor analógico-digital** (**ADC**, *Analog-to-Digital Converter*) es un dispositivo electrónico capaz de convertir una entrada analógica de voltaje en un valor binario, Se utiliza en equipos electrónicos como ordenadores, grabadores de sonido y de vídeo, y equipos de telecomunicaciones. La señal analógica, que varía de forma continua en el tiempo, se conecta a la entrada del dispositivo y se somete a un muestreo a una velocidad fija, obteniéndose así una señal digital a la salida del mismo. [Wikipedia]

El convertidor Analógico-digital del PIC16F887 permite la conversión de una señal de entrada analógica a un valor en representación binaria de 10 bits o de 8 bits. Este dispositivo usa entradas analógicas que son multiplexadas en un circuito de muestreo y retención (*Sample and Hold*). La salida del muestreo y retención es conectada a la entrada del convertidor. El convertidor genera un resultado binario de 10 bits o de 8 bits mediante aproximaciones sucesivas y almacena el resultado de la conversión los registros de resultado de ADC. Las figuras siguientes muestran el diagrama en bloques del módulo ADC y el modelo de la Entrada Analógica.

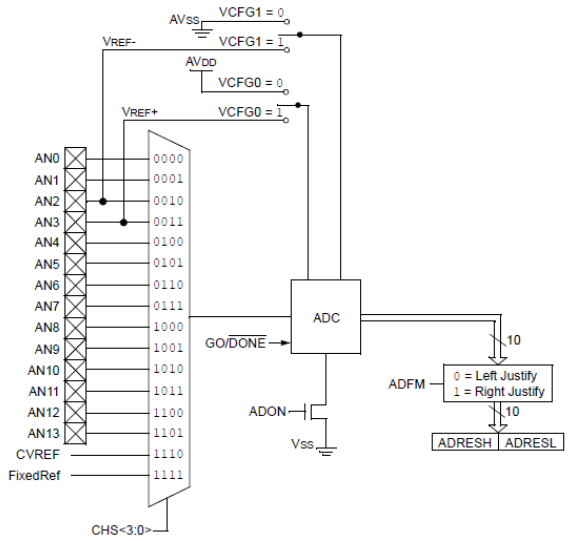
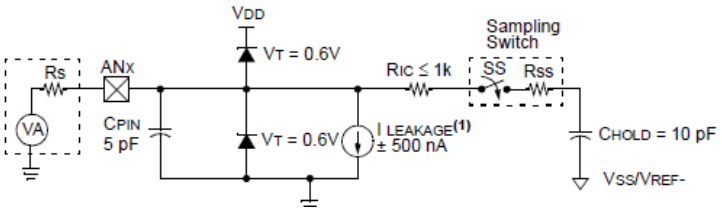
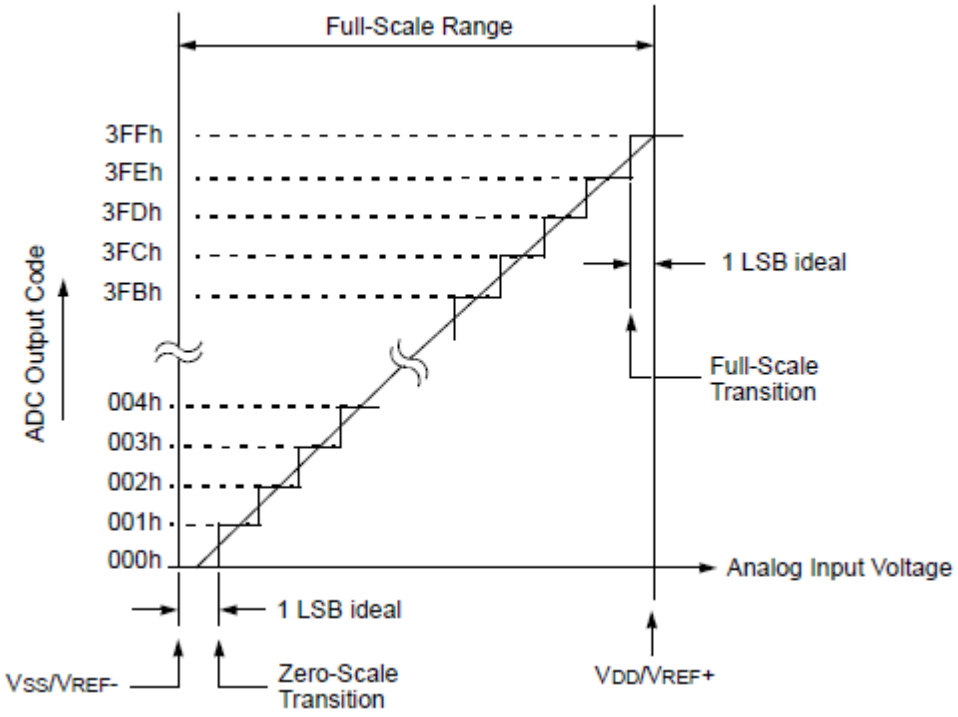


Diagrama en Bloque del ADC



Modelo de una Entrada Analógica

A continuación se muestra la función de transferencia del ADC.



3.4.1 Formato del resultado de la conversión.

Para cambiar el formato del resultado de la conversión se escribe arriba de todas las funciones lo siguiente:

#device adc=10

Donde el valor que se le asigna a *ADC* puede ser 8 o 10.

3.4.2 Configuración de los puertos ADC.

Para realizar la configuración de los puertos ADC se utiliza la siguiente función:

setup_adc_ports(*value***)**

Sintaxis	setup_adc_ports(***value***)
Parametros	***value*** podría ser una o más de las siguientes constantes: • sAN0 Selecciona AN0 analógica • sAN1 Selecciona AN1 analógica • sAN2 Selecciona AN2 analógica • sAN3 Selecciona AN3 analógica • sAN4 Selecciona AN4 analógica • sAN5 Selecciona AN5 analógica • sAN6 Selecciona AN6 analógica • sAN7 Selecciona AN7 analógica • sAN8 Selecciona AN8 analógica • sAN9 Selecciona AN9 analógica • sAN10 Selecciona AN10 analógica • sAN11 Selecciona AN11 analógica • sAN12 Selecciona AN12 analógica • sAN13 Selecciona AN13 analógica • NO_ANALOGS Selecciona ninguna analógica • ALL_ANALOG Selecciona todas analógica • VSS_VDD Seleccionamos Vss como Vref- y Vdd como Vref+ • VSS_VREF Seleccionamos Vss como Vref- y Vref+ como Vref+ • VREF_VREF Seleccionamos Vref- como Vref- y Vref+ como Vref+ • VREF_VDD Seleccionamos Vref- como Vref- y Vdd como Vref+
Ejemplo	`setup_adc_ports(sAN0|sAN1|VSS_VDD);` `//Seleccionamos AN0 y AN1 como analógicas, Vss y Vdd como Vref.`

3.4.3 Configuración del ADC.

Para realizar la configuración del ADC se utiliza la siguiente función:

setup_adc()

Sintaxis	setup_adc(***value***)
Parametros	***mode*** podría ser una o más de las siguientes constantes: • ADC_OFF Apagamos el ADC. • ADC_CLOCK_DIV_2 Utiliza Fosc/2 (ej. Fosc=1MHz) • ADC_CLOCK_DIV_8 Utiliza Fosc/8 (ej. Fosc=1MHz) • ADC_CLOCK_DIV_32 Utiliza Fosc/32 (ej. Fosc=20MHz y 8MHz) • ADC_CLOCK_INTERNAL Utiliza el reloj RC interno.
Ejemplo	`setup_adc_ports(sAN0|sAN1|VSS_VDD);` `//Seleccionamos AN0 y AN1 como analógicas, Vss y Vdd como Vref.`

3.4.4 Lectura del Valor de la conversión ADC.

Para realizar la lectura del valor de la conversión ADC se utiliza la siguiente función:

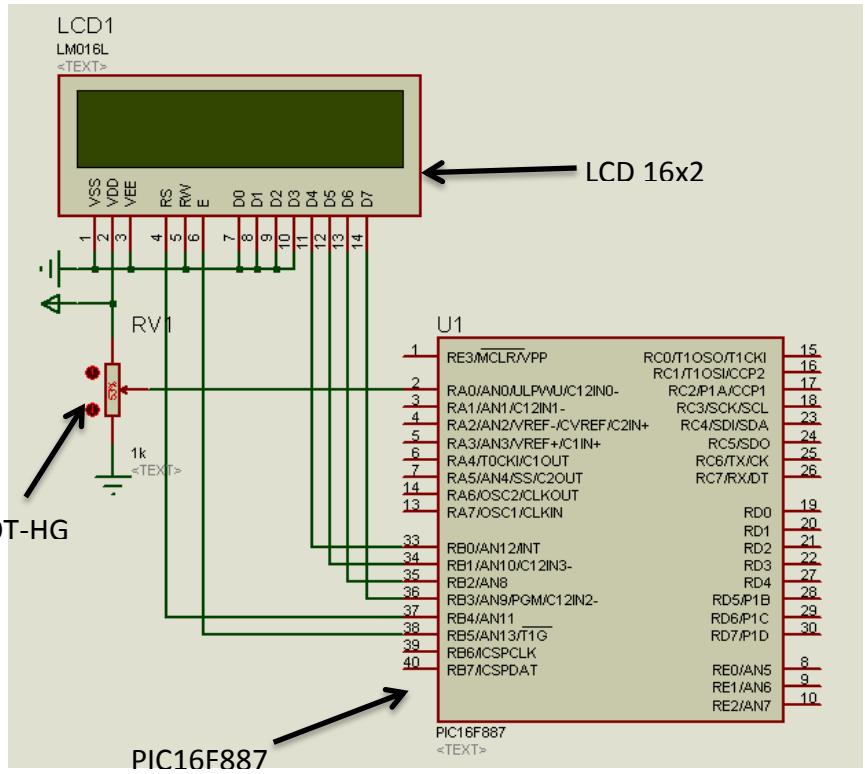
read_adc( )

Sintaxis	value = read_adc ([***mode***])
Parametros	***mode*** es un parámetro opcional. Si es usado puede ser uno de los siguientes valores: • ADC_START_AND_READ Toma lecturas continuamente. Este está por defecto.

	• ADC_START_ONLY	Inicia la conversión y no retorna valor.
	• ADC_READ_ONLY	Lee el último valor de conversión
Ejemplo	`setup_adc_ports(sAN0|sAN1|VSS_VDD);` `//Seleccionamos AN0 y AN1 como analógicas, Vss y Vdd como Vref.`	

3.5 Procedimiento.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**
2. Crear una Nueva carpeta llamada Lab3 en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
3. Abrir la carpeta Lab3 y guardar el proyecto con el nombre de *Leer_ADC*.
4. Copiar el Código LED_Botones. **Código Leer_ADC**.
5. Compilar el proyecto.
6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana *PICK DEVICE* el PIC16F887, el LCD 16x2 y el POT-HG.



7. **Cargar el Archivo fuente** *Leer_ADC.cof* y **Realizar la simulación.**

3.6 Códigos.

3.6.1 Código Leer_ADC.

```
////////////////////////////////////////////////////////////////////////////
//Autor:   Luis Alberto Vargas Tijerino.
//Alias:   Bigluis.
//Pais:    Nicaragua.
//Fecha:   16-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#device adc=10

#FUSES NOWDT       //No Watch Dog Timer
```

```
#FUSES HS          //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT       //No Power Up Timer
#FUSES NOMCLR      //Master Clear pin enabled
#FUSES NOPROTECT   //Code not protected from reading
#FUSES NOCPD       //No EE protection
#FUSES NOBROWNOUT  //No brownout reset
#FUSES IESO        //Internal External Switch Over mode enabled
#FUSES FCMEN       //Fail-safe clock monitor enabled
#FUSES NOLVP       //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG     //No Debug mode for ICD
#FUSES NOWRT       //Program memory not write protected
#FUSES BORV40      //Brownout reset at 4.0V
#FUSES RESERVED    //Used to set the reserved FUSE bits

#use delay (clock=8M)         //Fosc=8Mhz

//Definir Pines de conexion al LCD
#define LCD_ENABLE_PIN  PIN_B5
#define LCD_RS_PIN      PIN_B4
#define LCD_RW_PIN      PIN_B6
#define LCD_DATA4       PIN_B0
#define LCD_DATA5       PIN_B1
#define LCD_DATA6       PIN_B2
#define LCD_DATA7       PIN_B3

#include <lcd.c>  //Llamada a la libreria lcd.c

///PROGRAMA
void main(void){
   long val;
   setup_comparator(NC_NC_NC_NC);   //Los comparadores Analogicos estaran apagados.
   setup_adc_ports(sAN0|VSS_VDD);   //AN0 sera analogica las demas Digitales.
   SETUP_ADC(ADC_CLOCK_INTERNAL);   //La fuente de reloj para la conversion sera
                                    //El circuito RC interno del PIC, su TAD tipico
                                    //es de 4us.
//!   SETUP_ADC(ADC_CLOCK_DIV_32);     //La fuente de reloj para la conversion sera
                                    //Fosc externa entre 32, por lo tanto TAD=4us.
   SET_ADC_CHANNEL(0);              //Seleccionamos canal 0 para leerlo.
   lcd_init();                      //Inicializamos el LCD.
   lcd_putc("\fValor Leido=");      //Se muestra en el LCD la cadena "Valor Leido="
   while(TRUE){                     //Inicia el ciclo Infinito.
      val=READ_ADC();               //Se lee y guarda el valor del ADC
      printf(lcd_putc,"%4Lu",val);  //Se imprime en el LCD el valor leido del ADC
      lcd_gotoxy(13,1);             //Se envia el cursor a la columna 13 fila 1.
   }
}
```

Laboratorio 4: Interrupción por Lectura de Entradas.

4.1 Objetivo.

Familiarizarse con la utilización de la interrupción por cambio en las Entradas digitales mediante la implementación práctica del ejemplo.

4.2 Introducción.

En casi todos los proyectos es necesario leer alguna entrada de tipo digital conectada a pulsadores, interruptores, sensores digitales o similares. Este laboratorio trata de explicar cómo realizar esto de la manera más eficiente.

4.3 Interrupciones.

Una **Interrupción** consiste en un mecanismo por el cual un evento interno o externo puede interrumpir la ejecución de un programa en cualquier momento. Esto produce un salto automático a una **subrutina de atención a la interrupción**, ésta atiende inmediatamente el evento y retoma luego la ejecución del programa exactamente donde estaba en el momento de ser interrumpido.

Las fuentes de interrupción dependen del PIC utilizado. Por ejemplo, el PIC16F84 tiene 4 fuentes de interrupción mientras que la familia PIC16F88X tiene entre 13 y 14.

Los PIC DE gama baja y media tienen un único vector de interrupción situado en la dirección 04h de programa, mientras que los de gama alta tienen dos vectores de interrupción de distinta prioridad, alta y baja, situados en la posición 08h y 18h de la memoria.

4.3.1 Configuración de las interrupciones.

A continuación se describen un conjunto de funciones que se utilizan para configurar las interrupciones:

disable_interrupts(***level***)	Desabilita la interrupción Especificada.
enable_interrupts(***level***)	Habilita la interrupción especificada.
clear_interrupt(***level***)	Limpia el flag de interrupció especificado. Esto puede ser utilizado durante una interrupción global, o para prevenir la utilización de una interrpupción.

4.4 Interrupción Externa INT.

La fuente de interrupciones externa INT se utiliza para atender eventos externos en tiempo real, tales como detectar el cruce por cero de una señal. La interrupción se puede producir si y sólo si el cambio en RB0/INT es durante el flanco ascendente o si y sólo si es durante el flanco descendente.

4.4.1 Configuración de la interrupción externa INT.

Para realizar la configuración de la interrupción externa INT es necesario realizar lo siguiente:

1. Escribir la rutina de interrupción y sobre ella su directiva a como se muestra a continuación:

```
#INT_EXT
Void int_ext_isr(){
   …
```

```
    Código de la interrupcion
    …
}
```

2. Dentro de la rutina principal habilitar las interrupciones globales y Habilitar la interrupción INT a como se muestra a continuación:

```
Void main(){
…
enable_interrups(GLOBAL);  //Habilita las interrupciones Globales.
enable_interrups(INT_EXT); //Habilita las interrupcion INT_EXT.
…
}
```

4.5 Interrupción por cambio en PORTB.

La fuente de interrupciones por cambio en PORTB se utiliza para atender eventos externos en tiempo real, tales como manejo de botones o interruptores. La interrupción se puede producir si el cambio en alguno de los Pines en RB es durante el flanco ascendente o si es durante el flanco descendente.

Todos los pines del PORTB están configurados como pines de Interrupción-en-cambio. Para que se habilite la interrupción el valor presente es comparado con el valor almacenado en la última lectura para determinar cuál bit ha sido cambiado o cual no coincide.

Esta interrupción puede despertar al dispositivo del modo sleep. Para salir de la rutina de interrupción el usuario debe leer el PORTB para eliminar la condición de error.

4.5.1 Configuración de la interrupción por cambio en PORTB.

Para realizar la configuración de la interrupción por es necesario realizar lo siguiente:

1. Escribir la rutina de interrupción y sobre ella su directiva a como se muestra a continuación:

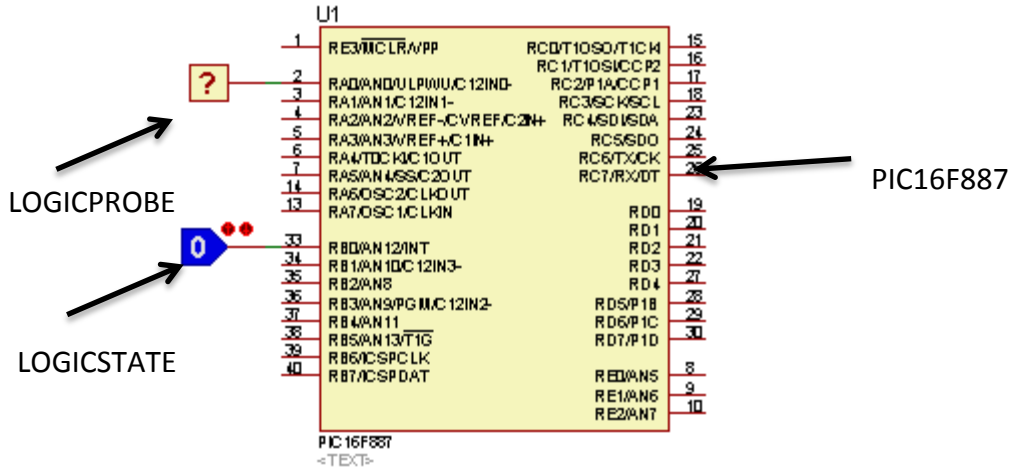
```
#INT_RBx
Void RBx_isr(){
    …
    Código de la interrupcion
    …
}
```

2. Dentro de la rutina principal habilitar las interrupciones globales y Habilitar la interrupción TMR0 a como se muestra a continuación:

```
Void main(){
…
enable_interrups(GLOBAL);  //Habilita las interrupciones Globales.
enable_interrups(INT_RBx); //Habilita las interrupcion INT_RBx.
…
}
```

4.6 Procedimiento.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**
2. Crear una Nueva carpeta llamada *Lab4* y una subcarpeta llamada *LED_Botones2* en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
3. Abrir la carpeta *LED_Botones2* y guardar el proyecto con el nombre de *LED_Botones2*.
4. Copiar el **Código Led_Botones2.**
5. Compilar el proyecto.
6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana PICK DEVICE el PIC16F887, el LOGICSTATE y el LOGICPROBE.



7. **Cargar el Archivo fuente** *Led_Botones2.cof* y **Realizar la simulación.**

4.7 Códigos.

4.7.1 Código Led_Botones2.

A continuación se muestra un código que hace que el LED conectado al pin RB0 se encienda cuando se presiones el botón conectado a RA0 mediante las interrupciones.

```
////////////////////////////////////////////////////////////////////////////
//Autor:   Luis Alberto Vargas Tijerino.
//Alias:   Bigluis.
//Pais:      Nicaragua.
//Fecha:   8-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT        //No Watch Dog Timer
#FUSES HS           //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT        //No Power Up Timer
#FUSES MCLR         //Master Clear pin enabled
#FUSES NOPROTECT    //Code not protected from reading
#FUSES NOCPD        //No EE protection
#FUSES NOBROWNOUT   //No brownout reset
#FUSES IESO         //Internal External Switch Over mode enabled
#FUSES FCMEN        //Fail-safe clock monitor enabled
#FUSES NOLVP        //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG      //No Debug mode for ICD
#FUSES NOWRT        //Program memory not write protected
#FUSES BORV40       //Brownout reset at 4.0V
#FUSES RESERVED     //Used to set the reserved FUSE bits

#use delay (clock=8M)          //Fosc=8Mhz
///PROGRAMA
```

```
#INT_RB
void RB_ISR(void){                          //Inicio de la interrupcion por cambio de PORTB
   int aux;                                 //Declaracion de la variable aux.
   output_bit(PIN_A0,input(PIN_B0));        //Hacemos que RA0 sea igual a RB0.
   aux=input B();                           //Leemos el valor del PORTB para eliminar la condicion
}                                           //de error y que sea posible salir de la interrupcion.

void main(void){
   setup adc ports(NO ANALOGS|VSS VDD);     //Todas las entradas del PIC Seran Digitales.
   setup_comparator(NC_NC);                 //Los comparadores Analogicos estaran apagados.
   ENABLE INTERRUPTS(GLOBAL);               //Habilitamos las interrupciones Globales
   ENABLE_INTERRUPTS(INT_RB0);              //Habilitamos las interrupciones por cambio en PIN_B0
   while(TRUE){                             //bucle infinito
      SLEEP();                              //Ponemos el PIC en modo de bajo consumo.
   }
}
```

Laboratorio 5: Interrupción por Desbordamiento del TMR0.

5.1 Objetivo.

Familiarizarse con la utilización de la interrupción por desbordamiento del TMR0 mediante la implementación práctica del **Código LED_Blink2.**

5.2 Introducción.

En el Laboratorio anterior se estudió el concepto de interrupción y se citaron las interrupciones producidas por cambio en las entradas. En este laboratorio estudiaremos la interrupción por desbordamiento del TMR0 en el cual realizaremos un ejemplo práctico.

5.3 Temporizadores.

Los TIMER o Temporizadores son módulos integrados en el PIC que permiten realizar cuentas tanto de eventos internos como externos. Cuando la cuenta es interna se habla de temporización y cuando la cuenta es externa se habla de contador.

Los Temporizadores son útiles en caso de requerir temporizaciones exactas, ya que sin ellos hay que tener en cuenta el tiempo de ejecución de las instrucciones y de los saltos.

Por lo general se comete el error de realizar las temporizaciones mediante *retardos* pero esto se hace imposible si necesitamos que el *PIC* realice otra tarea mientras se está ejecutando la temporización.

La manera más eficiente de realizar una temporización es cargar el valor en uno de los TIMER, luego se espera a que ocurra la interrupción cuando este se desborde, finalmente se realizan ajustes finos mediante *retardos* pequeños para aumentar la precisión de la temporización.

5.4 Módulo TIMER0.

El módulo TIMER0 es un Temporizador/Contador con las siguientes características:

- Temporizador/Contador de 8bits.
- Prescaler de 8 bits (compartido con el Watchdog Timer).
- Fuente de reloj externa o interna programable.
- Interrupción por desbordamiento.

5.4.1 Configuración de la interrupción por TIMER0.

Para realizar la configuración de la interrupción por es necesario realizar lo siguiente:

1. Escribir la rutina de interrupción y sobre ella su directiva a como se muestra a continuación:

```
#INT_TMR0
Void TMR0_isr(){
   …
   Código de la interrupcion
   …
}
```

2. Dentro de la rutina principal habilitar las interrupciones globales y Habilitar la interrupción TMR0 a como se muestra a continuación:

```
Void main(){
…
enable_interrups(GLOBAL);  //Habilita las interrupciones Globales.
enable_interrups(INT_TMR0);   //Habilita las interrupcion INT_RBx.
…
}
```

5.4.2 Configuración del TMR0.

La función que se encarga de realizar la configuración del módulo TIMER0 es la siguiente:

setup_timer_0 (***mode***)

Sintaxis	**setup_timer_0 (*mode*)**
Parametros	***mode*** podría ser una de las 2 siguientes constantes: • T0_INTERNAL • T0_EXT_L_TO_H • T0_EXT_H_TO_L • T0_DIV_1 • T0_DIV_2 • T0_DIV_4 • T0_DIV_16 • T0_DIV_32 • T0_DIV_64 • T0_DIV_128 • T0_DIV_256 ***period*** es un int de 0-255 que determina cuando el valor del reloj se reinicia. ***postscale*** es un numero de 1-16 que determina cuantos desbordes del timer antes de una interrupción. (1 significa una vez, 2 significa 2 veces, etc.).
Ejemplo	Setup_timer_0(**T0_INTERNAL|T0_DIV**);

5.4.3 Modificación y obtención del valor de TIMER0.

Las funciones que se encargan de realizar la obtención y modificación del módulo TIMER0 son las siguientes:

set_timer0() y get_timer0()

Sintaxis	**set_timer_0 (*value*) y get_timer0()**
Parametros	***value*** es el valor que tendrá TIMER0 en ese instante

5.4.4 Cálculo de la temporización.

Las temporizaciones se pueden calcular de la siguiente manera:

$$Temporizacion = \frac{4 \cdot T0DIV \cdot (256 - TMR0)}{Fosc}$$

5.4.4.1 Ejemplo

Se desea calcular una temporización de 20 ms para un PIC16F887 conectado a un CRYSTAL de 8MHz.

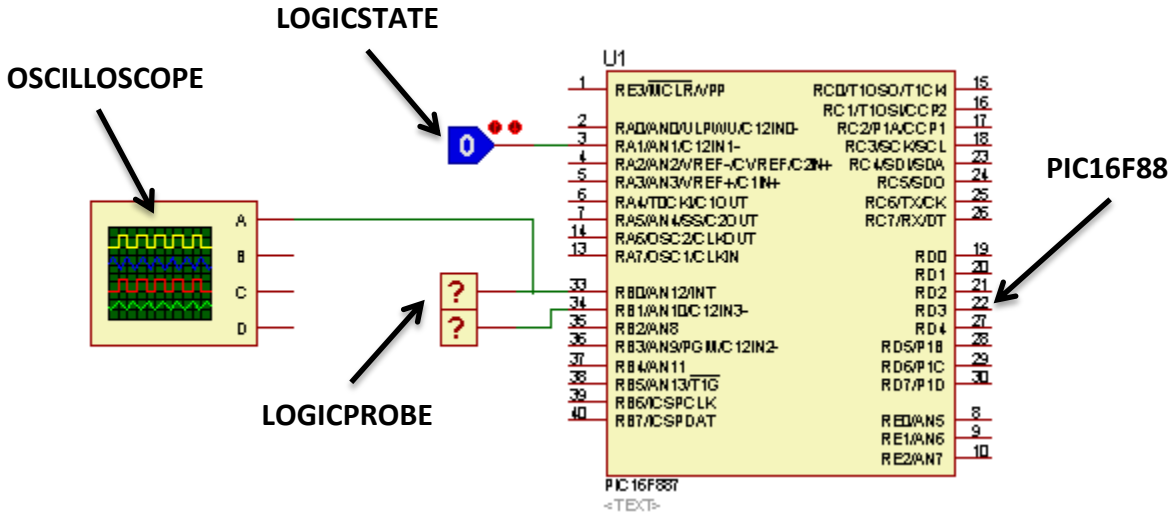
Sabemos que este PIC posee 9 prescaler para TIMER0, escogeremos T0DIV=256. Lo siguiente sería calcular el valor de TMR0.

$$TMR0 = 256 - \frac{Temporizacion \cdot F_{OSC}}{4 \cdot T0DIV}$$

$$TMR0 = 256 - \frac{20ms \cdot 8MHz}{4 \cdot 256} \approx 99$$

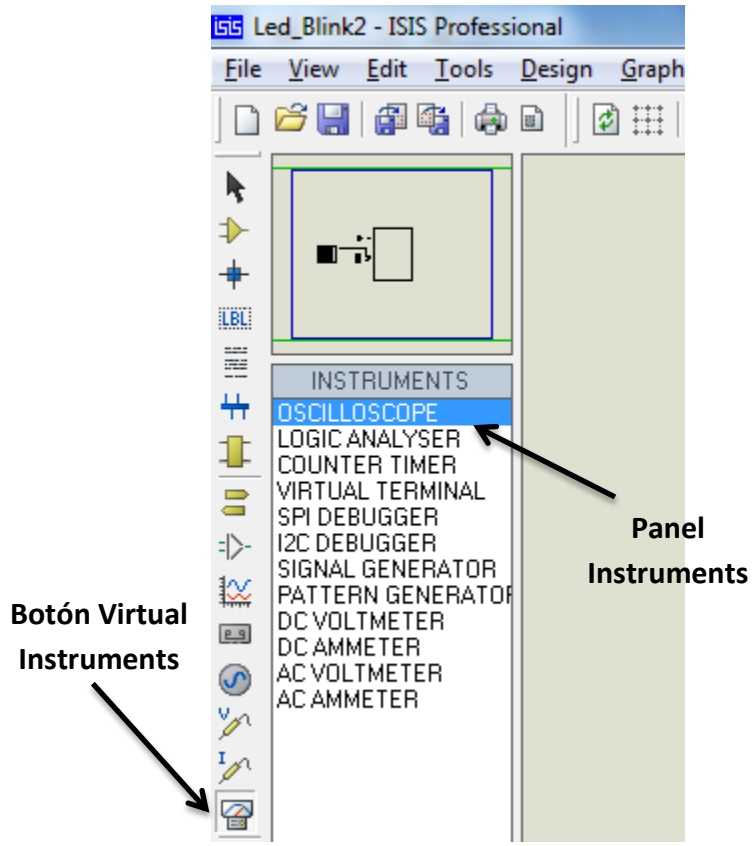
5.5 Procedimiento.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**
2. Crear una Nueva carpeta llamada *Lab5* y una subcarpeta llamada *LED_Blink2* en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
3. Abrir la carpeta *LED_Blink2* y guardar el proyecto con el nombre de *LED_Blink2*.
4. Copiar el **Código LED_Blink2.**
5. Compilar el proyecto.
6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana PICK DEVICE el PIC16F887, el LOGICSTATE y el LOGICPROBE.



7. Para insertar el OSCILLOSCOPE será necesario dar clic en el botón “Virtual Instrument” del cinta de botones izquierda y luego se debe dar clic en la opción “OSCILLOSCOPE” del panel lateral a como se

muestra en la siguiente figura.



8. **Cargar el Archivo fuente** *Led_Blink2.cof* y **Realizar la simulación.**

5.6 Códigos

5.6.1 Código LED_Blink2.

```
////////////////////////////////////////////////////////////////////////////////
//Autor:   Luis Alberto Vargas Tijerino.
//Alias:   Bigluis.
//Pais:    Nicaragua.
//Fecha:   24-Nov-10.
////////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT       //No Watch Dog Timer
#FUSES HS          //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT       //No Power Up Timer
#FUSES MCLR        //Master Clear pin enabled
#FUSES NOPROTECT   //Code not protected from reading
#FUSES NOCPD       //No EE protection
#FUSES NOBROWNOUT  //No brownout reset
#FUSES IESO        //Internal External Switch Over mode enabled
#FUSES FCMEN       //Fail-safe clock monitor enabled
#FUSES NOLVP       //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG     //No Debug mode for ICD
#FUSES NOWRT       //Program memory not write protected
#FUSES BORV40      //Brownout reset at 4.0V
#FUSES RESERVED    //Used to set the reserved FUSE bits

#use delay (clock=8M)         //Fosc=8Mhz
#use fast io(A)
#use fast_io(B)

//Interrupcion por TMR0.
//La interrupcion ocurrira (256-TMR0)*256*4/Fosc=(256-0)*256*4/8MHz=32.768ms
int1 aux;
#INT TIMER0
void TIMR0_ISR(){
   output_bit(PIN_B0,~input_state(PIN_B0));  //Si RB0 es igual a 1 entoneces el valor siguiente sera 0.
```

```
}                                          //Si RB0 es igual a 0 entoneces el valor siguiente sera 1.

///PROGRAMA Principal.
void main(void){
   set tris B(0);
   set tris A(255);
   setup_adc_ports(NO_ANALOGS|VSS_VDD);   //Todas las entradas del PIC Seran Digitales.
   setup_comparator(NC_NC_NC_NC);         //Los comparadores Analogicos estaran apagados.
   setup timer 0(T0 INTERNAL|T0 DIV 256); //La entrada de Reloj para TIMER0 sera interna.
                                          //Se dividira el reloj interno entre 256.
   setup timer 1(T1 DISABLED);            //Desabilitamos el TIMER1.
   setup_timer_2(T2_DISABLED,0,1);        //Desabilitamos el TIMER2.
   enable_interrupts(INT_TIMER0);         //Habilitamos la interrupcion por desborde del TIMER0.
   enable_interrupts(GLOBAL);             //Habilitamos la interrupciones globales.
   while(TRUE){                           //bucle infinito
      output_bit(PIN_B1,input(PIN_A1));   //RB1 sera igual a RA1.
   }
}
```

Laboratorio 6: Generador de PWM.

6.1 Objetivo.

Familiarizarse con la utilización del módulo CCP en modo PWM mediante la implementación práctica del Código GEN_PWM.

6.2 Introducción.

Este laboratorio trata de mostrar la teoría y el procedimiento utilizado para generar una señal PWM mediante el módulo CCP1 del PIC16F887.

6.3 Modulación por ancho de pulsos

La **modulación por ancho de pulsos** (**PWM**, *pulse-width modulation*) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

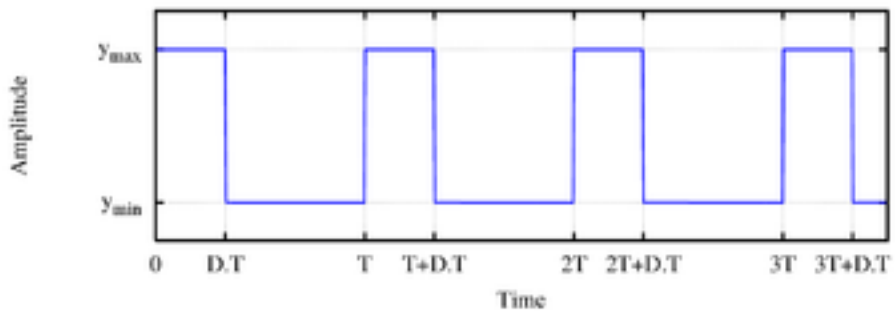
El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período. Expresado matemáticamente:

$$D = \frac{\tau}{T}$$

D es el ciclo de trabajo

τ es el tiempo en que la función es positiva (ancho del pulso)

T es el período de la función [Wikipedia]



[wikipedia]

6.4 Módulo CCP.

Los módulos CCP (*Capture/Compare/PWM*) permiten realizar las siguientes Funciones:

- Captura: obtiene el valor del temporizador en un momento dato, fijado por la acción de un terminal del PIC.
- Comparación: compara el valor del temporizador con el valor de un registro y provoca una acción en el PIC.

- PWM: genera una señal modulada por ancho de pulso.

Los PIC de gama media pueden tener hasta 2 módulos CCP.

6.4.1 Configuración del Módulo CCP en Modo PWM.

La función que se encarga de realizar la configuración del módulo CCP es la siguiente:

setup_ccpX( )

Sintaxis	**setup_ccp1 (*mode*) o setup_ccp2 (*mode*)**
Parametros	***mode*** es una constant. Las constantes válidas se encuentran en los archivos .h del dispositivo y son las siguientes: Deshabilita el módulo CCP CCP_OFF Configuración del CCP para modo PWM: CCP_PWM Habilita el Pulse Width Modulator

6.4.2 Configuración del ciclo de trabajo.

La función que se encarga de configurar el ciclo de trabajo es la siguiente.

Set_pwm_dutyX( )

Sintaxis	**set_pwm1_duty (*value*) ó set_pwm2_duty (*value*)**
Parametros	***value*** podría ser una variable o una constante de 8 o 16 bits.

6.5 Módulo TIMER2.

Es un temporizador de 8 bits, similar al TIMER0, con las siguientes características:

1. Registro TIMER2 de 8 bits.
2. Registro PR2 de 8 bits.
3. Interrupción al coincidir TIMER2 y PR2.
4. Prescaler programable por software (1:1, 1:4, 1:16).
5. Postcaler programable por software (1:1 hasta 1:16). [hoja de datos].

6.5.1 Configuración de la frecuencia PWM.

El Período de la señal se obtiene al configurar TIMER2 y el valor del registro PR2, para ello se utiliza la siguiente función:

setup_timer_2( )

Sintaxis	**setup_timer_2 (*mode*, *period*, *postscale*)**
Parametros	***mode*** podría ser*:* • T2_DISABLED • T2_DIV_BY_1 • T2_DIV_BY_4

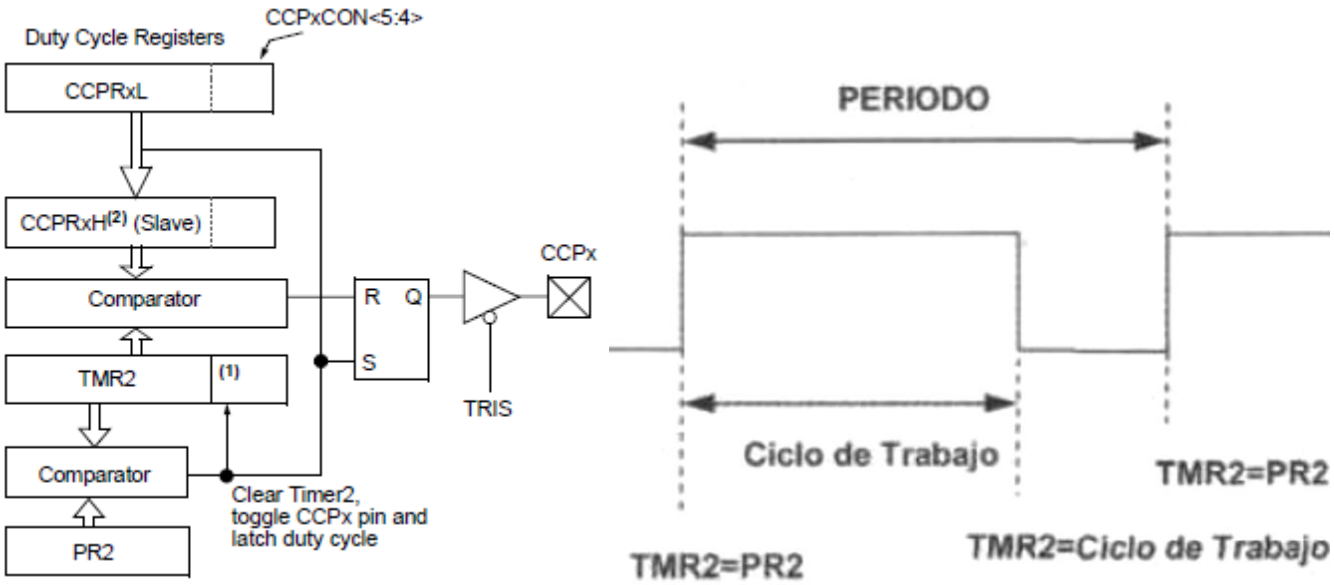
- T2_DIV_BY_16

period es un int de 0-255 que determina cuando el valor del reloj se reinicia.

postscale es un numero de 1-16 que determina cuantos desbordes del timer antes de una interrupción. (1 significa una vez, 2 significa 2 veces, etc.).

[ayuda ccs]

A continuación se describe el funcionamiento del módulo.



En la figura anterior podemos observar que cuando el registro PR2 coincide con el valor del TIMER2 ocurre lo siguiente.

1. Se reinicia TMR2.
2. El pin CCPx se pone a 1 (excepto cuando el Ciclo de trabajo es 0%).
3. El valor del Registro principal del Ciclo de Trabajo (*CCPRxL y CCPxCON<5:4>*) se carga en el registro auxiliar del Ciclo de trabajo (*CCPRxH mas 2bits)*.

El Registro principal del Ciclo de Trabajo (*CCPRxL y CCPxCON<5:4>*) puede ser escrito en cualquier momento pero no se cargan en el registro auxiliar (*CCPRxH mas 2bits)* hasta que finalice el Período.

Cuando el valor de TIMER2 coincide con el valor del Ciclo de Trabajo (*CCPRxH mas 2bits*) el pin CCPx se pone a 0 hasta que PR2 coincida de nuevo con TIMER2 y se inicie el nuevo Período.

6.5.2 Calculo de la Frecuencia PWM.

Con lo dicho anterior mente podemos determinar que la frecuencia PWM depende del valor del prescaler del TIMER2 *T2DIV* (valor de ***mode***) y *PR2* (valor de ***period***), a continuación se muestra la ecuación para calcular el Período PWM que nos brinda Microchip.

$$T_{PWM} = T_{OSC} 4 \cdot T2DIV \cdot (PR2 + 1)$$

Es decir:

$$F_{PWM} = \frac{F_{OSC}}{4 \cdot T2DIV \cdot (PR2+1)}$$

6.5.2.1 *Ejemplo:*

Deseamos que *Fpwm=1khz* para un PIC16F887 al que se le conecta un cristal de 8MHz.

Sabemos que este PIC posee 3 prescaler *T2DIV*=1, 4 y 16, por lo tanto el primer paso será despejar el valor de PR2 de la formula anterior.

$$PR2 = \frac{F_{OSC}}{4 \cdot F_{PWM} \cdot T2DIV} - 1$$

El segundo paso será sustituir valores y probar con T2DIV = 1.

$$PR2 = \frac{8MHz}{4 \cdot 1kHz \cdot 1} - 1 = 1999$$

Como el valor anterior es mayor al valor máximo de PR2, es decir 255, debemos aumentar el valor de T2DIV.

$$PR2 = \frac{8MHz}{4 \cdot 1kHz \cdot 4} - 1 = 499$$

Como aún el valor anterior es mayor al valor máximo de PR2, es decir 255, debemos aumentar el valor de T2DIV y finalmente encontramos que:

$$PR2 = \frac{8MHz}{4 \cdot 1kHz \cdot 16} - 1 = 124$$

Por lo tanto, para configurar la frecuencia PWM debemos ingresar la siguiente función:

```
Setup_timer_2(T2_DIV_16,124,1)
```

6.5.3 Resolución del Ciclo de Trabajo.

La resolución determina el número de posibles ciclos de trabajo para un período dado. Por ejemplo, una resolución de 10 bits resultará en 1024 ciclos de trabajo discretos, mientras que una resolución de 8 bits resultará en 256 ciclos de trabajo discretos.

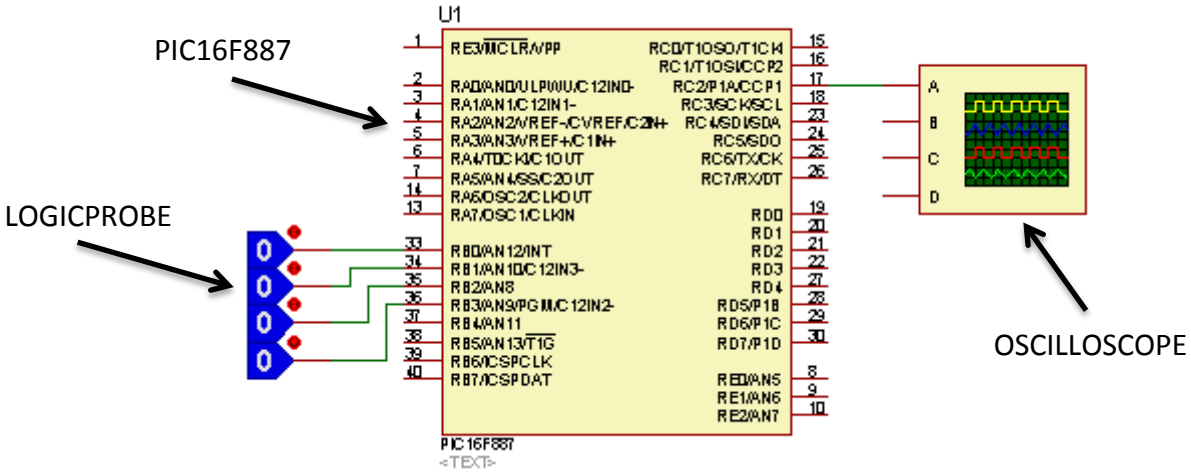
La máxima resolución es 10 bits cuando PR2 es 255. La resolución es función del valor del registro PR2 a como se muestra en la siguiente ecuación.

$$Resolution = \frac{log[4(PR2+1)]}{log(2)} \ bits$$

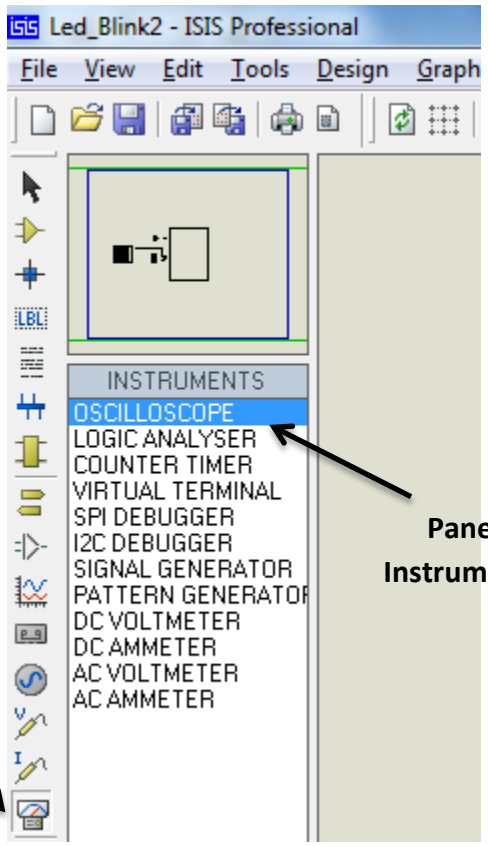
6.6 Procedimiento.

1. Repetir los pasos del 1 al 4 en **Crear el Proyecto en CCS.**

2. Crear una Nueva carpeta llamada *Lab5* y una subcarpeta llamada *LED_Blink2* en la ubicación que consideres necesaria (Ejemplo: Carpeta Mis Documentos).
3. Abrir la carpeta *LED_Blink2* y guardar el proyecto con el nombre de *LED_Blink2*.
4. Copiar el **Código LED_Blink2.**Código Led_Botones2.
5. Compilar el proyecto.
6. **Crear el circuito en Proteus**. Para ello será necesario buscar mediante la caja de búsqueda en la ventana PICK DEVICE el PIC16F887 y el LOGICPROBE.



7. Para insertar el OSCILLOSCOPE será necesario dar clic en el botón "Virtual Instrument" del cinta de botones izquierda y luego se debe dar clic en la opción "OSCILLOSCOPE" del panel lateral a como se muestra en la siguiente figura.



8. **Cargar el Archivo fuente** *GEN_PWM.cof* y **Realizar la simulación.**

6.7 Códigos.

6.7.1 Código GEN_PWM.

```
////////////////////////////////////////////////////////////////////////////
//Autor:   Luis Alberto Vargas Tijerino.
//Alias:   Bigluis.
//Pais:    Nicaragua.
//Fecha:   24-Nov-10.
////////////////////////////////////////////////////////////////////////////
#include <16f887.h>//pic a utilizar

#FUSES NOWDT       //No Watch Dog Timer
#FUSES HS          //High speed Osc (> 4mhz for PCM/PCH) (>10mhz for PCD)
#FUSES NOPUT       //No Power Up Timer
#FUSES MCLR        //Master Clear pin enabled
#FUSES NOPROTECT   //Code not protected from reading
#FUSES NOCPD       //No EE protection
#FUSES NOBROWNOUT  //No brownout reset
#FUSES IESO        //Internal External Switch Over mode enabled
#FUSES FCMEN       //Fail-safe clock monitor enabled
#FUSES NOLVP       //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NODEBUG     //No Debug mode for ICD
#FUSES NOWRT       //Program memory not write protected
#FUSES BORV40      //Brownout reset at 4.0V
#FUSES RESERVED    //Used to set the reserved FUSE bits

#use delay (clock=8M)         //Fosc=8Mhz
#use fast io(B)

void main(){
   signed int sel_Fpwm=0;
   unsigned int Duty=127;
   set_tris_B(0XFF);                    //Todos los pines de PORTB seran entradas.
   setup_comparator(NC_NC);             //Desconectamos los comparadores Analogicos
   SETUP ADC PORTS(NO ANALOGS);         //Solo AN0 sera analogico los demas digitales.
   SETUP_ADC(ADC_OFF);                  //Apagamos el ADC
   setup_ccp1(CCP_PWM);                 //Configuramos CCP1 en modo PWM simple.
   while(TRUE){                         //Inicio del ciclo Infinito
      while(!input_B());                //Espera a que se active un pin del PORTB.
      delay_ms(10);                     //Retardo de 10ms para eliminar rebotes.
      if(input(PIN B0))                 //Si se activa PIN B0
         if(++sel_Fpwm>3)               //preincrementamos sel_Fpwm y si es mayor que 3
            sel_Fpwm=0;                 //igualamos sel_Fpwm a 0.
      if(input(PIN_B1))                 //Si se activa PIN_B1
         if(--sel_Fpwm<0)               //predecrementamos sel_Fpwm y si es menor que 0
            sel_Fpwm=3;                 //igualamos sel_Fpwm a 2.
      if(input(PIN B2))Duty+=4;         //Si se activa PIN B2 incrementamos Duty en 4.
      if(input(PIN_B3))Duty-=4;         //Si se activa PIN_B3 decrementamos Duty en 4.


      set_pwm1_duty(duty);              //El %DT de la senial sera Duty.

      switch(sel Fpwm) {                        //Fpwm = Fosc/(4*T2 DIV*PR)
         case 0 :                               //Si sel Fpwm es 0
            setup_timer_2(T2_DIV_BY_16,249,1);//Fpwm = 8MHz/(4*16*249)=500Hz
            break;                              //Res=10Bits
         case 1 :                               //Si sel_Fpwm es 1
            setup timer 2(T2 DIV BY 16,124,1);//Fpwm = 8MHz/(4*16*124)=1000Hz
            break;                              //Res=7Bits
         case 2 :                               //Si sel_Fpwm es 2
            setup_timer_2(T2_DIV_BY_16,82,1); //Fpwm = 8MHz/(4*16*82)= 1500Hz
            break;                              //Res=6Bits
         default:                               //De lo contrario
            setup timer 2(T2 DIV BY 4,249,1); //Fpwm = 8MHz/(4*4*249)= 2000Hz
      };                                        //Res=10Bits
      while(input B());                         //Espera a que se desactiven los pines de PORTB
   }
}
```

Laboratorio 7: Transmisión Serial.