```
#define PB   PORTBbits.RB7

void main(void)
{
  TRISAbits.TRISA0=0;
  PORTAbits.RA0=0;
  ANSELAbits.ANSA0=0;
```

INTRODUCCIÓN

Un microcontrolador es el cerebro de un sistema embebido, obtiene datos desde el entorno, los procesa y genera señales para actuar sobre este. La interacción del microcontrolador y el medioambiente se realiza mediante pines, agrupados en los denominados puertos (**PORTX**), los cuales pueden ser configurados para funcionar como entradas o salidas.

Entre los microcontroladores PIC que serán utilizados en el curso sobre sistemas embebidos se encuentra el PIC16F1709, figura 1, el cual cuenta con 20 pines de los cuales 17 pueden ser configurados como entradas o salidas. Uno de los pines (**RA3**) es solamente entrada.

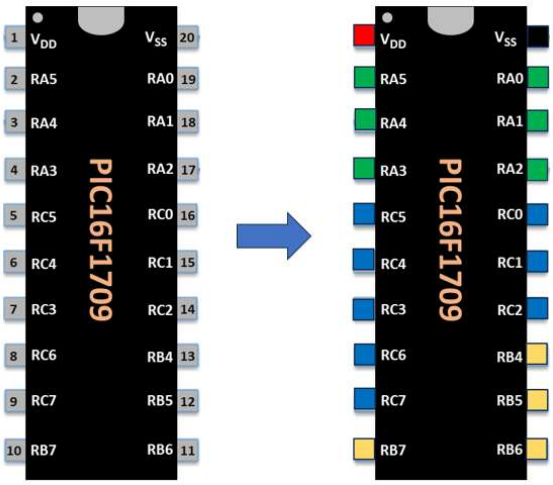

Figura 1. Pines en el PIC16F1709

La mayoría de los pines son agrupados en bloques denominados puertos (PORT en inglés) y para el microcontrolador bajo consideración tenemos:

PORTA: seis pines <RA5:RA0>

PORTB: cuatro pines <RB4:RB7>

PORTC: 8 pines <RC7:RC0>

En el diagrama de bloques del microcontrolador PIC16F1709, ver figura 2, podemos observar la presencia de los puertos mencionados, así como de varios periféricos tales como un convertidor analógico digital (ADC, por sus siglas en ingles), temporizadores como el TMR1, generador de señales de ancho de pulso modulado (PWM) entre otros.

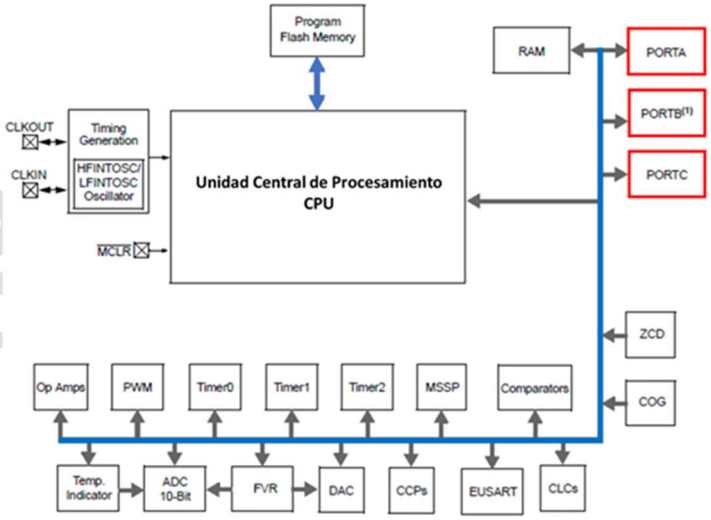

Figura 2. Diagrama de bloques PIC16F1709

Registros de Funciones Especiales (SFRs)

Los microcontroladores PIC presentan unos ***registros*** con funciones especiales (**SFR**, por sus siglas en inglés) para configurar el comportamiento de sus periféricos.

Un registro, en general, es una ubicación de memoria pequeña y de alta velocidad en la arquitectura interna del microcontrolador.

Los registros de funciones especiales son ubicaciones de memoria utilizados para:

- Controlar y manejar la operación de varios módulos periféricos del microcontrolador.

- Observar el estado y leer datos desde estos periféricos.

Algunas características de los registros especiales (SFRs):

- **Propósito específico:** Cada registro es diseñado para controlar un periférico o función particular.
- **Mapeados en memoria:** Los SFRs están ubicados en una región específica del espacio de memoria del microcontrolador, lo cual permite que el programa pueda acceder a ellos y modificarlos.
- **Control a nivel de bit:** Muchos SFRs están conformados de bits individuales, cada uno controlando un aspecto específico de la operación del periférico.

En esencia, los registros especiales proveen la interfaz entre el núcleo del microcontrolador y su mundo externo o periféricos internos, habilitando al programador para personalizar y controlar el comportamiento del dispositivo. Lo anterior se logra escribiendo un 1 o un 0 en la posición de un bit determinado.

La figura 3 muestra un listado parcial de los Registros de Funciones Especiales del microcontrolador PIC16F 1709. En la parte izquierda podemos identificar la dirección de memoria donde están ubicados, por ejemplo, el registro **PORTA** se encuentra en la dirección 00Ch, **PORTB** en 00Dh y **PORTC** en 00Eh.

TABLE 3-4: PIC16(L)F1709 MEMORY MAP (BANK

BANK 0		BANK 1		BANK 2	
000h	Core Registers (Table 3-2)	080h	Core Registers (Table 3-2)	100h	Core Registers (Table 3-2)
00Bh		08Bh		10Bh	
00Ch	PORTA	08Ch	TRISA	10Ch	LATA
00Dh	PORTB	08Dh	TRISB	10Dh	LATB
00Eh	PORTC	08Eh	TRISC	10Eh	LATC
00Fh	—	08Fh	—	10Fh	—
010h	—	090h	—	110h	—
011h	PIR1	091h	PIE1	111h	CM1CON0
012h	PIR2	092h	PIE2	112h	CM1CON1
013h	PIR3	093h	PIE3	113h	CM2CON0
014h	—	094h	—	114h	CM2CON1
015h	TMR0	095h	OPTION_REG	115h	CMOUT
016h	TMR1L	096h	PCON	116h	BORCON
017h	TMR1H	097h	WDTCON	117h	FVRCON
018h	T1CON	098h	OSCTUNE	118h	DAC1CON0
019h	T1GCON	099h	OSCCON	119h	DAC1CON1
01Ah	TMR2	09Ah	OSCSTAT	11Ah	—
01Bh	PR2	09Bh	ADRESL	11Bh	—
01Ch	T2CON	09Ch	ADRESH	11Ch	ZCD1CON
01Dh	—	09Dh	ADCON0	11Dh	—
01Eh	—	09Eh	ADCON1	11Eh	—
01Fh	—	09Fh	ADCON2	11Fh	—
020h	General Purpose Register 80 Bytes	0A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes
06Fh		0EFh		16Fh	
070h	Common RAM 70h – 7Fh	0F0h	Accesses 70h – 7Fh	170h	Accesses 70h – 7Fh
07Fh		0FFh		17Fh	

Figura 3. Registros de Funciones Especiales en el PIC16F1709 (Listado parcial)

Tabla 1: Registros Puertos Entrada/Salida

TRISx	**ODCONx**
PORTx	**SLRCONx**
LATx	**ANSELx**
INLVLx	**WPUx**

PUERTOS: REGISTROS

En la figura 1 es presentada la asignación de pines del PIC16F1709, PORTA en verde, PORTB en amarillo y PORTC en azul. Para configurar el comportamiento de los pines de los puertos, el microcontrolador cuenta con 8 registros de 8-bits cada uno y sus nombres son listados en la tabla 1.
En el PIC16F1709, la **x** puede ser **A**, **B** o **C.** Por el momento consideraremos los registros **TRISx**, **PORTx** y **ANSELx**.

Registros TRISx

En el PIC16F1709 los bits de los registros **TRISA**, **TRISB** y **TRISC** son utilizados para configurar si un pin de los puertos correspondientes funcionará como entrada o salida.

En la figura 4 se muestran los registros TRISX correspondientes a los tres puertos del PIC16F1709.

TRISA: PORTA TRI-STATE REGISTER							
U-0	U-0	R/W-1/1	R/W-1/1	U-1	R/W-1/1	R/W-1/1	R/W-1/1
---	---	TRISA5	TRISA4	---(1)	TRISA2	TRISA1	TRISA0
bit 7							bit0

TRISB: PORTB TRI-STATE REGISTER							
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	U-0	U-0	U-0	U-0
TRISB7	TRISB6	TRISB5	TRISB4	---	---	---	---
bit 7							bit0

TRISC: PORTC TRI-STATE REGISTER							
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
TRISC7(1)	TRISC6(1)	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
bit 7							bit0

Figura 4. Bits de los registros TRISA, TRISB y TRISC

- Si a un bit del registro **TRISx** se le asigna el valor '**1**' dicho pin será una **entrada**.

- Si a un bit del registro **TRISx** se le asigna el valor '**0**', dicho pin será una **salida**.

En los registros el bit menos significativo está ubicado en el extremo derecho (**bit0**) y el más significativo en el extremo izquierdo (**bit7**), ver figura 4.

Si para el funcionamiento deseado del sistema que estamos desarrollando se requiere que los pines <RC3:RC0> sean entradas y los pines <RC7:RC4> sean salidas, debemos asignarles los valores adecuados a los bits del registro TRISC. Para hacerlo es necesario utilizar el operador adecuado del lenguaje C.

Operadores en C

Un operador es un símbolo que le indica al compilador que debe realizar una determinada operación lógica o matemática. En un programa, los operadores son utilizados para manipular datos o variables.

El lenguaje C soporta un conjunto de operadores los cuales son clasificados, generalmente, como se muestra en el siguiente listado.

- Operadores aritméticos
- Operadores relacionales
- Operadores lógicos
- Operadores bitwise
- Operadores de asignación
- Operadores condicionales

Los operadores son componentes esenciales en cualquier lenguaje de programación y estos, al igual que otros componentes, serán presentados en el momento que su uso sea requerido para lograr la funcionalidad de un sistema dado.

Operadores de asignación

Tal como lo sugiere su nombre, la principal responsabilidad de los operadores de asignación en el lenguaje C es asignar valores a variables. La asignación se realiza ejecutando operaciones con ***operadores aritméticos***, o con ***operadores bitwise***, y asignándole el resultado a las variables.

- Los operadores aritméticos son usados para realizar cálculos matemáticos con tipos de datos numéricos en C.

- Los *operadores bitwise* en C se refiere a aquellos operadores que nos permiten manipular individualmente los bits de un número binario.

Uno de los operadores en la categoría de asignación es representado con el símbolo = y es llamado operador de asignación ya que toma el valor a su derecha y lo almacena en la variable a la izquierda.

Por ejemplo, para asignar el valor 0 o 1 a los bits del registro TRISC podemos utilizar el operador de asignación tal como se muestra a continuación:

TRISC **=** 0b00001111; (**TRISC=0x0F** en hexadecimal)

El valor en binario, 00001111, es almacenado en el registro TRISC. A los bits <RC7:RC4> se les asignó el valor 0 (salidas) y a los bits <RC3:RC0> se les asignó el valor 1 (entradas).

Los valores asignados a cada bit del registro TRISC son mostrados en la figura 5.

TRISC: PORTC TRI-STATE REGISTER							
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
TRISC7(1)	TRISC6(1)	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
bit 7							bit0
0	0	0	0	1	1	1	1

Figura 5. Configuración del registro TRISC

El diagrama de circuito de un semáforo básico, utilizando un PIC17F1709, es mostrado en la figura 6.

Los pines RA0, RA1 y RA2 del microcontrolador deben ser configurados como salidas y el pin RA3 como entrada.

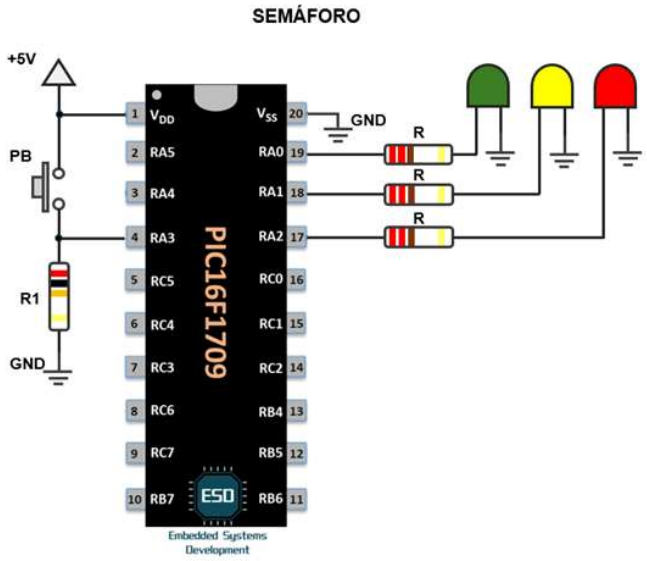

Figura 6. Diagrama de circuito de un semáforo

El valor de los bits puede ser seteado con las siguientes asignaciones:

```
TRISAbits.TRISA0=0;
TRISAbits.TRISA1=0;
TRISAbits.TRISA2=0;
TRISAbits.TRISA3=1;
```

En las asignaciones anteriores cada bit es puesto a 0, o a 1, de forma individual.

El registro TRISA del PIC16F1709 indica la existencia de 6 pines de los cuales RA3 es solamente entrada (TRISAbits.TRISA3=1). Si consideramos que los pines RA4 y RA5 serán salidas, debemos asignar al TRISA el valor 00001000. Es decir:

TRISA=0b00001000; (en binario)

TRISA=0x08; (en hexadecimal)

Dado que no conocemos el principio de funcionamiento de los diferentes componentes del circuito, por el momento aceptaremos que:

- Si **PB** (pulsador) no es presionado en el pin RA3 se aplican 0V (**0 lógico**).
- Si **PB** es presionado en el pin RA3 se aplican 5V (**1 lógico**).

En dependencia del nivel de voltaje en el pin RA3 (**entrada**) y del programa almacenado en la memoria del microcontrolador, este generará los voltajes, y los tiempos, en los pines RA0, RA1 y RA2 (**salidas**) requeridos para el funcionamiento apropiado del semáforo.

Registros ANSELx

Los registros **ANSELA**, **ANSELB** y **ANSELC** son utilizados para definir si un pin será una entrada digital o una entrada analógica. El funcionamiento es el descrito a continuación:

- Si a un bit del registro **ANSELx** se le asigna el valor '**1**' el microcontrolador lo considerará como una **entrada analógica**.
- Si a un bit del registro **ANSELx** se le asigna el valor '**0**', el microcontrolador la considerará como una **entrada digital**.

De los 18 pines de entrada/salida del microcontrolador PIC16F1709, 12 pueden ser configurados como entradas analógicas. En la hoja de datos (datasheet) encontraremos la información correspondiente.

En la figura 7, en el registro ANSELC a los bits <RC7:RC4> se les asignó el valor **1** y a los pines <RC3:RC0> el valor

0. Lo anterior significa que los pines de <RC7:RC4> serán entradas analógicas y los pines de <RC3:RC0> serán entradas o salidas digitales.

ANSELC: PORTC ANALOG SELECT REGISTER							
R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
ANSC7(2)	ANSC6(2)	ANSC5(3)	ANSC4(3)	ANSC3	ANSC2	ANSC1	ANSC0
bit 7							bit0
1	1	1	1	0	0	0	0

Figura 7. Configuración del registro ANSELC

Para lograr el comportamiento realizamos la asignación mostrada a continuación:

ANSELC=0b11110000; (**ANSELC=0xF0**, hexadecimal)

En la figura 7 podemos apreciar que todos los pines del PORTC pueden ser configurados como entradas analógicas.

En la figura 8 observamos que en el PORTA solamente cuatro pines pueden ser configurados como entradas analógicas (ANSA0, ANSA1, ANSA2 y ANSA4)

ANSELA: PORTA ANALOG SELECT REGISTER							
U-0	U-0	U-0	R/W-1/1	U-0	R/W-1/1	R/W-1/1	R/W-1/1
---	---	---	ANSA4	---	ANSA2	ANSA1	ANSA0
bit 7							bit0

Figura 8. Registro ANSELA

Registros PORTx

La interacción del microcontrolador y el medioambiente se realiza mediante pines, agrupados en los denominados puertos (**PORTX**), los cuales pueden ser configurados para funcionar como pines de entrada o pines de salida.

Los valores de los pines de un puerto pueden ser leídos o escritos. Conociendo el estado (**1** o **0**) de un pin dado nos permite tomar decisiones para incidir sobre el entorno. Los estados de los pines de salida pueden ser cambiados para lograr el comportamiento deseado del medio ambiente. Un ejemplo podría ser que cuando en un pin dado se apliquen 5V (**1 lógico**) provoque que un motor comience a girar. Lo último se lograría conectando el pin de salida del microcontrolador al manejador (driver) adecuado del motor tal como se muestra en la figura 9.

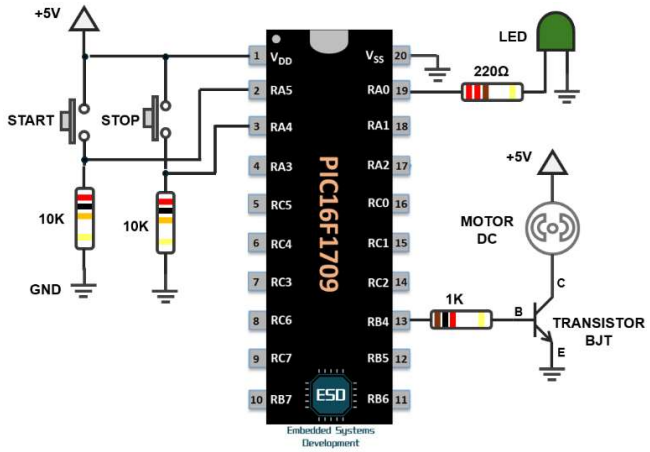

Figura 9. Motor DC conectado al pin RB4 (salida)

En el ejemplo #1 será presentada la herramienta básica para la toma de decisiones en el lenguaje C.

En este punto del camino tenemos los elementos suficientes para entender el manejo básico de las entradas y salidas digitales.

Procederemos al desarrollo del ejemplo #1 el cual, aunque es bastante simple, será de mucha utilidad para lograr lo siguiente:

- Presentar los pasos que pueden seguirse para desarrollar un sistema embebido basado en un microcontrolador
- Presentar varios elementos básicos utilizados para garantizar la funcionalidad de un sistema. Serán presentados nuevos operadores, el enunciado básico para la toma de decisiones en C y algunas formas de implementar un lazo (loop) infinito.

PASOS PARA EL DESARROLLO DE UN SE

Los pasos, y el orden de aplicación, para desarrollar un sistema embebido podrían ser los siguientes:

1. Descripción del proyecto
2. Diagrama de bloques del proyecto
3. Diagrama de circuito del proyecto

4. Descripción del hardware
5. Algoritmo del proyecto (solución)
6. Programa del proyecto
7. Efectividad del programa

Es muy importante para minimizar el tiempo de desarrollo y evitar la presencia de errores en el sistema embebido seguir los pasos listados. En cada ejemplo presentado será mostrado cada uno de los pasos.

Ejemplo #1: Encendido de un Led

1. Descripción del proyecto

El documento de los requerimientos de un sistema presenta una descripción detallada sobre lo que el sistema o aplicación debe hacer, como este debe comportarse, así como de las restricciones y otros factores que este debe satisfacer. Dicho documento es fundamental para asegurar que el sistema o aplicación sea desarrollado para cumplir con las necesidades del cliente. La elaboración de los requerimientos del sistema es un aspecto fundamental en el proceso de desarrollo de un sistema embebido.

Para el ejemplo #1, el párrafo siguiente sería suficiente para describir que desea el cliente.

Se requiere desarrollar un sistema, utilizando un ***microcontrolador PIC***, que garantice el encendido de un **LED** cuando se mantenga presionado un pulsador (**PB**). El LED debe estar apagado si el pulsador no es presionado. Para minimizar el tamaño, peso y costo del sistema, el reloj del sistema deberá ser generado utilizando el oscilador interno del microcontrolador.

Se deberá utilizar el compilador XC8 v2.36 de Microchip.

2. Diagrama de bloques del proyecto

Un diagrama de bloques es una representación visual de las principales partes, o funciones de un sistema, las cuales son conectadas mediante líneas que muestran la relación de los bloques. Es una representación de un alto nivel de abstracción cuya finalidad es tener una visión global del sistema sin tomar en cuenta los detalles de su implementación.

La figura 10 muestra el diagrama de bloques para el sistema de encendido del LED. PB representa el pulsador y el círculo el LED.

Para el funcionamiento deseado del pulsador y del LED se requieren algunos componentes que no son presentados en el diagrama de bloques. El diagrama de circuito del proyecto debe presentar todos los componentes.

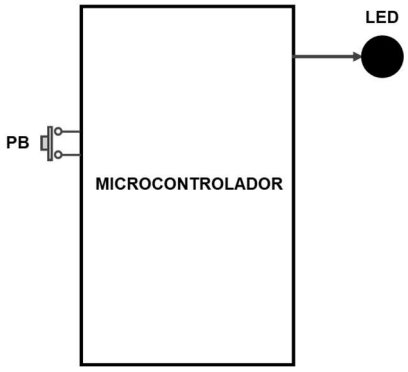

Figura 10. Diagrama de bloques del proyecto

3. Diagrama de circuito del proyecto

Un paso importante en el desarrollo de un sistema embebido es contar con el diagrama del circuito del proyecto, en el cual se muestran los diferentes componentes de este y como están conectados. Este diagrama juega un papel fundamental durante la elaboración del algoritmo del proyecto, así como en la codificación de la solución.

Para el sistema de encendido del LED el diagrama es el mostrado en la figura 11. **R1** es un resistor de Pull-down el cual, en conjunto con **PB1**, garantiza que al presionar el pulsador en el pin RA4 se apliquen 5V (**1 lógico**). **R2** es un resistor utilizado para limitar la corriente que circulará a través del diodo emisor de luz (LED, por sus siglas en inglés). Los valores permitidos de corriente deben buscarse en la hoja de datos del LED.

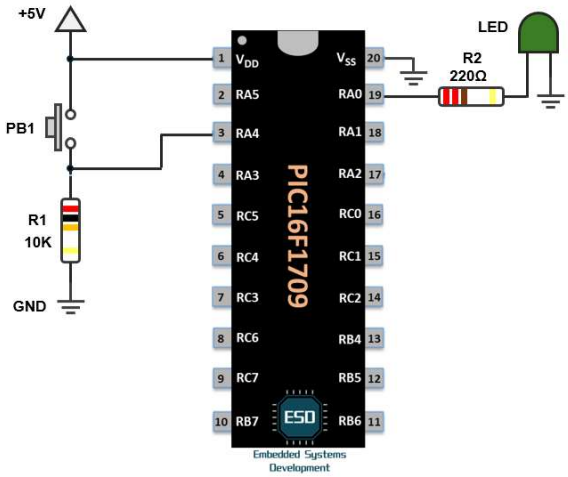

Figura 11. Diagrama del circuito del proyecto

La plataforma de hardware utilizada es un microcontrolador de 8-bit, el PIC16F1709. Desde luego, para un proyecto tan sencillo este microcontrolador es demasiado poderoso, lo recomendable es utilizar un microcontrolador con los recursos necesarios para garantizar el funcionamiento requerido. Durante la trayectoria aprenderemos a seleccionar el microcontrolador apropiado para una aplicación dada.

4. Descripción del Hardware

Es importante conocer cada uno de los componentes que conforman el circuito del proyecto, entender las leyes que rigen su comportamiento.

Por ejemplo, si nuestro sistema incorpora un LED debemos revisar su hoja de datos para obtener información de la corriente DC permitida a través del componente y sobre el voltaje entre sus extremos cuando está encendido. Dicha información juega un papel fundamental a la hora de determinar la resistencia del resistor limitador de corriente.

Aprovecharemos este ejemplo para introducir brevemente los componentes que conforman el circuito del proyecto, **el resistor**, **el pulsador**, **el LED**. El microcontrolador fue considerado en el apartado "Introducción a los microcontroladores". Serán presentadas dos configuraciones ampliamente utilizadas en los sistemas embebidos para aplicar 5V o 0V a un pin dado.

A continuación, es presentada una descripción breve de los componentes del circuito del proyecto.

- **Resistor**

Uno de los dispositivos más comunes en un sistema electrónico es el resistor el cual es un componente diseñado para introducir una resistencia eléctrica determinada entre dos puntos de un circuito.

La corriente eléctrica que circula a través de un resistor está determinada por la ley de Ohm la cual establece que la corriente que circula a través de este está determinada por el voltaje aplicado entre sus extremos y el valor de la resistencia del resistor. La figura 12 muestra un resistor conectado a una fuente de voltaje de 5V.

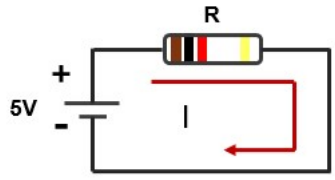

Figura 12. Resistor conectado en serie con la fuente de voltaje

La corriente que circula a través del resistor se determina utilizando la ley de Ohm tal como se muestra en la ecuación siguiente:

$$I = \frac{V}{R} \text{ Amperios}$$

Un resistor tiene una resistencia de 1 Ohm si al aplicar un voltaje de 1V entre sus extremos a través de este circula una corriente de 1 Ampere.

El valor de la resistencia de un resistor puede ser determinado utilizando un multímetro o utilizando una tabla de colores como la mostrada en la figura 13.

Generalmente trabajamos con resistores que cuentan con cuatro bandas de colores. En la tabla también es mostrado como proceder si el resistor es 5 o 6 bandas.

Como podemos apreciar, para el caso de cuatro bandas, el valor de la resistencia de un resistor es determinado de la siguiente forma:

R = (Primer dígito) (segundo dígito) * multiplicador.

Utilizando la tabla, determinar el valor de la resistencia del resistor en la figura 12. Calcular el valor de la corriente, I, en el circuito.

Primer dígito	**1**	**(café)**
Segundo dígito	**0**	**(negro)**
Multiplicador	**100**	**(rojo)**

R=10 x 100 = 1000Ω

Es común escribir el valor de la siguiente manera:

R = 1kΩ

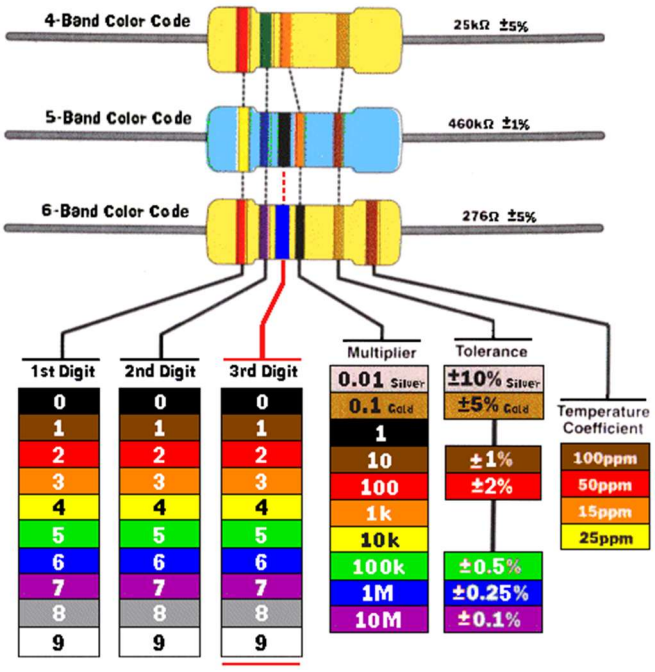

Figura 13. Código de colores para los resistores
Tomado de ….

La corriente a través del resistor será:

$$I = \frac{5V}{1000\Omega} = 0.005\ Amperios$$

Es decir, la corriente que circula a través del resistor es igual a 5 mA (miliamperios).

- **Pulsador (PB)**

Un ***pulsador*** es un componente que permite o impide el paso de la corriente eléctrica cuando se presiona. Al igual que los resistores, los pulsadores suelen estar presentes en los sistemas embebidos.

Podemos pensar en los pulsadores que aparecen en el panel ubicado en el interior de un ascensor o los pulsadores en un microondas.

En la figura 14 se puede apreciar que cuando el pulsador PB no es presionado, figura 14.a, no hay circulación de corriente a través del resistor.

En la figura 14.b, el pulsador está presionado lo que garantiza que se cierre el circuito permitiendo el paso de la corriente.

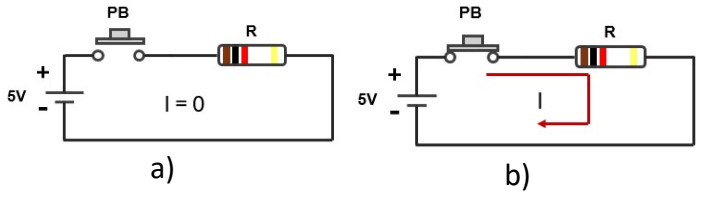

Figura 14. Efecto de un pulsador en un circuito

Resistores de Pull-up y Pull-down

En los sistemas embebidos es común encontrar combinaciones de **resistor + pulsador** como las mostradas en la figura 15.

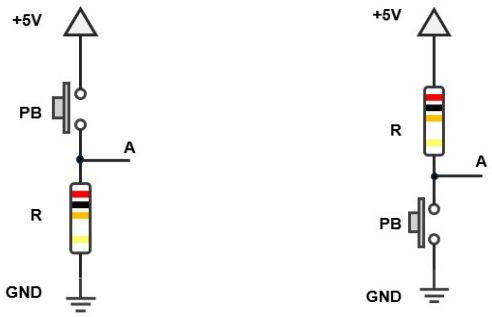

Figura 15. Resistores de Pull-down y Pull-up

- En el circuito mostrado a la izquierda, el voltaje en el punto A, si el pulsador no es **presionado**, será 0 voltios. Al presionar **PB**, el voltaje será 5 voltios. Al resistor se le denomina resistor de Pull-down.

- En el circuito de la derecha, el voltaje en el punto A, si el pulsador **no es presionado**, será 5 voltios. Al presionar **PB**, el voltaje será 0 voltios. Al resistor de le denomina resistor de Pull-up.

Las configuraciones mostradas son muy útiles cuando es necesario aplicar 5 o 0 voltios a otras partes de un sistema garantizando que el punto donde será aplicado el voltaje no quede flotante.

- **Diodo Emisor de Luz (LED)**

Otro dispositivo que frecuentemente aparece en los sistemas electrónicos es el LED, una fuente de luz que emite fotones cuando a través de este circula una corriente eléctrica de muy baja intensidad.

En la figura 16 es mostrado un circuito en el cual un LED fue conectado en serie con el pulsador y el resistor. En la figura 16.a el pulsador se encuentra en su estado normal, no-presionado, lo cual impide el paso de la corriente evitando que el LED sea encendido.

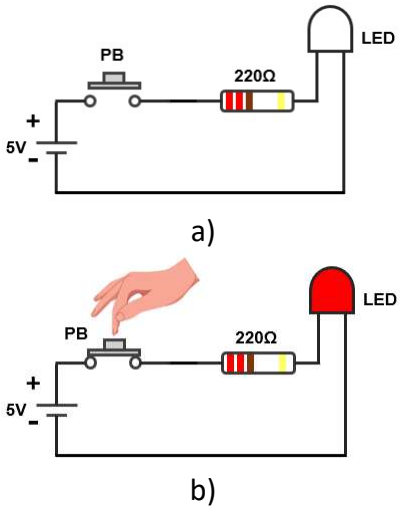

Figura 16. Circuito con diodo emisor de luz

En la figura 16.b el pulsador está presionado permitiendo el paso de la corriente y por consiguiente que el LED sea encendido.

La ecuación para calcular la corriente que circulará a través del resistor y del diodo LED (están conectados en serie) se puede obtener aplicando la regla de Kirchhoff para los voltajes la cual dice que la suma de las diferencias de voltaje en un lazo es igual a cero:

Si aplicamos la regla al circuito mostrado en la figura 17, obtendríamos lo siguiente:

$$+5V - V_R - V_D = 0$$

V_R es el voltaje entre los extremos del resistor

V_D es el voltaje entre los extremos del LED

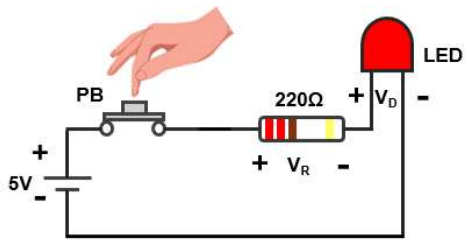

Figura 17. Voltajes en los componentes de un circuito

Asumimos que el voltaje entre los extremos del pulsador es cero.

La corriente que circula a través del resistor es la misma que circula a través del diodo, por lo tanto, $V_R = I*R$ y tomando en consideración dicha relación obtenemos:

$$I = \frac{5V - V_D}{220\Omega}$$

Para garantizar el funcionamiento apropiado del LED es fundamental revisar la hoja de datos (data sheet) de este, ahí encontraremos el valor de V_D entre otros datos. Generalmente es indicado el voltaje V_D relacionado con la corriente a través del LED y se debe determinar el valor de la resistencia del resistor para obtener la corriente requerida.

Si al revisar la hoja de datos encontramos que el voltaje entre los extremos del LED es de 2V (V_D) al utilizar la ecuación para determinar la corriente a través del LED encontramos:

$$I = \frac{5 - V_D}{220} = \frac{5 - 2}{220} = 13.6\,mA$$

Al igual que el pulsador, el LED puede estar en uno de dos posibles estados, encendido (**ON**) o apagado (**OFF**).

El concepto de estado es importante cuando modelamos el comportamiento de un sistema utilizando una máquina de estado finito. El **anexo A** presente información sobre está herramienta de modelación.

Un LED puede ser conectado a un microcontrolador en dos formas:

- ✓ El modo current – sourcing: el ánodo del LED es conectado, a través del resistor, al pin de salida. El cátodo es conectado a tierra. Ver figura 17.
- ✓ El modo current – sinking: el ánodo del LED es conectado a la fuente de alimentación, a través del resistor, y el cátodo es conectado al pin del microcontrolador. Ver figura 18.

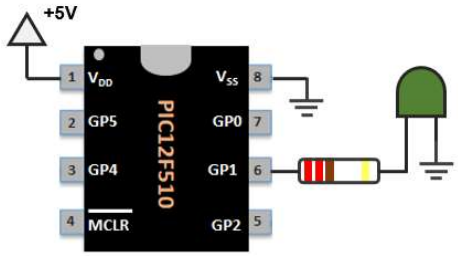

Figura 17. LED conectado en modo current-sourcing

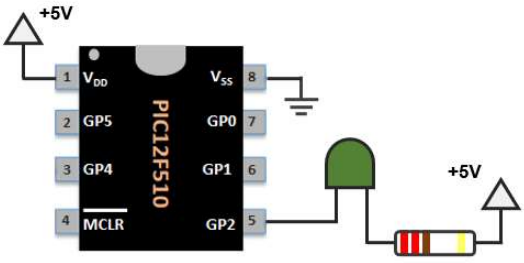

Figura 18. LED conectado en modo current-sinking

5. Lenguaje Descripción del Programa (PDL)

Antes de comenzar a escribir el código de una solución es recomendable contar con un algoritmo de esta ya que esto facilita considerablemente el proceso. Se puede utilizar un diagrama de flujo , seudocódigo o una máquina de estados finitos entre otras alternativas.

En el **anexo A** es presentada una descripción breve de las alternativas mencionadas.

En el curso básico será utilizado el Lenguaje de Descripción de Programa (**PDL**, por sus siglas en inglés).

Un PDL es un texto tipo inglés de formato libre que describe el flujo de control y datos en un programa. El PDL no es un lenguaje de programación, es una herramienta que ayuda al programador a pensar sobre la lógica del programa antes de escribirlo. Es una colección de palabras claves que ayudan al programador en la descripción de la operación de un programa en una manera lógica y gradual.

Cada descripción de programa PDL debe ser iniciado con la palabra **START** y terminado con la palabra **END**. Las palabras claves en un PDL deben ser resaltadas en negritas para hacer el PDL más claro. También es una buena práctica dejar un espacio entre las palabras claves y los enunciados del programa para mejorar la legibilidad del PDL.

Los enunciados **DO** -- **ENDDO** deben ser usados cuando se requiere crear iteraciones, o lazos condicionales e incondicionales en un programa. Cada enunciado DO debe ser terminado con la palabra clave ENDDO.

Otras palabras clave, tales como **FOREVER** o **WHILE**, pueden ser utilizadas después del enunciado DO para indicar un lazo infinito o un lazo condicional, respectivamente.

IF, **THEN**, **ELSE**, y **ENDIF** deben ser utilizadas para el cambio condicional del flujo de control en un programa. Cada palabra clave **IF** debe ser terminada con un **THEN**, y cada bloque IF debe ser terminado con la palabra clave **ENDIF**.

PDL del ejemplo #1.

```
START
    Configurar RA0 como salida
    Configurar RA4 como entrada digital
    Inicializar PORTA en cero
    Configurar oscilador interno a 4MHz
    DO FOREVER
        IF PB1 es presionado THEN
            Encender LED
        ELSE
            Apagar LED
        ENDIF
    ENDDO
END
```

La primera parte del PDL, entre **START** y **DO FOREVER**, es utilizada para definir las variables y para configurar los periféricos que serán utilizados.

En el ejemplo #1 son utilizados dos pines del PORTA y el oscilador interno el cual debe ser configurado a 4MHz.

El estado del pulsador **PB1** (presionado o no presionado) debe ser indagado de forma repetitiva e indefinida. En dependencia del resultado será ejecutado un bloque de código. Lo anterior es indicado entre las palabras clave **DO FOREVER** – **ENDDO**.

La decisión es tomada utilizando las palabras clave **IF-THEN-ELSE-ENDIF**.

Si el pulsador PB1 está presionado entonces se debe encender el LED, de lo contrario el LED deberá permanecer apagado.

La escritura del programa se realiza a partir del PDL y se simplifica considerablemente como veremos durante nuestra travesía. El algoritmo es muy importante para obtener retroalimentación del cliente y garantizar que los requerimientos no presenten ambigüedad alguna.

6. Programa del proyecto

El firmware del proyecto es desarrollado utilizando el ambiente integrado de aprendizaje MPLAB X IDE y el compilador XC8. Ambos pueden ser descargados desde la página de Microchip sin costo alguno.

Los pasos para seguir son listados a continuación.

1. Hacer click en el ícono de MPLAB X IDE
2. Seleccionar standalone Project
3. Seleccionar el microcontrolador a utilizar
4. Seleccionar el compilador XC8 (versión x.xx)
5. Nombrar el proyecto y salvarlo
6. Agregar new main.c y renombrarlo main.c

En la ventana del editor aparecerá lo mostrado en la figura 19. Pronto la función de cada uno de los elementos mostrados será explicada.

Antes de iniciar la edición del programa requerido para garantizar el comportamiento deseado del diodo LED, serán presentados los elementos que generalmente aparecen en un programa escrito para un microcontrolador PIC usando el compilador XC 8.

```
1  /*
2   * File:   main.c
3   * Author: Alejandro
4   *
5   * Created on April 24, 2024
6   */
7
8
9  #include <xc.h>
10
11 void main(void) {
12     return;
13 }
```

Figura 19. Área para la edición del programa

Del algoritmo al código

Un programa está conformado por una serie de elementos que cumplen una función y ocupan un lugar en este. Para los microcontroladores PIC es necesario, entre otras cosas, definir las palabras de configuración, definir las variables, inicializar periféricos, describir y definir funciones, escribir el código del cuerpo del programa.

En la figura 20 se muestra un formato con los principales elementos que conforman un programa escrito para un PIC con el compilador XC8. El formato debe ser considerado como una guía.

En el formato podemos apreciar que en la parte superior del programa podemos incluir información sobre el proyecto. Elementos importantes en esta parte son la descripción del proyecto, el autor, el microcontrolador utilizado, así como el compilador y su versión. La versión del programa es un dato de mucha importancia.

A continuación, será presentada una descripción breve de los diferentes elementos considerando el papel que juegan en el programa. Iniciaremos con las palabras de configuración (CONFIGURATION WORDS) las cuales tienen una importancia vital para el funcionamiento del dispositivo.

```
/*                          PROGRAM DESCRIPTION
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 Company: DIVERSA
 Author: Full name
 Date:Month, Year
 File Name:main.c
 Integrated Development Environment: MPLAB X IDE v6.00?
 Microcontroller: PIC?
 Compiler: XC8 v.36?
 Program version: v1.x
*/

/*CONFIGURATION WORDS*/

/*HEADER FILES*/

/*MACROS*/

/*FUNCTION DECLARATIONS*/

/*GLOBAL VARIABLES DECLARATION*/

/*INTERRUPT SERVICE ROUTINE (ISR)*/

/*MAIN FUNCTION*/

void main(void)
{
 /*LOCAL VARIABLES DEFINITION*/

 /*INITIALIZATION*/

 //PORTS
 //PERIPHERALS
 //OSCILLATOR

 /*BODY OF THE PROGRAM*/
  while(1)
  {
    code;
  }
}

/*FUNCTIONS DEFINITION*/
```

Figura 20. Formato elementos de un programa

PALABRAS DE CONFIGURACIÓN

Los microcontroladores PIC tienen registros que contienen los bits de configuración. Estos bits especifican la operación del dispositivo tal como el modo del oscilador, el perro guardián (watchdog), modo de programación y protección del código. Los bits deben ser seteados correctamente de lo contrario podría tener una falla en la ejecución del código o un microcontrolador que no funciona.

En lo que respecta a las palabras de configuración (**CONFIGURATION WORDS**), 2 en el PIC16F1709, tendríamos los siguientes registros:

CONFIG1: CONFIGURATION WORD 1												
R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
FCMEN	IESO	$\overline{\text{CLKOUTEN}}$	BOREN<1:0>		---	$\overline{\text{CP}}^{(1)}$	MCLRE	$\overline{\text{PWRTE}}$	WDTE<1:0>		FOSC<2:0>	
bit 13												bit 0

CONFIG2: CONFIGURATION WORD 2													
R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-1	U-1	U-1	U-1	R/P-1	R/P-1	R/P-1
LVP	$\overline{\text{DEBUG}}^{(2)}$	$\overline{\text{LPBOR}}$	BORV$^{(3)}$	STVREN	PLLEN	ZCDDIS	---	---	---	---	PPS1WAY	WRT<1:0>	
bit 13													bit 0

Figura 21. Palabras de configuración del PIC16F1709

La configuración del dispositivo consiste en poner a **1** o **0**, en dependencia del comportamiento deseado, los bits de las Palabras de Configuración las cuales, en el caso del PIC16F1709, son **CONFIG1** y **CONFIG2**.

```
#pragma config CONFIG1=0x08A4
#pragma config CONFIG2=0x3883
```

La directiva **#pragma config** permite que los bits de configuración del dispositivo sean especificados.

El significado de los valores asignados a CONFIG1 y CONFIG2 será explicado en el desarrollo de la unidad correspondiente.

Es importante señalar que el programa no debe mostrar errores o advertencias como resultado de la compilación. Se recomienda la compilación del programa a medida que las instrucciones son incorporadas. De esa forma es más fácil percatarse temprano si hay algún error en el código e identificar la causa. Lo anterior garantiza la funcionalidad del sistema y minimiza el tiempo de desarrollo de este.

ARCHIVOS DE CABECERA (HEADER FILES)

Los archivos de cabecera (header files) en lenguaje C contienen un conjunto de funciones de librería estándar predefinidas y otras entidades. Dichos archivos son incluidos en un programa usando ***la directiva #include*** la cual es usada para incluir los contenidos de otro archivo en el archivo fuente actual. Por ejemplo, para incluir los archivos asociados a las funciones que integra el compilador.

El compilador XC8 incluye un archivo de cabecera (header file) que es generalmente incluido en cada archivo fuente que escribamos. El archivo es **xc.h** y es un archivo de cabecera genérico que incluirá los archivos de cabecera de otros dispositivos, así como su arquitectura.

```
#include <xc.h>
```

El archivo de cabecera <xc.h> consiste en tipos, macros y funciones específicas para el dispositivo. Incluye archivos de cabecera específicos para el dispositivo que también proveen acceso a los registros de funciones especiales. Es incluido en cada programa escrito usando el compilador XC 8.

MACROS

En lenguaje C los macros son herramientas poderosas que le permiten al desarrollador definir porciones de código que pueden ser reutilizables. Los macros son definidos usando directivas del preprocesador y son usadas principalmente para generación y sustitución de código. Ellos presentan una alternativa para escribir código compacto y eficiente, mejorando la legibilidad y mantenimiento de los programas. Los macros son definidos usando la ***directiva #define***.

La sintaxis para definir un macro es la siguiente:

#define MACRO_NAME value

Cuando el compilador de C encuentra un macro en el código fuente, sustituye directamente el macro con el valor o expresión especificada. Los macros pueden ser utilizados para realizar sustituciones textuales simples o para definir bloques de código complejos.

Si queremos legibilidad en nuestro programa podemos utilizar la directiva #define para asignarle un nombre a un pin particular. Por ejemplo, en el diagrama del circuito del proyecto identificamos que el pulsador está conectado al pin RA4 y el LED, a través del resistor, está conectado al pin RA0. Usando la directiva #define podemos, en vez de referirnos a los pines, referirnos a los componentes conectados a ellos.

```
#define PB1 PORTAbits.RA4
#define LED PORTAbits.RA0
```

De igual forma podríamos asignarle a la palabra ON el valor 1 y a la palabra OFF el valor 0.

```
#define ON  1
#define OFF 0
```

DECLARACIÓN DE FUNCIONES

En lenguaje C, una función es un bloque de código que realiza una tarea específica y que puede ser invocada desde cualquier parte del programa las veces que sea necesario. Las funciones son un elemento de construcción fundamental en C y permiten, entre otras cosas, la modularidad y reusabilidad del código.

La declaración de una función es un enunciado que define las características esenciales de una función. Define su nombre, el tipo de valor de retorno y el tipo de cada uno de sus parámetros. La declaración de una función le indica al compilador que hay una función con el nombre dado definida en algún lugar del programa. La sintaxis para la declaración de una función en C es:

Return_type Function_name (Parameters – separated by comas)

El tipo de retorno (return type) de la función indica que tipo de valor es retornado una vez que la función ha sido ejecutada.

No todos los elementos mostrados en el formato formarán parte de un programa. Por ejemplo, en el ejemplo #1 no requiere el uso de funciones y dicho elemento no aparecerá en el programa.

DECLARACIÓN DE VARIABLES GLOBALES

Variables en C

Una variable no es más que el nombre de una localidad de memoria que utilizamos para almacenar datos. En C el valor almacenado en la variable puede ser cambiada durante la ejecución del programa.

En dependencia de su alcance (scope) las variables poder ser clasificadas como ***locales*** o *globales*. El alcance se refiere a la visibilidad y vida de una variable en el programa.

- Variable local: es visible solo en la función donde fue declarada.

- Variable global: es visible en todo el programa.

Entender el alcance de una variable nos ayuda a mantener la integridad de los datos y a prevenir conflictos potenciales en el programa.

Las variables deben ser declaradas e inicializadas antes de ser utilizadas. La declaración le indica al compilador que existe una variable con el nombre y tipos especificados de forma que este pueda continuar con su trabajo de compilación sin necesidad de más detalles acerca de la variable.

En el ejemplo #1 no son utilizadas variables globales y ese elemento no formará parte del programa. En ejemplos posteriores veremos cómo se describen e inicializan este tipo de variables y su impacto en el programa.

RUTINA DE SERVICIO A LA INTERRUPCIÓN

La rutina de respuesta a la interrupción (ISR, por sus siglas en inglés) es una función que es ejecutada cuando se produce una interrupción. En el compilador XC8 se utiliza el especificador ***interrupt*** para indicar que la función es una ISR. La estructura utilizada es mostrada a continuación:

```
void __interrupt() TMR0_INT(void)
{

}
```

El código que será ejecutado en respuesta a la interrupción se escribe entre las llaves.

En el ejemplo #1 no se utilizan interrupciones, por lo tanto, este elemento no aparecerá en el programa. Las interrupciones serán estudiadas en la unidad V.

FUNCIÓN PRINCIPAL (main function)

La función principal es el punto de entrada en un programa en C y en este solo debe existir una función main(). Los programas en C comienzan su ejecución llamando a la función main().

void main(void)
{ —> indica el inicio de la función main ()

La primera línea de código en la función main() es la declaración de variables. En C todas las variables deben ser declaradas antes de que sean utilizadas.

} —> indica el fin de la función main ()

En el seno de la función principal tenemos la definición de variables locales, la inicialización de los periféricos y el cuerpo del programa.

- **DEFINICIÓN DE VARIABLES LOCALES**

En el ejemplo #1 no son utilizadas variables locales y ese elemento no formará parte del programa.

- **INICIALIZACIÓN**

En la parte correspondiente a la inicialización se configuran los periféricos que serán utilizados en la solución. Para el ejemplo #1 solamente serán utilizados dos pines del PORTA, RA4 como entrada y RA0 como salida. En esta parte del programa aparecerá todo lo descrito entes del **DO FOREVER** en el PDL.

La configuración de los bits indicados se logra mediante las siguientes asignaciones:

```
TRISAbits.TRISA0=0;  //pin RA0 as output
TRISAbits.TRISA4=1;  //pin RA0 as input
ANSELA=0;            //PORTA pins, digital I/O
PORTA=0;             //PORTA pins set to 0
```

En la descripción se indica el uso del oscilador interno configurado a 4MHz. Eso es logrado asignando el valor indicado al registro de control del oscilador (OSCCON, por sus siglas en inglés). Los bits del registro OSCCON son mostrados en la figura 22.

OSCCON: OSCILLATOR CONTROL REGISTER							
R/W-0/0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
SPLLEN	IRCF<3:0>				---	SCS<1:0>	
bit 7							bit0

Figura 22. Registro OSCCON, PIC16F1709

En el PIC17F1709 el reloj del sistema puede ser generado por fuentes internas o externas lo cual es determinado poniendo a 0 o 1 los bits correspondientes en el registro OSCCON (SCS<1:0>).

La frecuencia del oscilador interno es determinada poniendo a 0 o 1 los bits correspondientes en el registro OSCCON (IRCF<3:0>).

Para el ejemplo #1 se requiere trabajar con el oscilador interno configurado a 4MHz. La siguiente asignación garantiza dicho requerimiento.

```
OSCCON=0x68;
```

Nada más que inicializar en el ejemplo #1.

- **CUERPO DEL PROGRAMA**

En el caso del ejemplo #1, se requiere que el LED sea encendido cuando se presiona el pulsador PB1 y sea apagado cuando no este presionado.

Analizando el algoritmo del ejemplo #1, identificamos dos actividades que deben ser realizadas:

1. El estado del pulsador (PB1) debe ser determinado, presionado o no presionado, y tomar una decisión respecto al estado del LED (encender o apagar).
2. El estado del pulsador debe ser interrogado repetida e indefinidamente.

Toma de decisiones en C

El lenguaje C incluye una serie de herramientas poderosas denominadas "Enunciados de Control". Dichos enunciados permiten controlar el flujo de ejecución en los programas, permitiendo comportamientos más complejos y dinámicos más allá de una ejecución lineal. Dos enunciados de control en C son:

- Enunciados para la toma de decisiones: ejecutan un bloque de código particular basado en ciertas condiciones. Los enunciados primarios para la toma de decisiones en C son el **if**, **if-else**, y **switch**.

- Enunciados para la ejecución de lazos (loops): usados para ejecutar repetidamente un bloque de código hasta que una condición dada es cumplida. C provee tres tipos de lazos: **for**, **while**, y **do-while**.

El enunciado (statement) más simple para tomar decisiones es el **if**. La sintaxis es la siguiente:

```
if(expresión)
{
  statements;
}

Siguiente instrucción;
```

- El enunciado **if** evalúa la expresión entre los paréntesis. Si la evaluación es verdadera las instrucciones en el cuerpo del if son ejecutadas.

- Si la evaluación es falsa las instrucciones en el cuerpo del **if** no son ejecutadas y el programa salta a **Siguiente instrucción**.

Las expresiones son escritas con la ayuda de operadores. Por ejemplo, usando el compilador XC8, podríamos escribir para el circuito de la figura 9:

```
if(PORTAbits.RA4==1)
{
  PORTAbits.RA0=1;
}
```

El símbolo == es un ***operador relacional*** y tiene el significado "igual a". El código anterior le indica al compilador que:

- Si el valor en el pin RA4 es igual a 1 (5V) entonces el pin RA0 debe ser puesto a 1 (5V). Dado que el LED está conectado, por medio del R2, al pin RA0 al aplicar a dicho pin 5V hará que el LED se encienda (**ON**).

- Si el valor en el pin RA4 es 0V la instrucción entre paréntesis no debe ser ejecutada.

El enunciado básico **if** puede ser extendido para tener mayor flexibilidad. La ampliación genera el enunciado

if-else cuya sintaxis es la siguiente:

```
if(expresión)
{
  Statement1;
}
else
{
  Statement2;
}
```

- Si la expresión es verdadera, el código en el cuerpo del enunciado **if** es ejecutado y el código en el cuerpo del enunciado **else** obviado (no ejecutado).

- Si la expresión es falsa, el código en el cuerpo del enunciado **else** es ejecutado y el código en el cuerpo del **if** es obviado (no ejecutado).

Por ejemplo, usando el compilador XC8, podríamos escribir para el circuito del ejemplo #1:

```
if(PORTAbits.RA4==1)
{
  PORTAbits.RA0=1;
}
else
{
  PORTAbits.RA0=0;
}
```

El código anterior le indica al compilador que:

- Si el valor en el pin RA4 es igual a 1 (5V) entonces el pin RA0 debe ser puesto a 1 (5V). Dado que el LED está conectado, por medio del R2, al pin RA0 al aplicar a dicho pin 5V hará que el LED se encienda (**ON**).

- Si el valor en el pin RA4 es 0V ejecutar el código en el enunciado **else**, es decir, poner a 0 el pin RA0.

En el ejemplo #1, el estado de RA4 debe ser monitoreado de forma repetida e indefinidamente.

Podemos utilizar uno de los enunciados utilizados para ejecutar repetidamente un bloque de código.

Por ejemplo, el while-loop. La sintaxis general es:

```
while(condición)
{
  Cuerpo del loop;
  Incrementar o decrementar;
}
```

El loop while consiste en una condición de control y se ejecuta mientras la condición sea verdadera. El código en el cuerpo del loop while es ejecutado después que la condición es ejecutada.

- La condición para continuar en el lazo es evaluada al inicio. Si el resultado de la evaluación es falso no se ejecutan el código entre las llaves. Mientras el resultado de la evaluación sea verdadero el código entre las llaves es ejecutado repetidamente.

- Si deseamos que un bloque de código sea ejecutado repetidamente de forma indefinida podemos utilizar cualquiera de las siguientes alternativas:

```
while(1)
{
  Statement1;
  Statement2;
}
```

while loop

```
for(;;)
{
  Statement1;
  Statement2;
}
```

for loop

Código del cuerpo de la función principal:

```
while(1)      //infinite loop
{
  if(PORTAbits.RA4==1)
  {
    PORTAbits.RA0=1;
  }
  else
  {
    PORTAbits.RA0=0;
  }
}
```

La condición siempre será verdadera (1), y el código entre las llaves se ejecutará indefinidamente.

DEFINICIÓN DE FUNCIONES

La definición de una función muestra el nombre de la función, el número y tipo de parámetros que espera recibir, y su tipo de retorno. La definición de la función también incluye el cuerpo la función con la declaración de las variables locales, y los enunciados que determinan que hace la función. La sintaxis es la siguiente:

Return_type Function_name(Parameters – separated by comas)
{
 Código;
}

El bloque de código entre las llaves es llamado cuerpo de la función (**function body**).

En el ejemplo #1 no se utilizan funciones y el elemento FUNCTION DEFINITION no aparecerá en el programa.

A continuación, el programa es mostrado por partes.

En la primera parte aparecen, en el orden listado, las palabras de configuración, el archivo de cabecera <xc.h> y los macros utilizados para definir símbolos que darán mayor legibilidad al programa. Con las definiciones mostradas en vez de escribir, en el código en el cuerpo del programa, PORTAbits.RA4 podemos escribir PB1. Algo similar con PORTAbits.RA0, podemos escribir simplemente LED.

```
18  #pragma config CONFIG1=0x08A4
19  #pragma config CONFIG2=0x3883
20
21  #include <xc.h>
22  #define PB1 PORTAbits.RA4
23  #define LED PORTAbits.RA0
24  #define ON  1
25  #define OFF 0
```

La segunda parte contiene la inicialización, en este caso de los registros del PORTA y del registro OSCCON.

```
27  void main(void)
28  {
29    TRISAbits.TRISA0=0;
30    TRISAbits.TRISA4=1;
31    ANSELA=0;
32    PORTA=0;
33
34    OSCCON=0x68;
```

La tercera parte contiene el cuerpo del programa y contiene el código que será ejecutado repetitiva y continuamente.

```
36    while(1)      //infinite loop
37    {
38      if(PORTAbits.RA4==1)
39      {
40        PORTAbits.RA0=1;
41      }
42      else
43      {
44        PORTAbits.RA0=0;
45      }
46    }
47  }
```

Figura 23. Código para el encendido de un LED

Una vez finalizada la edición del código se procede a su compilación. El resultado no debe mostrar errores (errors) o advertencias (warnings).

En la figura 24 es presentado el resultado de la compilación del programa correspondiente al ejemplo

#1. Como se puede apreciar no se reportan errores o advertencias. Si estuvieran presentes, el compilador nos brinda la información necesaria para identificar dónde se cometió el error o el porqué de las advertencias.

El compilador utilizado es el XC8, versión 2.36.

```
CLEAN SUCCESSFUL (total time: 13ms)
make -f nbproject/Makefile-default.mk SUBPROJECTS= .build-conf
make[1]: Entering directory 'C:/Users/alxof/OneDrive/DIVERSA_ESD/DIVERSA_
make  -f nbproject/Makefile-default.mk dist/default/production/UII_EJ_1.X
make[2]: Entering directory 'C:/Users/alxof/OneDrive/DIVERSA_ESD/DIVERSA_
"C:\Program Files\Microchip\xc8\v2.36\bin\xc8-cc.exe"  -mcpu=16LF1709 -c
"C:\Program Files\Microchip\xc8\v2.36\bin\xc8-cc.exe"  -mcpu=16LF1709 -Wl

Memory Summary:
    Program space        used    1Ch (    28) of  2000h words   (  0.3%)
    Data space           used     2h (     2) of   400h bytes   (  0.2%)
    EEPROM space         None available
    Configuration bits   used     2h (     2) of     2h words   (100.0%)
    ID Location space    used     4h (     4) of     4h bytes   (100.0%)

make[2]: Leaving directory 'C:/Users/alxof/OneDrive/DIVERSA_ESD/DIVERSA_E
make[1]: Leaving directory 'C:/Users/alxof/OneDrive/DIVERSA_ESD/DIVERSA_E

BUILD SUCCESSFUL (total time: 2s)
Loading code from C:/Users/alxof/OneDrive/DIVERSA_ESD/DIVERSA_ESD_CURSOS_
Program loaded with pack,PIC12-16F1xxx_DFP,1.3.90,Microchip
Loading completed
```

Figura 24. Resultado de la compilación con XC8

Se debe evitar la presencia de advertencias (warnings) ya que, aunque el programa al inicio puede funcionar con el tiempo pueden aparecer bugs que afectarán su buen funcionamiento.

• Efectividad del programa

Una vez que el programa ha sido editado y compilado es necesario verificar la efectividad de este. Existen diferentes alternativas, unas basadas en software como son los simuladores tales como PROTEUS y otras basadas en hardware, como son las tarjetas de desarrollo como el EASYPIC o el sistema mismo ensamblado en una tabla de nodos.

En el desarrollo del curso para verificar la efectividad de los programas escritos será utilizado **PROTEUS** en primer lugar y posteriormente la tarjeta de desarrollo **EASYPIC v7**. En ciertos momentos, con el fin de ejemplificar algunos detalles respecto a la optimización del código, utilizaremos el simulador que incorpora **MPLAB X IDE**.

Por lo tanto, las tres alternativas son:

- Suite PROTEUS
- EASYPIC V7
- Simulador incorporado en MPLAB X IDE

En el anexo D serán presentados los aspectos básicos de las herramientas mencionadas.

7.1 Simulación con PROTEUS

El poder simular el circuito del proyecto antes de ensamblarlo en una tabla de nodos o en un circuito impreso es fundamental tanto para garantizar la efectividad del sistema como para reducir el tiempo de desarrollo.

La simulación permite probar nuestro sistema mientras obtenemos los componentes requeridos para implementar la solución. También permite realizar experimentos, por ejemplo, del tipo "**qué pasa sí?**" lo que permite conocer la respuesta del sistema ante determinadas condiciones.

PROTEUS nos permite la simulación de un circuito mediante software y lo primero que debemos hacer es construir el esquemático del circuito. PROTEUS cuenta con una amplia librería con los modelos de cientos de componentes entre los que destacan los microcontroladores, plataforma de hardware utilizada en el curso para implementar un sistema embebido.

La figura 25 muestra el esquemático, elaborado en PROTEUS, para el sistema de encendido del LED. Al realizar la simulación podemos observar que mientras no se presione PB1 el LED no es encendido.

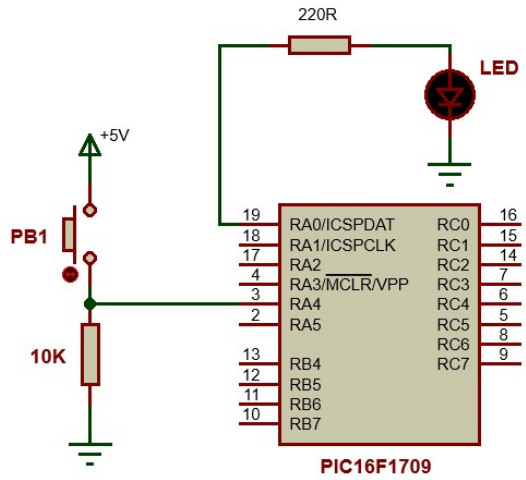

Figura 25. Esquema elaborado en PROTEUS

Los esquemáticos para cada uno de los ejemplos presentados en el curso estarán disponibles para que los estudiantes puedan realizar la simulación sin tener que invertir tiempo en la creación de este lo cual les permitirá concentrarse en la elaboración del código.

Para realizar la simulación utilizando PROTEUS:

1. Hacer click en el ícono de PROTEUS
2. Abrir el esquemático correspondiente al ejemplo bajo estudio.
3. Hacer click derecho sobre el microcontrolador y en editar propiedades:
   - Ajustar la frecuencia a 4MHz
   - Cargar el archivo HEX ubicado en la carpeta del proyecto.
4. Activar la simulación, ver figura 25.

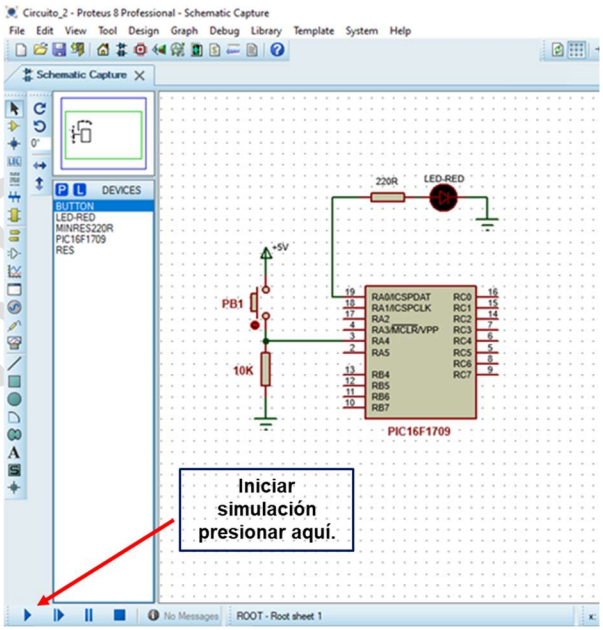

Figura 25. Inicio de la simulación en PROTEUS

Mientras la simulación esté activa, podemos manipular los valores o el estado de algunos de sus componentes y observar la reacción del sistema.

En el esquema del ejemplo #1, una vez inicializada la simulación, podemos presionar el pulsador PB1 y observar la respuesta del sistema. Si todo está correcto, el LED será encendido mientras PB1 este siendo presionado, es decir, el LED permanecerá en su estado encendido (**ON**), tal como se muestra en la figura 26.

Si dejemos de presionar a PB1, el LED pasar a su estado apagado (**OFF**).

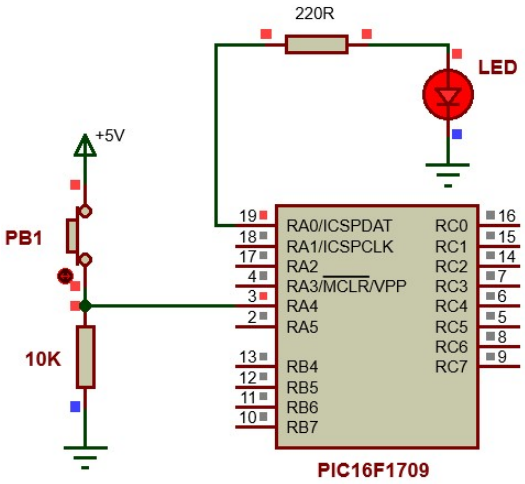

Figura 26. Simulación de un circuito en PROTEUS

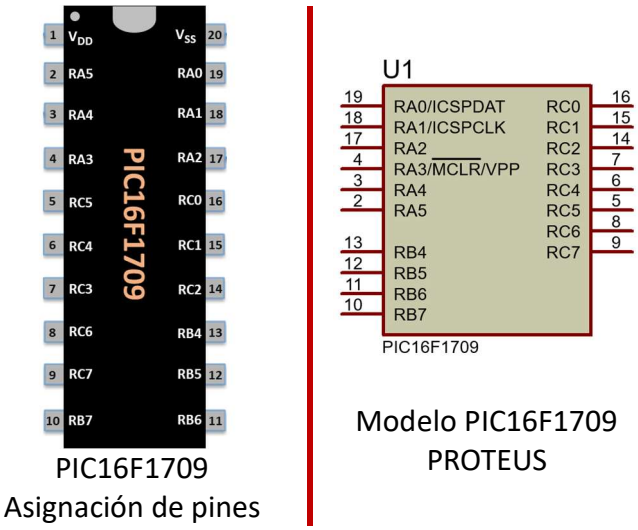

Figura 27. PIC16F1709 y su modelo en PROTEUS

El resultado sería el mismo si en el programa cambiamos el código:

```
36  while(1)      //infinite loop
37  {
38    if(PORTAbits.RA4==1)
39    {
40      PORTAbits.RA0=1;
41    }
42    else
43    {
44      PORTAbits.RA0=0;
45    }
46  }
```

por el código:

```
36  while(1)      //infinite loop
37  {
38    if(PB1==1)
39    {
40      LED=ON;
41    }
42    else
43    {
44      LED=OFF;
45    }
```

Es importante destacar que los modelos presentados por PROTEUS no presentan los pines del microcontrolador tal y como estos están dispuestos en el dispositivo real. Ver figura 27.

Debemos tener presente dicha situación a la hora de trazar las pistas del circuito impreso (PCB, por sus siglas en inglés).

El esquema para la simulación en PROTEUS y el video de la simulación están en:

- El esquema para el ejemplo #1 encuentra en UNIDADII – PROTEUS – ESQUEMAS – UII_EJ_1.
- El video para el ejemplo #1 se encuentra en UNIDADII – PROTEUS – VIDEOS – UII_EJ_1

7.2 Verificación con el EASYPIC v7

A diferencia del simulador del MPLAB X IDE y de PROTEUS, el EASYPIC v7 es una tarjeta de desarrollo que nos permite verificar la efectividad del programa utilizando componentes reales. La versión utilizada en el curso soporta más de 350 microcontroladores PIC de 8-bit los cuales van desde 8 pines hasta 40 pines.

En la figura 28, el contorno amarillo encierra las bases que permiten el uso de los diferentes microcontroladores. El empaquetado de los microcontroladores debe ser PDIP.

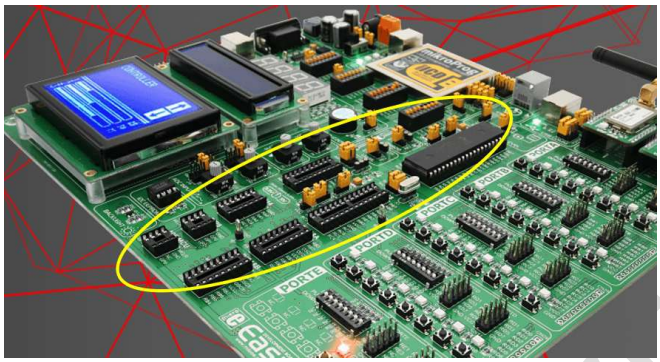

Figura 28. Más de 350 microcontroladores PICs

El EASYPIC v7 cuenta con una serie de switches mediante los cuales se puede configurar la tarjeta para que funcione apropiadamente en dependencia del número de pines del microcontrolador o en dependencia de si se utilizará un cristal o el oscilador interno de este.

En la figura 29 están remarcados algunos de los recursos, integrados en el EASYPIC v7, que facilitan la verificación de la efectividad de un programa.

Entre otros recursos destacan los siguientes:

- Pulsadores (Pushbutton)
- Diodos emisores de luz (LED)
- Dip Switch para cada puerto
- Potenciómetros para generar señales analógicas (2)
- Sensores de temperatura (DS18B20, LM35)
- Siete segmentos (7 SEG)
- Pantalla de cristal líquido (LCD)
- Interfaz para comunicación UART y USB

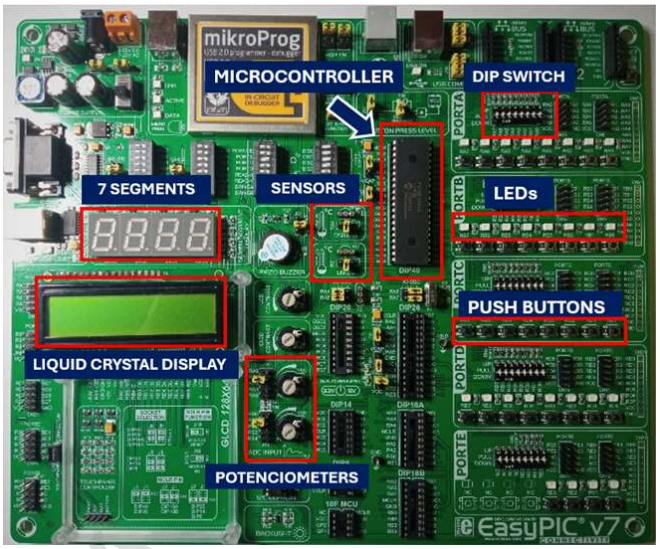

Figura 29. Recursos en el EASYPIC v7

Para ejecutar el programa utilizando EASYPIC v7 seguir los siguientes pasos:

1. Configurar el EASYPIC v7 de acuerdo con el número de pines del microcontrolador PIC17F1709 y tomando en cuenta que será utilizado el oscilador interno.

2. Colocar el microcontrolador en socket correspondiente.

3. Conectar el EASYPIC v7 a la computadora mediante el cable USB y energizar la tarjeta. Deberá encenderse el led amarillo (LINK)

4. Abrir el mikroProg Suite for PIC y cargar el archivo .hex (**LOAD**) correspondiente al ejemplo #1.

5. Grabar(**WRITE**) el archivo .hex en el microcontrolador.

6. Proceder con la verificación del funcionamiento del programa.

En el ejemplo #1, encendido de un LED, debemos presionar el pulsador (Pushbutton) etiquetado como

RA4 y, si el programa está correcto, el LED etiquetado RA0 deberá encenderse. Ver figura 29.

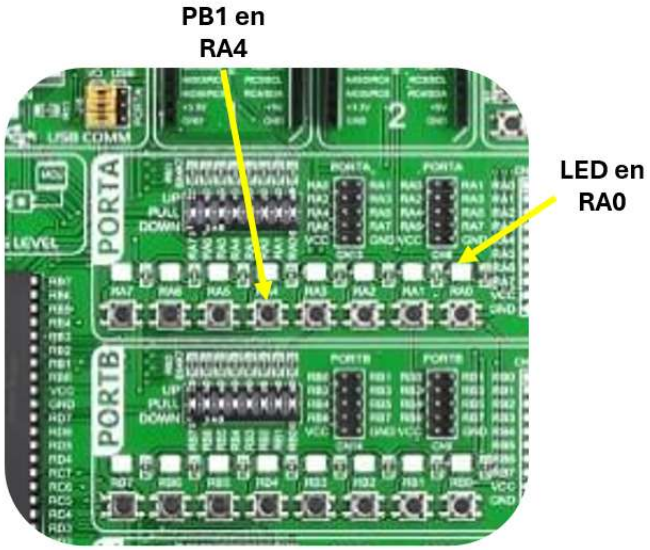

Figura 29. Pulsadores y Leds del PORTA en el EASYPIC

En la figura 30 podemos apreciar, en la parte izquierda, que cuando no se presiona el pulsador el LED permanece apagado. En la parte derecha cuando se presiona el pulsador, se enciende el LED conectado en RA0 y permanecerá en ese estado mientras el pulsador esté presionado.

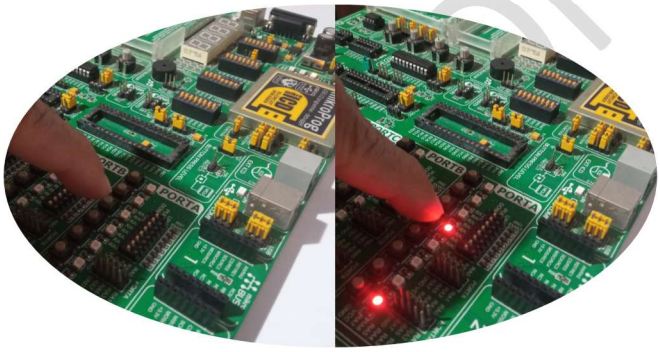

Figura 30. Usando el EASYPIC v7 para probar el código

Durante el desarrollo del curso el EASYPIC v7 y sus recursos serán presentados con detalle ya que es la principal herramienta para probar la efectividad de los programas. El EASYPIC es una herramienta basada en hardware y nos permite trabajar directamente con el microcontrolador de nuestro interés.

7.3 Simulador del MPLAB X IDE

El simulador de MPLAB X es un simulador de eventos discretos para dispositivos tales como:

- Las familias de microcontroladores PIC
- Las familias de Controladores de señales digitales dsPIC

El simulador, integrado en el MPLAB X IDE, es una herramienta diseñada para modelar la operación de los microcontroladores de Microchip como soporte en el proceso de depuración (debugging) del software elaborado para estos dispositivos. Una vez editado el programa basta con presionar la tecla indicada en la figura 31 para comenzar la simulación del programa.

Figura 31. Iniciar simulación con el simulador de MPLAB X

En la figura 32 es mostrada la pantalla que aparece cuando se inicia la simulación. En la parte inferior se puede apreciar que se han incluido los pines de entrada y salida correspondientes al ejemplo #1. RA4 aparece etiquetada como entrada digital (**Din**) y RA0 como salida digital (**Dout**). En dicha figura RA4=0 y por lo tanto el LED (RA0) permanece en 0.

```
28  void main(void)
29  {
30      TRISAbits.TRISA0=0;  //pin RA0 as output
31      TRISAbits.TRISA4=1;  //pin RA4 as input
32      PORTA=0;             //PORTA pins set to 0
33      ANSELA=0;            //PORTA pins are digital I/O
34
35      OSCCON=0x68;         //Internal oscillator, 4MHZ
36
37      while(1)             //infinite loop
38      {
        if(PB1==1)  //PB1 pressed
40        {
41          LED=1;   //+5V at pin RA0
42        }
43        else
44        {
45          LED=0;   //0V at pin RA0
46        }
47      }
48  }
```

Output | I/O Pins × | Call Stack | Breakpoints

Pin	Mode	Value
RA4	Din	0
RA0	Dout	0

Figura 32. Ejemplo con el simulador de MPLAB X IDE

En la figura 33 el valor de RA4 fue cambiado a 1. En la línea 39 como PB1 es igual a 1 se ejecutará la instrucción de la línea 41 poniendo a 1 RA0 (pin donde está conectado el LED).

```
28  void main(void)
29  {
30      TRISAbits.TRISA0=0;  //pin RA0 as output
31      TRISAbits.TRISA4=1;  //pin RA4 as input
32      PORTA=0;             //PORTA pins set to 0
33      ANSELA=0;            //PORTA pins are digital I/O
34
35      OSCCON=0x68;         //Internal oscillator, 4MHZ
36
37      while(1)             //infinite loop
38      {
        if(PB1==1)  //PB1 pressed
40        {
41          LED=1;   //+5V at pin RA0
42        }
43        else
44        {
45          LED=0;   //0V at pin RA0
46        }
47      }
48  }
```

Output | I/O Pins × | Call Stack | Breakpoints

Pin	Mode	Value
RA4	Din	1
RA0	Dout	1

Figura 33. Ejemplo con el simulador de MPLAB X IDE

Durante el desarrollo del curso, las herramientas presentadas serán utilizadas para probar la efectividad de los programas. Los recursos que nos ofrecen serán más evidentes en la medida que los problemas a resolver sean más complejos.

Ejemplo #2: Activación motor DC, v1

Continuaremos nuestro camino desarrollando el ejemplo #2 el cual, al igual que el ejemplo #1, servirá para introducir nuevos elementos de hardware y software que vendrán a fortalecer nuestra caja de herramientas necesarias para el diseño e implementación de un sistema embebido.

1. Descripción del proyecto

Se requiere el desarrollo de un sistema, basado en un microcontrolador, que permita la energización de un motor DC pequeño cuando dos pulsadores sean presionados simultáneamente. Un LED deberá ser encendido cuando el motor esté energizado. El reloj del sistema debe ser implementado con el oscilador interno del microcontrolador. Si uno o ambos pulsadores no son presionados, el motor y el LED deben ser desenergizados.

2. Diagrama de bloques del proyecto

La figura 34 muestra el diagrama de bloques del proyecto y podemos apreciar la presencia de un nuevo componente, un motor.

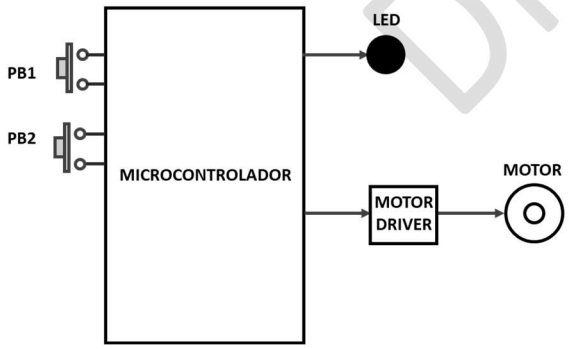

Figura 34. Diagrama de bloques del proyecto

3. Diagrama de circuito del proyecto

La figura 35 muestra el diagrama de circuito para el ejemplo #2. La mayoría de los componentes que conforman el circuito fueron presentados en Descripción del Hardware del ejemplo #1. Dos nuevos componentes, el transistor BJT y el MOTOR DC, forman parte del sistema.

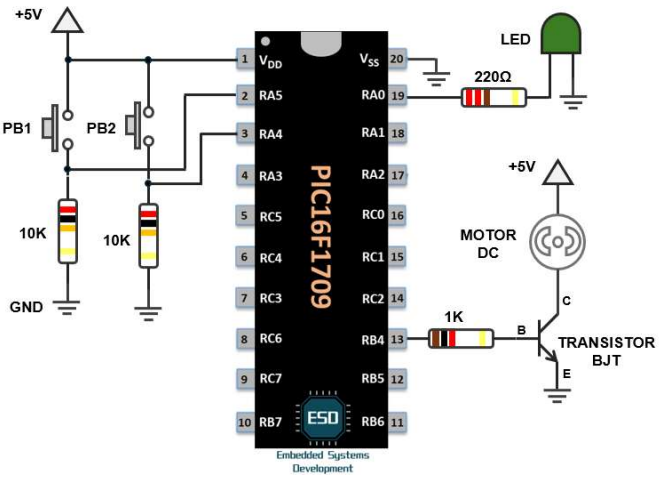

Figura 35. Diagrama del circuito del proyecto

4. Descripción del Hardware

El pulsador, el resistor y el diodo LED fueron presentados en el ejemplo #1. En el proyecto actual aparecen dos nuevos elementos un motor DC y un transistor BJT. A continuación, será presentada una descripción breve del transistor BJT y el motor DC.

- **Transistor BJT**

Un transistor de unión bipolar (**BJT**, por sus siglas en inglés) es un componente electrónico fundamental que puede amplificar una corriente o actuar como un switch. Está hecho de un material semiconductor y tiene tres terminales: base, colector y emisor. Aplicando una corriente pequeña en la base podemos controlar una corriente mayor que circula entre el colector y el emisor. Existen dos tipos principales, NPN y PNP, y aunque difieren en su construcción ambos trabajan de igual forma. Los transistores BJT son ampliamente utilizados en diferentes dispositivos electrónicos tales como amplificadores, switches, y en circuitos integrados complejos. La figura 36 muestra los símbolos de los transistores NPN y PNP.

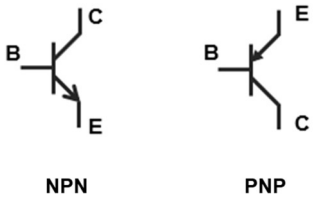

Figura 36. Símbolos de los transistores BJT.

En los sistemas embebidos el transistor es utilizado, principalmente, como switch. La idea básica es la siguiente:

Switch Abierto: El transistor debe estar en su región de corte, en la cual no circula corriente entre colector y emisor actuando como un switch abierto. Esto se logra aplicando un voltaje igual a cero en su base.

Switch cerrado: El transistor debe estar en su región de saturación permitiendo que la corriente máxima circule entre colector y emisor actuando como un switch cerrado. Esto se logra aplicando suficiente voltaje en su base.

En esencia, una pequeña corriente en la base controla una corriente mayor entre colector y emisor, haciendo que el transistor se comporte como un switch controlado por corriente.

Para implementaciones prácticas, necesitaremos componentes adicionales tal como resistores para controlar la corriente de base apropiadamente.

En la figura 37 se muestra un circuito en el cual el transistor actúa como un switch abierto.

- ✓ Cuando el switch está en la posición 2, en la base del transistor se aplican 0V y el transistor actúa como un switch abierto.

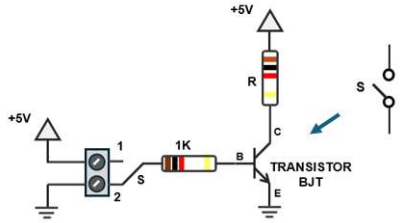

Figura 37. Transistor como switch abierto.

En la figura 38 se muestra un circuito en el cual el transistor actúa como un switch cerrado.

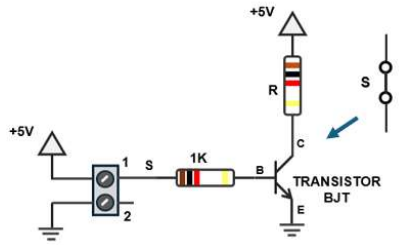

Figura 38. Transistor como switch cerrado.

- ✓ Cuando el switch está en la posición 1, se aplican 5V en el extremo izquierdo del resistor lo cual provoca que circule una corriente en la base del transistor cuyo valor puede ajustarse, mediante el resistor en la base, de forma que el transistor entre en su región de saturación actuando como un switch cerrado.

- **Motor DC**

Un motor DC es una máquina que utiliza corriente eléctrica directa (DC) para rotar. Funciona usando magnetismo: la electricidad crea campos magnéticos, y dichos campos mueven partes del motor para hacerlo rotar. Los motores DC son comunes en muchos dispositivos, desde herramientas de potencia a abanicos de computadoras.
En una implementación práctica se requerirá de un manejador (driver) para el funcionamiento adecuado del motor. El manejador funciona como un traductor entre el controlador y el motor DC.

Si el controlador es implementado usando un microcontrolador la corriente que este puede suministrar no es suficiente para suplir la requerida por el motor. El manejador actúa como un amplificador tomando la señal pequeña del microcontrolador y convirtiéndola en una señal alta corriente para manejar el motor. En breve, un manejador para un motor DC es esencial para controlar un motor DC efectiva y seguramente cuando utilizamos microcontroladores.

En la figura 35 podemos ver que la activación del motor es controlada desde el pin RB4. Si el pin RB4 es puesto a cero, el transistor actúa como un switch abierto y el motor no es activado. Si el pin RB4 es puesto a 1 (5V) el transistor actuará como un switch cerrado permitiendo la circulación de corriente y por ende la activación del motor.

5. Lenguaje Descripción del Programa (PDL)

El siguiente algoritmo se recoge el requerimiento de que ambos pulsadores deben ser presionados simultáneamente, mostrado en letras rojas, para que el motor sea energizado.

START
Configurar RA4 y RA5 como entradas
Configurar RA0 y RB4 como salidas
Inicializar PORTA y PORTB en cero
Configurar oscilador interno a 4MHz
DO FOREVER
IF PB1 **and** PB2 son presionados **THEN**
Encender LED
Energizar MOTOR
ELSE
Apagar LED
Desenergizar MOTOR
ENDIF
ENDDO
END

6. Programa del proyecto

La primera parte del programa muestra….

```
19  #pragma config CONFIG1=0x08A4
20  #pragma config CONFIG2=0x3883
21
22  #include <xc.h>
23  #define PB2 PORTAbits.RA5
24  #define PB1 PORTAbits.RA4
25  #define LED PORTAbits.RA0
26  #define MOTOR PORTBbits.RB4
27  #define ON  1
28  #define OFF 0
```

En la segunda parte del programa …..

```
30  void main(void)
31  {
32    TRISA=0x3E;
33    ANSELA=0;
34    PORTA=0;
35    TRISBbits.TRISB4=0;
36    ANSELB=0;
37    PORTB=0;
38    OSCCON=0x68;
```

La tercera parte del programa

```
40    while(1)      //infinite loop
41    {
42      if((PB1==1) && (PB2==1))
43      {
44        MOTOR=ON;
45        LED=ON;
46      }
47      else
48      {
49        MOTOR=OFF;
50        LED=OFF;
51      }
52    }
53  }
```

En la codificación del algoritmo podemos utilizar el operador lógico AND (**&&**) el cual es un operador binario dado que opera sobre dos elementos de datos. El operador && combina dos expresiones lógicas- es decir, dos expresiones que tienen un valor verdadero o falso (1 o 0).

En el ejemplo #2 podemos escribir el siguiente enunciado:

```
if((PB1==1) && (PB2==1))
{
  código;
}
```

La expresión es verdadera si ambas expresiones son verdaderas (PB1 está presionado y PB2 está presionado1) lo cual llevará a la ejecución del código entre las llaves del **if**. Si una o ambas expresiones son falsas, el resultado de la operación es falsa. No se ejecutará el código entre las llaves del **if**.

Otro operador lógico es el **OR** (||) y será presentado en el ejemplo #3.

En la figura 39 se muestra el programa para controlar la activación del motor DC. Las palabras de configuración CONFIG1 y CONFIG2 se mantienen igual que en el ejemplo #1. Entre la línea 23 y 28 se han definido una

serie de constantes simbólicas con el objetivo de hacer más legible el código.

Las asignaciones entre las líneas 32 y 37 se inicializan los puertos involucrados en la solución y en la línea 38 se configura el oscilador interno a 4MHz.
En la línea 40 utilizamos el while(**1**) para generar un lazo lo cual permitirá la ejecución del código entre las llaves del while de manera infinita.

En la línea 42 se utiliza el operador lógico AND (&&) para determinar si ambos pulsadores son presionados simultáneamente. Si el resultado es verdadero, el motor será energizado y el LED encendido. En caso contrario se ejecutará el código en el cuerpo del else lo que hará que el motor sea desenergizado y el LED apagado.

```
19  #pragma config CONFIG1=0x08A4
20  #pragma config CONFIG2=0x3883
21
22  #include <xc.h>
23  #define PB1 PORTAbits.RA5
24  #define PB2 PORTAbits.RA4
25  #define LED PORTAbits.RA0
26  #define MOTOR PORTBbits.RB4
27  #define ON  1
28  #define OFF 0
29
30  void main(void)
31  {
32    TRISA=0x3E;             //pin RA0 as output
33    ANSELA=0;               //PORTA pins, digital I/O
34    PORTA=0;                //PORTA pins set to 0
35    TRISBbits.TRISB4=0;
36    ANSELB=0;
37    PORTB=0;
38    OSCCON=0x68;          //Internal oscillator, 4MHZ
39
40    while(1)        //infinite loop
41    {
42     if((PB1==1) && (PB2==1))
43     {
44       MOTOR=ON;
45       LED=ON;
46     }
47     else
48     {
49       MOTOR=OFF;
50       LED=OFF;
51     }
52    }
53  }
```

Figura 39. Programa activación de motor DC

7. Efectividad del programa

Utilizaremos PROTEUS para verificar la efectividad del código elaborado. La figura 40 muestra el esquema del circuito.

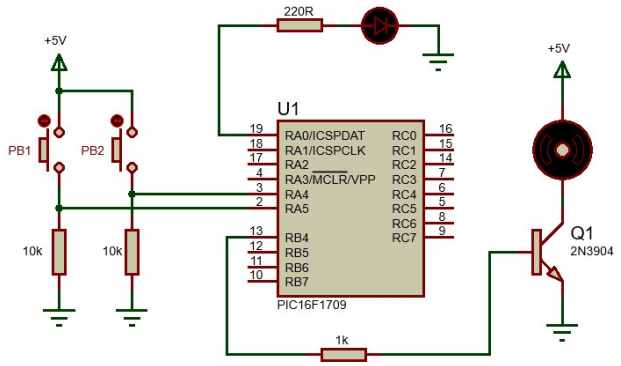

Figura 40. Esquema del circuito ejemplo #2

En las figuras 41 y 42 se muestra el resultado cuando solamente uno de los pulsadores está presionado. En ambos casos la expresión no es verdadera y por lo tanto no se ejecuta el código en el cuerpo del **if**, se ejecuta el del **else**.

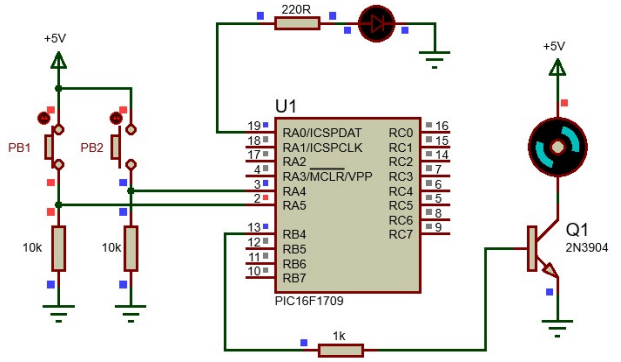

Figura 41. Solamente PB1 es presionado

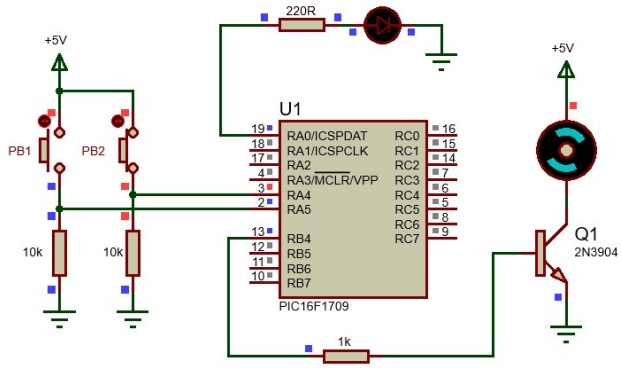

Figura 42. Solamente PB2 es presionado

En la figura 43, ambos pulsadores son presionados y se puede apreciar como el pin RB4 está rojo indicando que hay 5V. El LED está encendido y el motor girando lo cual puede ser visto en el video.

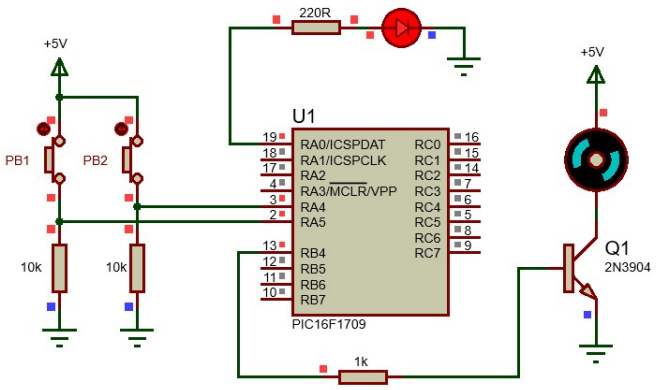

Figura 43. Ambos pulsadores están presionados

Se ponen a disposición del estudiante el esquema del circuito para su simulación en PROTUES y el video de la simulación para ver los pasos seguidos en la verificación.

- El esquema para el ejemplo #2 encuentra en UNIDADII – PROTEUS – ESQUEMAS – UII_EJ_2.
- El video para el ejemplo #1 se encuentra en UNIDADII – PROTEUS – VIDEOS – UII_EJ_2

Ejemplo #3: Activación motor DC v2

1. Descripción del proyecto

Se requiere el desarrollo de un sistema, basado en un microcontrolador, que permita la energización de un motor DC pequeño al presionar uno, o los dos, de dos pulsadores. Un LED deberá ser encendido cuando el motor esté energizado. El reloj del sistema debe ser implementado con el oscilador interno del microcontrolador.

El diagrama de bloques y el diagrama de circuito del proyecto son similares al del ejemplo #2. Dado que no hay componentes nuevos en el sistema, para aclarar dudas respecto a los componentes ver la descripción del hardware del ejemplo #2.

5. Lenguaje Descripción del Programa (PDL)

El algoritmo permanece igual excepto la primera línea después del DO FOREVER:

START
    Configurar RA4 y RA5 como entradas
    Configurar RA0 y RB4 como salidas
    Inicializar PORTA y PORTB en cero
    Configurar oscilador interno a 4MHz
    DO FOREVER
        IF PB1 **or** PB2 son presionados **THEN**
            TURN ON LED
            TURN ON MOTOR
        ELSE
            TURN OFF LED
            TURN OFF MOTOR
        ENDIF
    ENDDO
END

6. Programa del proyecto

Para implementar el código requerido en el compilador XC8 podemos utilizar el operador lógico OR (**||**) el cual cubre la situación cuando necesitamos chequear si una de dos o más condiciones es verdadera. Si uno o ambos operando del operador es verdadero, el resultado es verdadero. El resultado es falso cuando ambas expresiones son falsas.

```
19  #pragma config CONFIG1=0x08A4
20  #pragma config CONFIG2=0x3883
21
22  #include <xc.h>
23  #define PB2 PORTAbits.RA5
24  #define PB1 PORTAbits.RA4
25  #define LED PORTAbits.RA0
26  #define MOTOR PORTBbits.RB4
27  #define ON  1
28  #define OFF 0
29
30  void main(void)
31  {
32    TRISA=0x3E;            //pin RA0 as output
33    ANSELA=0;             //PORTA pins, digital I/O
34    PORTA=0;              //PORTA pins set to 0
35    TRISBbits.TRISB4=0;
36    ANSELB=0;
37    PORTB=0;
38    OSCCON=0x68;         //Internal oscillator, 4MH2
39
40    while(1)        //infinite loop
41    {
42     if((PB1==1) || (PB2==1))
43     {
44       MOTOR=ON;
45       LED=ON;
46     }
47     else
48     {
49       MOTOR=OFF;
50       LED=OFF;
51     }
52    }
53  }
```

Para el ejemplo #3, la línea 42 del programa tiene la siguiente forma:

if ((PB1==1) || (PB2==1))

El código en la línea 42 le indica al compilador que si PB1, o PB2, es presionado el motor y el LED deben ser energizados. Si los dos pulsadores son presionados simultáneamente el motor y el LED también serán energizados.

7. Efectividad del programa

Utilizaremos PROTEUS para verificar la efectividad del código elaborado. La figura 44 muestra el esquema del circuito.

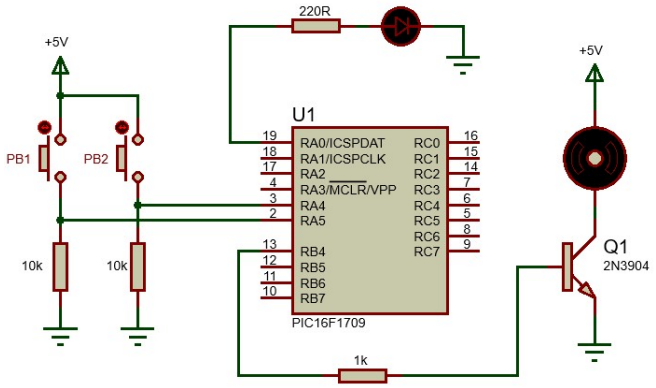

Figura 44. Esquema del circuito ejemplo #2

En las figuras 45 y 46 se muestra el resultado cuando solamente uno de los pulsadores está presionado. En ambos casos la expresión es verdadera y por lo tanto será ejecutado el código en el cuerpo del **if**.

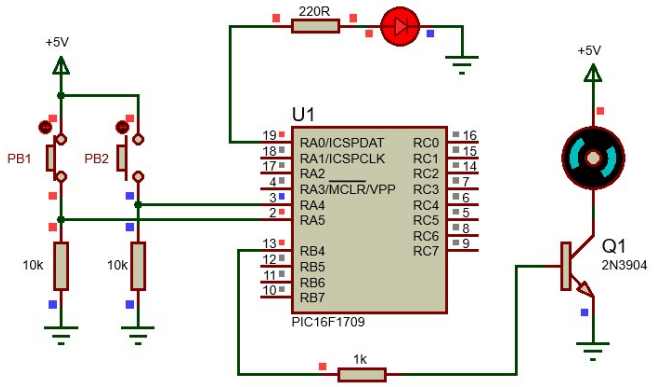

Figura 45. Solamente PB1 es presionado

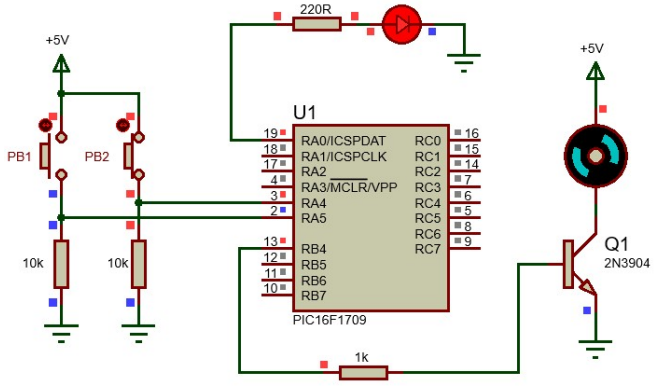

Figura 46. Solamente PB2 es presionado

En la figura 47, ambos pulsadores son presionados y se puede apreciar como el pin RB4 está rojo indicando que hay 5V. El LED está encendido y el motor girando lo cual puede ser visto en el video.

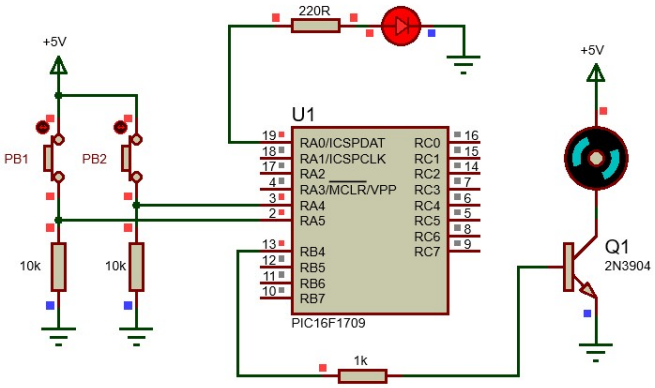

Figura 47. Ambos pulsadores están presionados

Se ponen a disposición del estudiante el esquema del circuito para su simulación en PROTUES y el video de la simulación para ver los pasos seguidos en la verificación.

- El esquema para el ejemplo #2 encuentra en UNIDADII – PROTEUS – ESQUEMAS – UII_EJ_3.
- El video para el ejemplo #1 se encuentra en UNIDADII – PROTEUS – VIDEOS – UII_EJ_3

Resumen en este punto del camino

Se desarrollaron tres ejemplos relacionados con el manejo de los puertos del microcontrolador PIC16F1709. Los ejemplos han permitido presentar algunos recursos tanto de hardware como de software.

En hardware fueron introducidos el resistor, el pulsador, el diodo LED, el transistor BJT y el motor DC.

En software se presentaron varios operadores básicos tales como el operador de asignación(=), el operador relacional “igual a”(==), los operadores lógicos AND (&&) y OR (||). También fueron introducidos elementos para la toma de decisiones como el if-else y para la implementación de lazos como el while.
Un formato de la estructura de un programa para un PIC fue presentado.

La efectividad de los programas se utilizaron el simulador de PROTEUS y el EASYPIC v7.

Ejemplo #4: Banda Transportadora

1. Descripción del proyecto

Se ha solicitado un sistema basado en un microcontrolador para controlar una banda transportadora, movida por un motor DC, usada para mover cajas entre dos puntos de la planta de producción. El funcionamiento deseado es el siguiente:

- Al presionar un pulsador (**START**) la banda debe ser activada, MOTOR energizado.
- El **MOTOR** deberá ser desenergizado cuando un segundo pulsador (**STOP**) sea presionado o cuando el número de cajas alcance el valor indicado. Para la detección de las cajas deberá utilizarse un **SENSOR** capacitivo.
- Si el MOTOR se detiene debido a que se ha alcanzado el número de cajas, se deberá activar un **LED**.
- Si el MOTOR se detiene debido a que se presionó el STOP deberá mantenerse el conteo y el proceso deberá continuar cuando se presione nuevamente START.
- Para realizar un nuevo proceso deberá presionarse START.

2. Diagrama de bloques del proyecto

La figura 48 muestra el diagrama de bloques del proyecto. El sensor es un nuevo elemento de hardware.

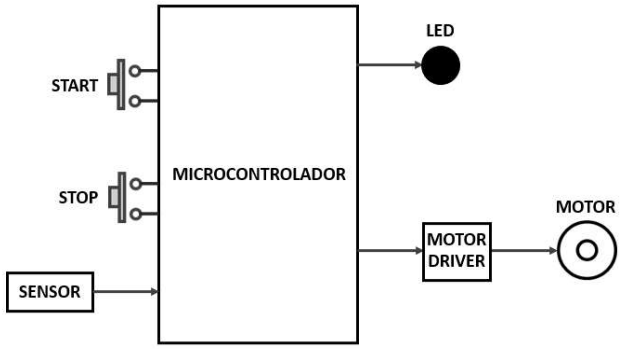

Figura 48. Diagrama de bloques banda trasportadora

3. Diagrama de circuito del proyecto

En la figura 49 muestra un sensor capacitivo en configuración PNP. El sensor es alimentado con 24VDC y, para poder utilizarlo con el microcontrolador es necesario convertir los 24VDC a su salida a 5V para satisfacer las características de entrada de este.

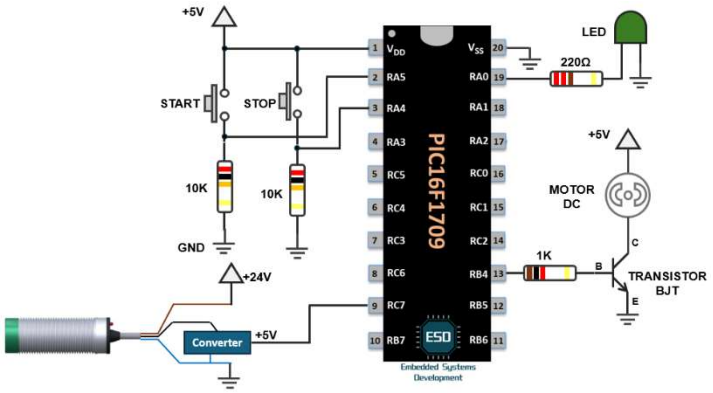

Figura 49. Diagrama de bloques banda trasportadora

PROTEUS no cuenta con un modelo para este tipo de sensor y para verificar la efectividad del programa utilizaremos, para simular el sistema, la configuración mostrada en la parte izquierda inferior del circuito de la figura 50.

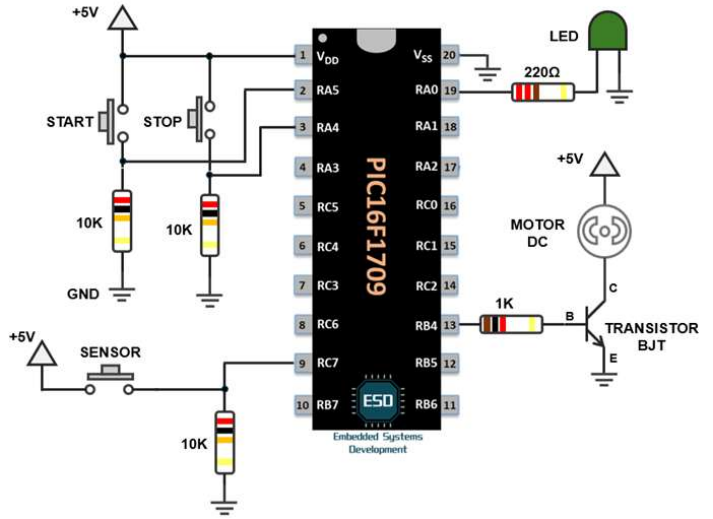

Figura 50. Diagrama de bloques banda trasportadora

4. Descripción del hardware

El único elemento nuevo en el ejemplo #4 es el sensor capacitivo. A continuación, una descripción breve del sensor.

- **Sensor Capacitivo**

Un sensor capacitivo es un dispositivo electrónico que puede detectar sólidos o líquidos sin contacto físico. Para detectar los objetivos, los sensores capacitivos emiten un campo eléctrico desde uno de los extremos del dispositivo. Cualquier objetivo que pueda distorsionar el campo eléctrico puede ser detectado por un sensor capacitivo.

Figura 49. Conexiones Sensor capacitivo

En la figura 51, se utiliza la salida normalmente abierta. Cuando la botella es detectada por el sensor en el cable negro tendremos 24VDC y para poder utilizar el sensor con el microcontrolador dicho voltaje debe ser reducido a 5VDC.

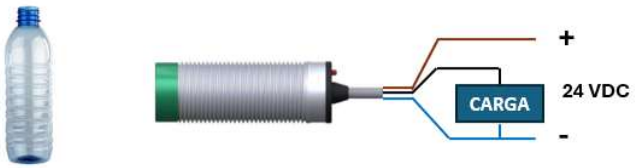

Figura 51. Conexiones Sensor capacitivo

Durante el desarrollo del ejemplo, en la clase, se presentarán más detalles sobre los sensores capacitivos.

5. Lenguaje Descripción del Programa (PDL)

En el ejemplo #4 es necesario llevar el conteo de las cajas movidas por la banda transportadora y debemos, como se muestra en el PDL, declarar e inicializar una variable para tal fin.

El PDL indica que el estado del pulsador START debe ser indagado repetidamente y de eso se encarga el **DO-FOREVER**. Si START es presionado se activa el MOTOR y se apaga el LED. Si START no es presionado, el motor debe permanecer desenergizado y el LED apagado.

Una vez presionado START y encendido el MOTOR, el sistema entra a un lazo condicional, establecido por el **REPEAT** – **UNTIL**, en el cual se interroga el estado del SENSOR de forma repetida. Si el sensor es presionado la variable conteo es incrementada en 1. El sistema saldrá del lazo cuando sea presionado STOP o sea alcanzado el número de cajas establecido. Si la salida es debido a que se alcanzó el número de cajas deseado el LED deberá ser encendido y la variable conteo puesta a cero.

Si el sistema sale del lazo porque STOP fue presionado, el programa se ejecuta nuevamente a partir del DO FOREVER interrogando el estado de START el cual debe ser presionado para reanudar el proceso.

```
START
    Definir variable conteo e inicializarla en cero
    Configurar RA4, RA5 y RC7 como entradas
    Configurar RA0 y RB4 como salidas
    Inicializar PORTA, PORTB y PORC en cero
    Configurar oscilador interno a 4MHz
    DO FOREVER
      IF START es presionado THEN
        Energizar MOTOR
        Apagar LED
        REPEAT
          IF SENSOR es activado THEN
            Incrementar variable conteo
          ENDIF
        UNTIL STOP presionado o Conteo igual a X
        IF conteo alcanzó valor deseado
          Encender LED
          Conteo igual a cero
        ENDIF
      ELSE
        Desenergizar MOTOR
        Apagar LED
      ENDIF
    ENDDO
END
```

6. Programa del Proyecto

La solución del ejemplo #4 requiere la utilización de una variable.

Variables en C

Una variable no es más que el nombre de una localidad de memoria que utilizamos para almacenar datos. En C el valor almacenado en la variable puede ser cambiada durante la ejecución del programa.

Cada variable en C tiene lo siguiente:

1. Un tipo específico, el cual determina el tamaño y estructura de la memoria de la variable.
2. El rango de valores que pueden ser almacenados en la memoria y
3. El conjunto de operaciones que pueden ser aplicadas a la variable.

El nombre de una variable puede estar compuesto de letras, dígitos, y el carácter guion bajo. Debe comenzar con una letra o con un dígito.

En dependencia de su alcance existen diferentes tipos de variables y serán presentadas en el momento que sea requerido. Algunos tipos de variables son los siguientes:

- Variables globales
- Variables locales
- Variables estáticas
- Variables externas
- Variables volátiles

Las variables deben ser declaradas antes de ser declaradas e inicializadas antes de ser utilizadas. La declaración le indica al compilador que existe una variable con el nombre y tipos especificados de forma que este pueda continuar con su trabajo de compilación sin necesidad de más detalles acerca de la variable.

Declaración de una variable:

Tipo_de_dato nombre_variable;

Inicialización de una variable:

nombre_variable=valor;

Se pueden combinar ambas cosas y presentar la variable de la siguiente manera:

uint8_t nombre_variable = 0;

En el ejemplo #4 declararemos e inicializaremos una variable que llamaremos "conteo" en la cual almacenaremos el valor del número de cajas movidas por la banda transportadora.

uint8_t conteo=0;

Lo anterior indica al compilador que existe una variable entera de 8-bit (máximo valor es 255) llamada "conteo".

En la primera parte del programa las palabras de configuración, el archivo de cabecera xc.h y las constantes simbólicas definidas con el objetivo de hacer más legible el código.

```
 8   #pragma config CONFIG1=0x08A4
 9   #pragma config CONFIG2=0x3883
10
11   #include <xc.h>
12
13   #define START   PORTAbits.RA5
14   #define STOP    PORTAbits.RA4
15   #define SENSOR  PORTCbits.RC7
16   #define MOTOR   PORTBbits.RB4
17   #define LED     PORTAbits.RA0
18   #define ON      1
19   #define OFF     0
```

En la segunda parte podemos apreciar la definición de la variable conteo y la inicialización de los puertos utilizados en la solución. Igual que en los otros ejemplos se utiliza el oscilador interno del microcontrolador configurado a 4MHz (línea 34).

```
20   void main(void)
21   {
22     uint8_t conteo=0;
23
24     TRISA=0x38;
25     ANSELA=0;
26     PORTA=0;
27     TRISB=0x00;
28     ANSELB=0;
29     PORTB=0;
30     TRISC=0x80;
31     ANSELC=0;
32     PORTC=0;
33
34     OSCCON=0x68;
```

Para implementar el DO FOREVER utilizamos el loop tal como se indica en el código. El enunciado for(;;) en la línea 36 indica que es un lazo infinito.

El REPEAT UNTIL fue implementado utilizando el do-while loop. Las instrucciones en el cuerpo del lazo son ejecutadas primero antes de chequear la condición. Si la condición es falsa, el do-while es ejecutado una vez.

```
do
{
 statements;
}while(condición);
```

En la línea 44 el estado del SENSOR es chequeado y mientras no haya una caja presente la variable conteo no será incrementada. El lazo será ejecutado mientras la variable conteo no alcance el valor indicado, 5 en el ejemplo, o STOP no sea presionado.

```
36      for(;;)
37      {
38        if(START==1)
39        {
40          MOTOR=ON;
41          LED=OFF;
42          do
43          {
44            if(SENSOR==1)
45            {
46              while(SENSOR==1);
47              conteo++;
48            }
49          }while((conteo<5)&&(STOP==0));
50
51          if(conteo==5)
52          {
53            LED=ON;
54            conteo=0;
55          }
56        }
57        else
58        {
59          MOTOR=OFF;
60        }
61      }
62    }
```

En la línea 51, si el valor de conteo establecido es alcanzado el LED es encendido y la variable conteo es puesta a cero.

7. Efectividad del Programa

La efectividad del programa será verificada en primer lugar utilizando el simulador PROTEUS. Luego será utilizado el EASYPIC v7. La figura 52 muestra el esquema del circuito del ejemplo #5.

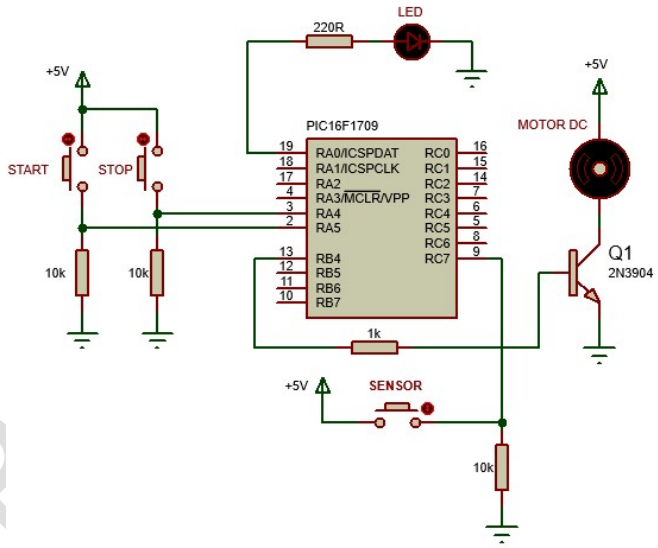

Figura 52. Esquema PROTEUS, banda transportadora

Para observar el incremento de la variable conteo podemos agregar el código requerido, después de conteo++ en la línea 47, para ver el valor en los pines <RC3:RC4> del PORTC.

PORTC=(0xFF) & (conteo);

El programa completo se encuentra al final del tema Entradas y Salidas digitales.

- El esquema para el ejemplo #4 encuentra en UNIDADII – PROTEUS – ESQUEMAS – UII_EJ_4.

- El video para el ejemplo #4 se encuentra en UNIDADII – PROTEUS – VIDEOS – UII_EJ_4

Con el ejemplo #4 damos por finalizado el tema de la unidad II "Entradas y Salidas Digitales." En el próximo tema será considerada la temporización y su importancia en los sistemas embebidos.