



**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**  
**FACULTAD DE INGENIERÍA**  
**DIVISIÓN DE INGENIERÍA ELÉCTRICA**  
**DEPARTAMENTO DE INGENIERÍA EN COMPUTACIÓN**

**PRACTICAS**  
**LABORATORIO DE MICROCOMPUTADORAS**

**Versión Simulación**  
**PROTEUS**

RUBÉN ANAYA GARCÍA  
JOSE ANTONIO ARREDONDO GARZA  
ANTONIO SALVA CALLEJA  
MOISES MELENDEZ REYES  
ANGELICA QUIÑONES JUAREZ  
DIANA CRUZ HERNANDEZ  
AMARANTO DAVILA JAUREGUI  
LUIS SERGIO DURÁN ARENAS

Trabajo realizado con el apoyo del programa UNAM-DGAPAPAPIME al proyecto **PE109719**  
“Microcomputadoras”

## **Introducción**

Atendiendo la situación actual que se vive a nivel mundial, y de acuerdo a las modalidades existentes en lo académico, para que sea posible la continuidad de su preparación académica, se propone la realización de prácticas utilizando el simulador PROTEUS, el cual es una herramienta de software útil ya que permite la ejecución de los programas a nivel de simulación; con algunas pequeñas modificaciones se podrá ejecutar los programas de forma física.

Este manual contiene 12 prácticas, en cada una de ellas se presenta la información introductoria, que servirá de antecedente al inicio de la realización de cada sesión y con esta forma clarificar los objetivos de la práctica.

En el manual se usarán los microcontroladores PIC16F877, del ATMEGA328 contenido en la plataforma Arduino y de la plataforma Raspberry Pi; los cuales complementarán la parte práctica, los ejercicios aquí propuestos inducen y llevan al alumno a un mejor entendimiento de estos dispositivos y a un mayor aprovechamiento de los mismos.

Se iniciará con la familiarización del lenguaje ensamblador del PIC y la utilización del ambiente MPLAB, donde se editará, ensamblará y simularán los programas resueltos, además se cubrirá el concepto de modos de direccionamiento; esta práctica se enfoca en la programación en el direccionamiento directo. La práctica dos servirá como reforzamiento de los conceptos en la práctica anterior, empleando el modo de direccionamiento indirecto.

Para la presentación de las actividades propuestas, a partir de la practica 3 se utilizará la herramienta de software PROTEUS para simular el funcionamiento de los ejercicios; el alumnado tendrá la habilidad de interpretar el circuito esquemático para trasladar al alambrado del mismo; además se emplearán puertos paralelos en la modalidad de salida.

La práctica cuatro percibirá entradas y generará salidas a través de diferentes puertos paralelos.

En la práctica cinco se controlarán motores de corriente directa, motores a pasos y servomotores, además se entenderá la importancia del uso de drivers de potencia.

La práctica seis mostrará al alumno las ventajas de contar con un convertidor analógico-digital dentro de los recursos del microcontrolador, además de ampliar las posibilidades de aplicación de este recurso.

En la práctica siete aplicará la comunicación serie en la modalidad asíncrona, con la finalidad que el alumno controle las funciones del PIC por medio de otro dispositivo serie, que puede ser la computadora personal o un dispositivo móvil con conexión inalámbrica.

La práctica 8 se enfocará su aprendizaje en el desarrollo de algoritmos en programación en lenguajes de alto nivel; se empleará el compilador de C usando las funciones disponibles para controlar los puertos paralelos y el puerto serie asíncrono.

La práctica nueve estudiará la comunicación serie síncrona a través del protocolo SPI e I2C.

En la práctica diez, se estudiarán las funciones de los temporizadores, la programación de las interrupciones y la modulación de ancho de pulso (PWM).

En la práctica once se estudiará la plataforma Arduino y el software del mismo nombre. Finalmente, la práctica doce hará uso de la plataforma Raspberry Pi.

Al concluir las prácticas, el alumno deberá haber comprendido las ventajas que se tiene al realizar aplicaciones con microcomputadoras así mismo describir los diferentes elementos constituidos de éstas.

### Contenido

	<i>Contenido</i>
Práctica No. 1	Introducción a la programación del microcontrolador PIC16F877; “Direccionamiento Directo”
Práctica No. 2	Programación en ensamblador; “ <i>Direccionamiento Indirecto</i> ”
Práctica No. 3	Sistema mínimo
Práctica No. 4	Puertos paralelos Entrada/Salida
Práctica No. 5	Control de actuadores
Práctica No. 6	Convertidor Analógico Digital
Práctica No. 7	Comunicación Serie Asíncrona
Práctica No. 8	Programación en C Puertos paralelos E/S, Puerto Serie Asíncrono
Practica No. 9	Programación en C; Comunicación Serie Síncrona SPI e I2C
Práctica No. 10	Programación en C Temporizadores, interrupciones y PWM
Practica No. 11	Plataforma Arduino
Práctica No. 12	Plataforma Raspberry Pi
Evaluación	Proyecto Final

Las prácticas anteriormente descritas cubrirán el contenido del temario de la teoría de Microcomputadoras; así como la aportación a los atributos de egreso en los que impacta de manera directa.

---

### **Objetivo de la teoría de Microcomputadoras**

El alumno aprenderá y aplicará los conocimientos de la teoría y funcionamiento de los microprocesadores y su interconexión con diferentes circuitos periféricos para la construcción y funcionamiento de microcomputadoras. Diseñará y construirá aplicaciones utilizando microprocesadores y sus periféricos para diferentes sistemas, simulando aplicaciones industriales en tiempo real, así como aplicaciones científicas.

Cubriendo el siguiente temario

- 
1. Conceptos básicos
  2. Conjunto de instrucciones
  3. Modos de direccionamiento
  4. Señales de control y diseño de un sistema con microprocesadores
  5. Periféricos e interfaces con microprocesadores
  6. Técnicas de diseño de sistemas con microprocesadores
  7. Características generales de microprocesadores de 16 y 32 bits
- 

### **Atributos de egreso**

Con el desarrollo de las prácticas, los proyectos, así como de los reportes de estas actividades aportará en:

- 
- a. Analizar y aplicar soluciones empleando conocimientos de diseño de ingeniería en computación que resulten en proyectos que satisfacen requerimientos específicos.
  - b. Diseñar, experimentar, procesar datos e interpretar resultados para establecer conclusiones basadas en su formación de ingeniero.
  - c. Comunicarse efectivamente de manera oral y escrita ante distintas audiencias o interlocutores.
-

**Laboratorio de Microcomputadoras**  
**Práctica No. 1**  
**Introducción General al Microcontrolador PIC16F877**

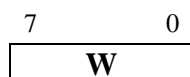
**Objetivo.** Familiarizar al alumno en el conocimiento del ensamblador, el simulador, el conjunto de instrucciones de un microcontrolador y ejecutar programas en tiempo de simulación.

### Introducción

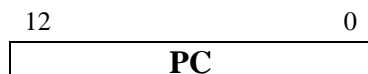
Algunas de las características más importantes del microcontrolador son:

- 8K de memoria FLASH
- 368 bytes de memoria RAM
- 255 bytes de memoria EEPROM
- 35 instrucciones
- 5 puertos paralelos (A, B, C, D, E)
- Convertidor Analógico Digital
- Comunicación Serie Asíncrona
- Comunicación Serie Síncrona (SPI, I2C)
- Tres módulos temporizadores
- Dos módulos CCP que pueden operar como Comparación, Captura o PWM
- 14 fuentes de interrupción

Los registros disponibles para el programador son:

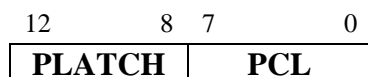


*Registro de trabajo W*



*Registro Contador de Programa*

El registro PC se forma con dos registros de ocho bits:



*Registro de banderas*

IRP	Selecciona el banco de memoria RAM en direccionamiento indirecto en conjunto con la bandera más significativa del registro FSR.
RP1, RPO	Seleccionan el banco de memoria RAM en direccionamiento directo.
TO,PD	Banderas de interrupciones.
Z	Indica el estado del resultado de la última operación, si Z=1 el resultado fue cero y si Z=0 el resultado previo fue diferente de cero.
DC	Indica si existió un medio acarreo.
C	Indica si fue generado in acarreo como resultado de la última operación; si C=1 existió un acarreo y si C=0 no existió acarreo.

El mapa de memoria y distribución de registros se muestra en la figura 1.2.

00H	<b>INDF</b>	80H	<b>INDF</b>	100H	<b>INDF</b>	180H	<b>INDF</b>
01H	<b>TMR0</b>	81H	<b>OPTION_REG</b>	101H	<b>TMR0</b>	181H	<b>OPTION_REG</b>
02H	<b>PCL</b>	82H	<b>PCL</b>	102H	<b>PCL</b>	182H	<b>PCL</b>
03H	<b>STATUS</b>	83H	<b>STATUS</b>	103H	<b>STATUS</b>	183H	<b>STATUS</b>
04H	<b>FSR</b>	84H	<b>FSR</b>	104H	<b>FSR</b>	184H	<b>FSR</b>
05H	<b>PORTA</b>	85H	<b>TRISA</b>	105H		185H	
06H	<b>PORTB</b>	86H	<b>TRISB</b>	106H	<b>PORTB</b>	186H	<b>TRISB</b>
07H	<b>PORTC</b>	87H	<b>TRISC</b>	107H		187H	
09H	<b>PORTD</b>	89H	<b>TRISD</b>	109H		189H	
09H	<b>PORTE</b>	89H	<b>TRISE</b>	109H		189H	
0AH	<b>PCLATCH</b>	8AH	<b>PCLATCH</b>	10AH	<b>PCLATCH</b>	18AH	<b>PCLATCH</b>
0BH	<b>INTCON</b>	8BH	<b>INTCON</b>	10BH	<b>INTCON</b>	18BH	<b>INTCON</b>
0CH	<b>PIR1</b>	8CH	<b>PIE1</b>	10CH	<b>EEDATA</b>	18CH	<b>EECON1</b>
0DH	<b>PIR2</b>	8DH	<b>PIE2</b>	10DH	<b>EEADR</b>	18DH	<b>EECON2</b>
0EH	<b>TMR1L</b>	8EH	<b>PCON</b>	10EH	<b>EEDATH</b>	18EH	
0FH	<b>TMR1H</b>	8FH		10FH	<b>EEADRH</b>	18FH	
10H	<b>T1CON</b>	90H		110H		1A0H	
11H	<b>TMR2</b>	91H	<b>SSPCON2</b>	111H		1A1H	
12H	<b>T2CON</b>	92H	<b>PR2</b>	112H		1A2H	
13H	<b>SSPBUF</b>	93H	<b>SSPADD</b>	113H		1A3H	
14H	<b>SSPCON</b>	94H	<b>SSPSTAT</b>	114H		1A4H	
15H	<b>CCPR1L</b>	95H		115H		1A5H	
16H	<b>CCPR1H</b>	96H		116H		1A6H	
17H	<b>CCPCON1</b>	97H		117H		1A7H	
18H	<b>RCSTA</b>	98H	<b>TXSTA</b>	118H		1A8H	
19H	<b>TXREG</b>	99H	<b>SPBRG</b>	119H		1A9H	
1AH	<b>RCREG</b>	9AH		11AH		1AAH	
1BH	<b>CCPR2L</b>	9BH		11BH		1ABH	
1CH	<b>CCPR2H</b>	9CH		11CH		1ACH	
1DH	<b>CCP2CON</b>	9DH		11DH		1ADH	
1EH	<b>ADRESH</b>	9EH	<b>ADRESL</b>	11EH		1AEH	
1FH	<b>ADCON0</b>	9FH	<b>ADCON1</b>	11FH		1AFH	
20H		A0H		120H		1A0H	
	<b>96 REGISTROS DE PROPÓSITO GENERAL</b>		<b>80 REGISTROS DE PROPÓSITO GENERAL</b>		<b>80 REGISTROS DE PROPÓSITO GENERAL</b>		<b>80 REGISTROS DE PROPÓSITO GENERAL</b>
		EFH		16FH		1EF	
		F0H		170H		1F0	
7FH		FFH		17FH		1FFH	
<b>Banco 0</b>		<b>Banco 1</b>		<b>Banco 2</b>		<b>Banco 3</b>	

No disponible

Figura 1.2 Mapa de memoria de datos



## Memoria de Programa

Memoria Flash de 8K x 14 bits, cuya función será almacenar los programas que ejecutará el procesador; el mapa de memoria es el siguiente:

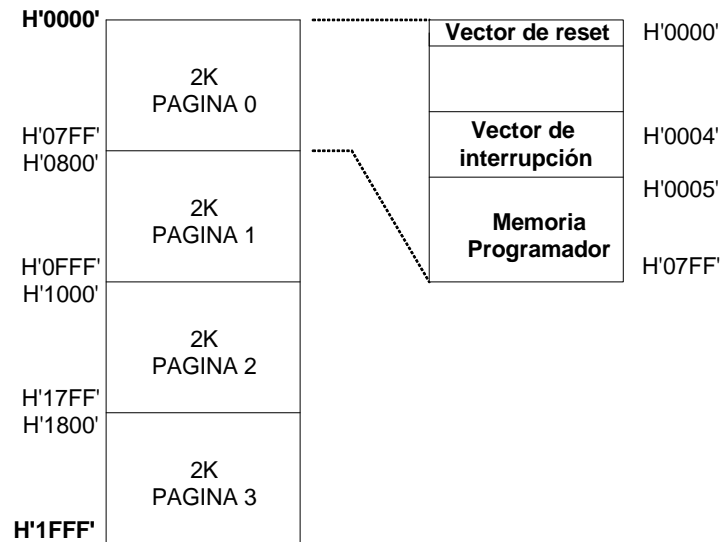


Figura 1.3 Mapa de memoria de programa

De acuerdo a lo anterior; el programa en ensamblador deberá cumplir el siguiente formato.

Etiquetas	Directivas y/o Instrucciones	Comentarios
	PROCESSOR 16f877	<i>; Indica la versión de procesador</i>
	INCLUDE <p16f877.inc>	<i>; Incluye la librería de la versión del procesador</i>
	<b>ORG 0</b>	<i>; Especifica un origen (vector de <b>reset</b>)</i>
	<b>GOTO INICIO</b>	<i>; Código del programa</i>
	<b>ORG 5</b>	<i>; Indica origen para inicio del programa</i>
<b>INICIO:</b>	INSTRUCCIÓN 1 :	<i>; Instrucciones que se ejecutan una sola vez</i>
<b>LOOP:</b>	INSTRUCCIÓN N	<i>; Instrucciones que se repiten</i>
	INSTRUCCION M <b>GOTO LOOP</b>	<i>; repite el ciclo (salta a la etiqueta LOOP)</i>
	<b>END</b>	<i>; Directiva de fin de programa</i>

La gama media de la familia de los PIC's tiene el siguiente conjunto de instrucciones.

Mnemonic, Operands	Description	Cycles	14-Bit Opcode			Status Affected
			MSb		LSb	
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d Add W and f	1	00	0111	dfff ffff	C,DC,Z
ANDWF	f, d AND W with f	1	00	0101	dfff ffff	Z
CLRF	f Clear f	1	00	0001	1fff ffff	Z
CLRWF	- Clear W	1	00	0001	0xxxx xxxxx	Z
COMF	f, d Complement f	1	00	1001	dfff ffff	Z
DECF	f, d Decrement f	1	00	0011	dfff ffff	Z
DECFSZ	f, d Decrement f, Skip if 0	1(2)	00	1011	dfff ffff	
INCF	f, d Increment f	1	00	1010	dfff ffff	Z
INCFSZ	f, d Increment f, Skip if 0	1(2)	00	1111	dfff ffff	
IORWF	f, d Inclusive OR W with f	1	00	0100	dfff ffff	Z
MOVF	f, d Move f	1	00	1000	dfff ffff	Z
MOVWF	f Move W to f	1	00	0000	1fff ffff	
NOP	- No Operation	1	00	0000	0xxxx 0000	
RLF	f, d Rotate Left f through Carry	1	00	1101	dfff ffff	C
RRF	f, d Rotate Right f through Carry	1	00	1100	dfff ffff	C
SUBWF	f, d Subtract W from f	1	00	0010	dfff ffff	C,DC,Z
SWAPF	f, d Swap nibbles in f	1	00	1110	dfff ffff	
XORWF	f, d Exclusive OR W with f	1	00	0110	dfff ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b Bit Clear f	1	01	00bb	bfff ffff	
BSF	f, b Bit Set f	1	01	01bb	bfff ffff	
BTFSC	f, b Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff ffff	
BTFSS	f, b Bit Test f, Skip if Set	1 (2)	01	11bb	bfff ffff	
LITERAL AND CONTROL OPERATIONS						
ADDLW	k Add literal and W	1	11	111x	kkkk kkkk	C,DC,Z
ANDLW	k AND literal with W	1	11	1001	kkkk kkkk	Z
CALL	k Call subroutine	2	10	0kkk	kkkk kkkk	
CLRWDI	- Clear Watchdog Timer	1	00	0000	0110 0100	TO,PD
GOTO	k Go to address	2	10	1kkk	kkkk kkkk	
IORLW	k Inclusive OR literal with W	1	11	1000	kkkk kkkk	Z
MOVLW	k Move literal to W	1	11	00xx	kkkk kkkk	
RETFIE	- Return from interrupt	2	00	0000	0000 1001	
RETLW	k Return with literal in W	2	11	01xx	kkkk kkkk	
RETURN	- Return from Subroutine	2	00	0000	0000 1000	
SLEEP	- Go into standby mode	1	00	0000	0110 0011	TO,PD
SUBLW	k Subtract W from literal	1	11	110x	kkkk kkkk	C,DC,Z
XORLW	k Exclusive OR literal with W	1	11	1010	kkkk kkkk	Z

Figura 1.4 Conjunto de instrucciones del PIC 16F877

## Descripción de las instrucciones

### a. Instrucciones orientadas a registros

Formato: **INSTRUCCIÓN F,D**

INSTRUCCIÓN Corresponde al mnemónico de la instrucción a realizar.

F Es el registro a operar en la instrucción.

D Es el destino de la instrucción.

Donde, si:

D = 1, F, f ,DEF\_REG ó ; el resultado se almacena en el registro.  
nada

Ejemplo:

ADDWF **OX20**

ADDWF OX30,**1**

ADDWF OX40,**F**

ADDWF REG,**REG**

ADDWF **REG**

D = W, w ó 0 ; el resultado se almacena en el registro W.

Ejemplo:

ADDWF OX20,**0**

ADDWF OX30,**W**

ADDWF REG,**W**

ADDWF REG,**w**

### b. Instrucciones orientadas a manejo de bits

Formato: **INSTRUCCIÓN F,B**

INSTRUCCIÓN Corresponde al mnemónico de la instrucción a ejecutar.

F Es el registro a utilizar en la instrucción.

B Es la posición del bit en el registro (0,1,2,3,4,5,6,7).

<i>MSB</i>				<i>LSB</i>			
7	6	5	4	3	2	1	0

Ejemplo:

BCF 0X20,0 ; Coloca en 0 al bit 0 del registro 0x20.

BSF REG,7 ; Coloca en 1 al bit 7 del registro definido como REG.

BTFSC REG,2 ;Prueba el bit 2 del registro REG y salta una instrucción si este  
INST\_FALSA bit es cero.  
**INST\_VERDADERA**

### c. Instrucciones orientadas a manejo de constantes e instrucciones de control

Formato: **INSTRUCCIÓN K**

INSTRUCCION Corresponde al mnemónico de la instrucción a realizar.

K Constante a operar con la instrucción

MOVLW H'10' ; Mueve el número 10 en hexadecimal al registro W.

GOTO LOOP ; Cambia el control del programa a la ubicación de la etiqueta LOOP.

CALL RET ; Ejecuta la subrutina RET (el procesador almacena en la pila la dirección de la siguiente instrucción).

RETURN ; Regresa de la subrutina (recupera de la pila la dirección de la instrucción pendiente).

### Modos de direccionamiento

Esta versión de microcontrolador dispone de dos modos de direccionamiento:

- Directo
- Indirecto

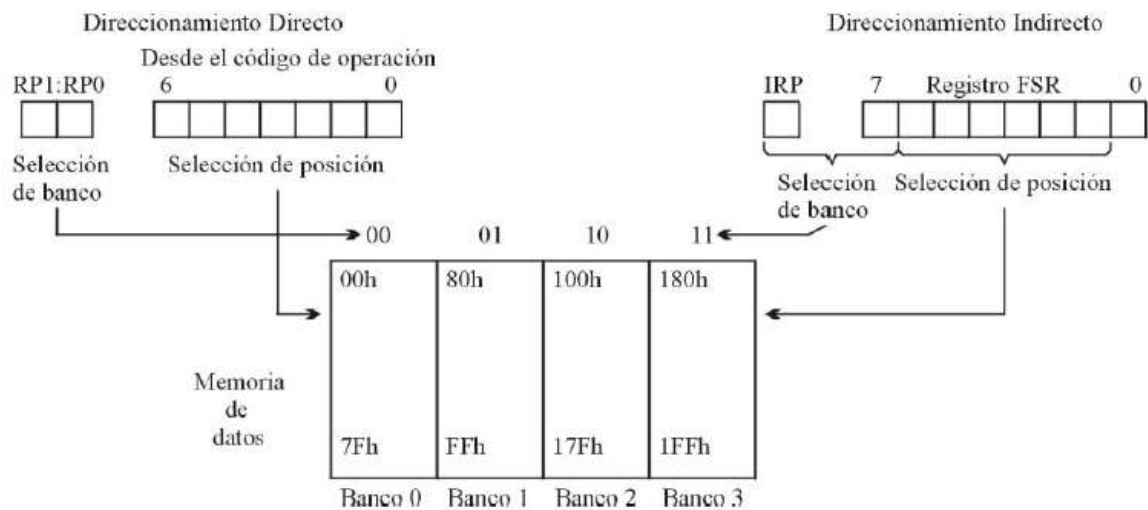


Figura 1.5 Modos de direccionamiento

Direccionamiento directo

La dirección de memoria estará indicada por el valor de las banderas RP1 y RP0 accediendo al banco deseado; así como la dirección de memoria (registro) indicado en la instrucción.

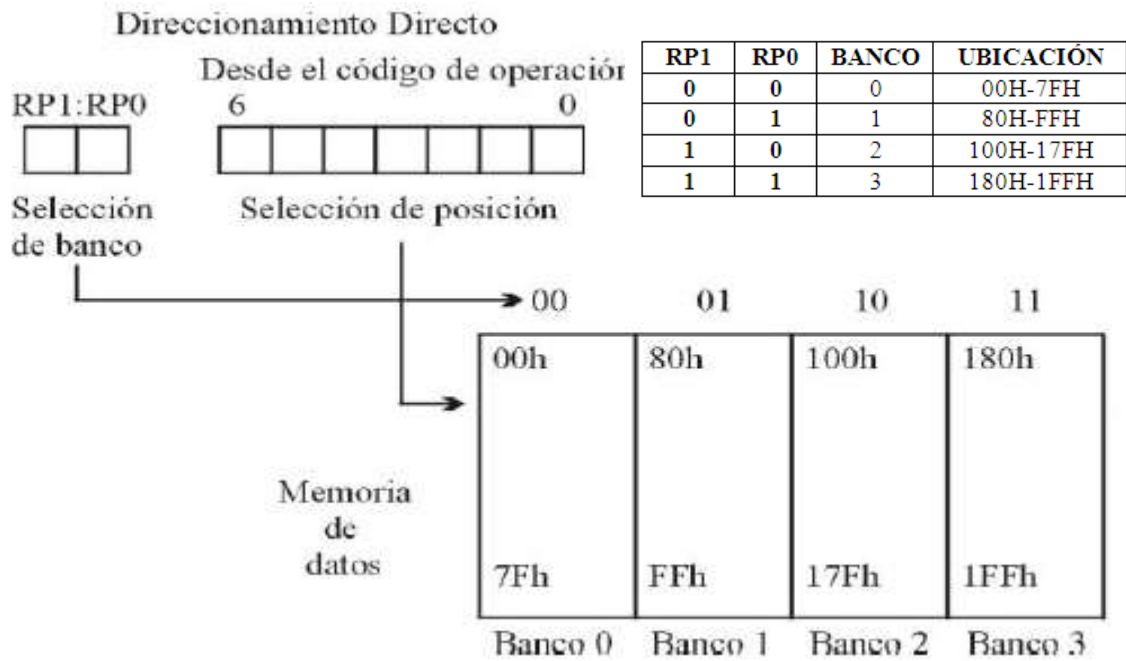


Figura 1.6 Direccionamiento directo

Ejemplo:

```
MOVWF 0X20      ; Mueve el contenido del registro W a la dirección del registro 0x20
INCF 0X20        ; Incrementa el contenido del registro 0X20
DECF 0X20        ; Decrementa el contenido del registro 0X20
```

*Nota: En caso que  $RP0=0$  y  $RP1=0$  el contenido de W será enviado a la dirección 0x20 del banco 0; alguna otra codificación posicionará el dato en el banco seleccionado.*

## Herramienta de desarrollo MPLAB

El MPLAB es un Ambiente de Desarrollo Integrado **IDE**, que permite escribir, ensamblar y simular un programa, e incluso, usando determinado hardware, se puede simular en circuito y programar al microcontrolador. Este **IDE** lo puedes bajar de manera gratuita de la dirección electrónica de Microchip ([www.microchip.com](http://www.microchip.com)).

Al ejecutar MPLAB, presenta una pantalla como la siguiente:

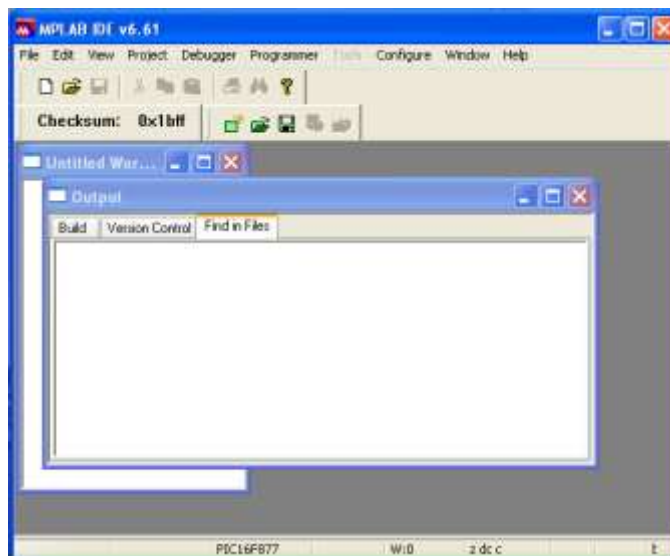


Figura 1.7 Entorno de MPLAB

En el menú **File** seleccionar **New**, entonces aparece la ventana de trabajo con el encabezado **Untitled**, escribir el programa en esta área, una vez terminado, salvarlo usando nuevamente el menú **File** y el submenú **Save as** del tipo **ASM**.

Para ensamblar el programa usar el comando **Project**, buscar el submenú **Quickbuild**, donde aparecerá incluido el nombre del programa a ensamblar que es el que está activo en el área de captura.



Figura 1.8 Ensamblar un programa

Al no existir errores de sintaxis, se genera el mensaje **BUILD SUCCEEDED**, lo cuál indica que el proceso de ensamblado ha concluido satisfactoriamente.

El siguiente proceso será simular el programa, para lo cuál del menú se elige el comando **View** y las opciones requeridas.

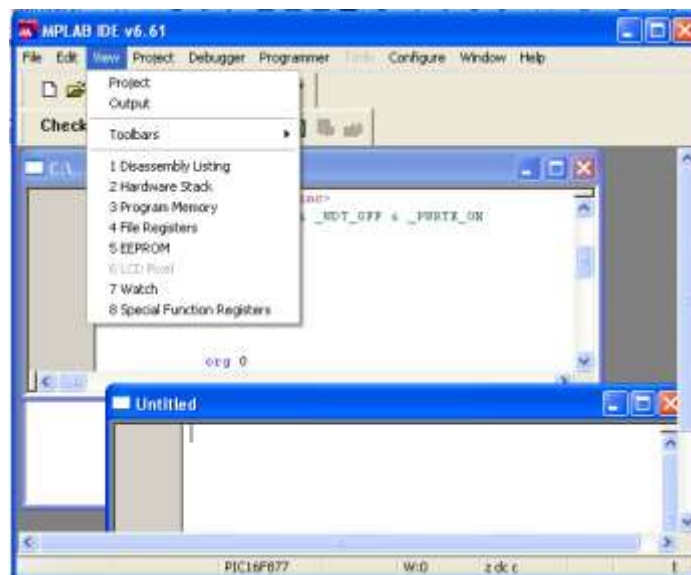


Figura 1.9 Selección de ventanas de visualización para el proceso de simulación

Por lo general solo se selecciona **File Registers**, el cuál muestra los registros y sus valores actuales; para modificar el contenido de alguna localidad, sólo se tiene que escribir el valor deseado y si el programa genera un valor, este será actualizado.

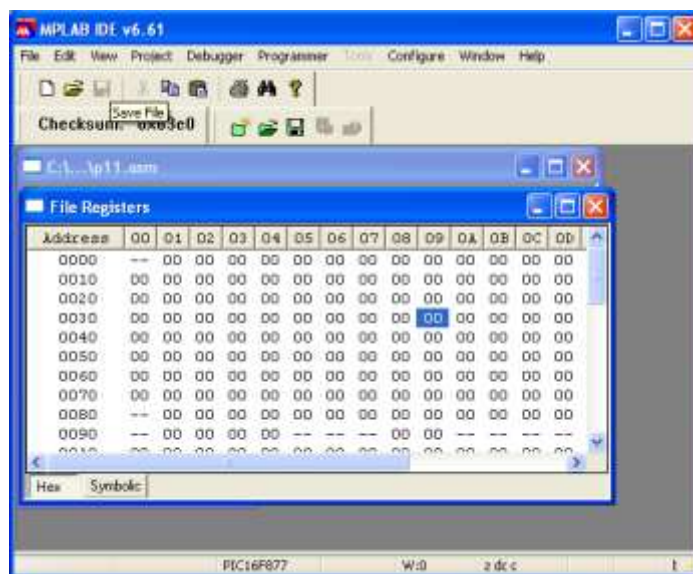


Figura 1.10 Mapa de memoria RAM

Para iniciar el proceso de simulación se debe seleccionar el simulador **MPLAB SIM**, accediendo al menú principal, dar click en **Debugger**, luego seleccionar **Select Tool** y entonces **Mplab Sim**; se habilitarán los iconos de simulación.

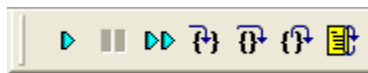


Figura 1.11 Iconos de simulación

Permitirá iniciar el proceso de simulación por instrucción o en forma continua, también es posible simular usando teclas de función, acceder al comando Debugger del menú principal.

Nota: Es importante mencionar que las instrucciones y directivas pueden ser escritas en mayúsculas y minúsculas de manera indistintas, no así las etiquetas y variables que sean declaradas por el alumnos; estas deben ser usadas tal como hayan sido definidas. En el caso de usar las definidas dentro del archivo PIC16F877.INC, deben ser referidas en mayúsculas.



**Desarrollo de la práctica 1:**

Para cada uno de los siguientes ejercicios, realizar los programas solicitados y simular el funcionamiento de ellos.

1.- Siguiendo las indicaciones previas, escribir el siguiente programa, ensamblar y simular el funcionamiento de este.

```
PROCESSOR 16f877
INCLUDE <p16f877.inc>

K    equ    H'26'
L    equ    H'27'

      ORG    0
      GOTO   INICIO

INICIO: ORG    5
        MOVLW H'05'
        ADDWF K,0
        MOVWF L
        GOTO   INICIO
        END
```

Recomendaciones:

- Las etiquetas y la definición de variables en la primer columna.
- Las instrucciones deben iniciar a partir de la segunda columna.
- Las instrucciones y directivas pueden ser en mayúsculas o minúsculas de manera indistintas, no así las variables.

Ingresa un dato de 8 bits a la dirección reservada a la variable **K** y ejecutar la simulación del programa utilizando diferentes valores.

K	L
66	<b>6B</b>

2.- Escribir, ensamblar y ejecutar el siguiente programa:

```
PROCESSOR 16f877
INCLUDE <p16f877.inc>
```

```
K    equ    H'26'
L    equ    H'27'
M    equ    H'28'
```

```
ORG 0
GOTO INICIO
```

```
INICIO:  ORG 5
          MOVF K,W
          ADDWF L,0
          MOVWF M
          GOTO INICIO
          END
```

- Comentar e indicar que hace el programa
- Realizar la ejecución con diferentes valores en K y L
- Revisar el valor que se genera en la bandera C

K	L	M
50	55	A5

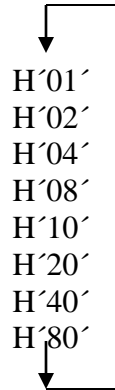
3.- Modificar el programa anterior, para que ahora los datos que operará se encuentren en las localidades **J** y **K** respectivamente, el resultado almacenarlo en otras direcciones, reservadas para **C1** y **R1**; C1 representa el valor de la bandera de acarreo y R1 el resultado.

Un ejemplo de datos y del resultado de la suma es:

J	K	C1	R1
FF	FF	01	FE

4.- Realice un programa que ejecute la siguiente secuencia, misma que deberá ver en la dirección de memoria (registro) de su elección.

Secuencia:



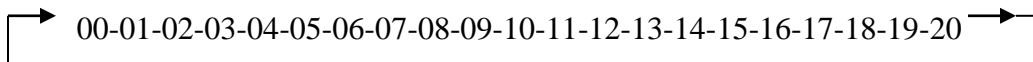
Donde H'01' indica que el dato esta dado en hexadecimal.

En caso de seleccionar el registro cuya dirección es 0X20

DIR	20	20	20	20	20	20	20
DATO	01	02	04	08	10	20	80

Nota: La secuencia indicada, deberá mostrarse en una misma dirección de memoria.

5.- Desarrollar un programa que presente la cuenta en numeración decimal en la localidad de memoria de su elección, como se indica a continuación.



Nota: La secuencia indicada, deberá mostrarse en una misma dirección de memoria, tal como fue realizado en el ejercicio anterior.

## Laboratorio de Microcomputadoras

## Práctica No. 2

## Programación en ensamblador direccionamiento indirecto

**Objetivo.** Analizar la programación en lenguaje ensamblador. Realizar algoritmos en lenguaje ensamblador empleando direccionamiento indirecto.

**Introducción**

Como fue mencionado en la práctica anterior, este procesador dispone de dos modos de direccionamiento, esta sesión centrará su atención en el empleo del modo de direccionamiento indirecto.

**Direccionamiento indirecto**

El banco de memoria RAM es seleccionado por la codificación de los bits más significativos de los registros STATUS (IRP) y FSR. La dirección dentro del banco será especificada por los bits restantes del registro FSR.

IRP <sub>STATUS</sub>	MSB <sub>FSR</sub>	Banco
0	0	0
0	1	1
1	0	2
1	1	3

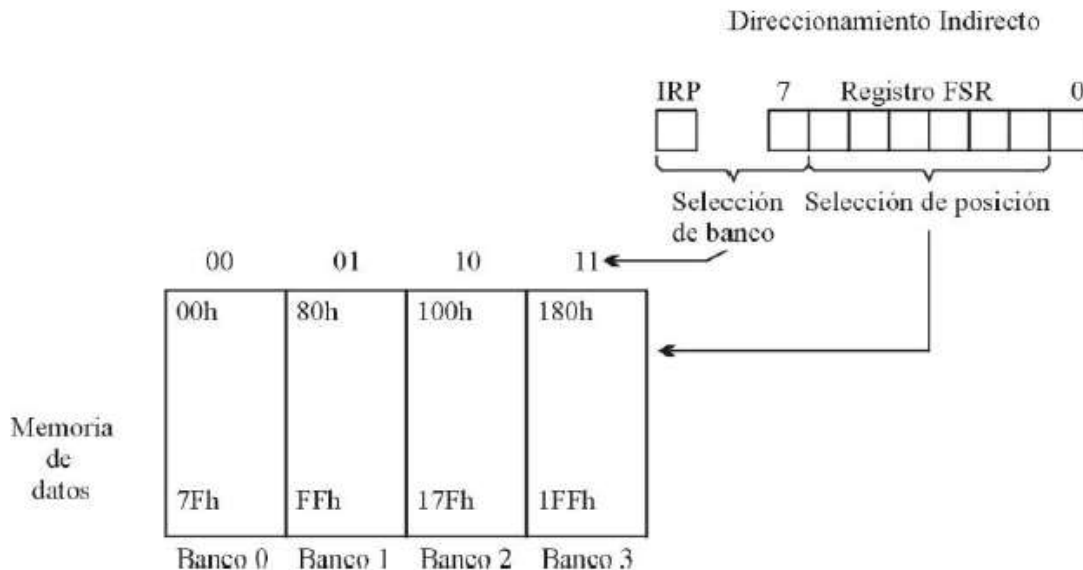


Figura 2.1 Direccionamiento indirecto

Para acceder a la dirección especificada por **FSR**, deberá ser indicando como parámetro de la instrucción al registro **INDF**.

Ejemplo:

MOVLW 0x20	; Carga un 0x20 al registro W
MOVWF <b>FSR</b>	; Mueve el contenido de W al registro FSR (FSR=0x20)
MOVLW 5	; Carga el valor 5 al registro W
MOVWF <b>INDF</b>	; Mueve el contenido de W a la dirección apuntada por ; el registro FSR; ahora la dirección de memoria 0x20 ; tendrá como contenido el valor 5.
INCF <b>FSR</b>	; incrementa al registro FSR (FSR=0x21)
MOVWF <b>INDF</b>	; en este caso el valor del registro W será almacenado en la localidad 0x21

Desarrollo de la práctica No. 2

1.- Escribir, comentar y ejecutar la simulación del siguiente programa:

```
PROCESSOR 16f877
INCLUDE <p16f877.inc>

ORG 0
GOTO INICIO

ORG 5
INICIO: BCF STATUS,RP1
        BSF STATUS,RP0
        MOVLW 0X20
        MOVWF FSR
LOOP:  MOVLW 0X5F
        MOVWF INDF
        INCF FSR
        BTFSS FSR,6
        GOTO LOOP
        GOTO $
        END
```

a. Describir el funcionamiento

2.- Elaborar un programa que encuentre el número **menor**, de un conjunto de datos ubicados entre las localidades de memoria 0x20 a 0X3F; mostrar el valor en la dirección **40H**.

Ejemplo:

Dirección	Dato
20	FF
21	FE
22	FD
23	FC
24	FB
25	FA
26	89
27	88
28	87
29	86
2A	85
2B	84
2C	83
2D	82
2E	81
2F	80

Dirección	Dato
30	6B
31	69
32	68
33	67
34	40
35	41
36	42
37	43
38	44
39	45
3A	46
3B	47
3C	48
3D	49
3E	90
3F	<b>01</b>

Dirección	Dato
40	01

3.- Desarrollar el algoritmo y el programa que ordene de manera ascendente un conjunto de datos ubicados en el banco 0 del registro 0X20 al 0X2F.

Ejemplo:

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
<b>0F</b>	<b>0E</b>	<b>0D</b>	<b>0C</b>	<b>0B</b>	<b>FF</b>	<b>09</b>	<b>08</b>	<b>07</b>	<b>06</b>	<b>05</b>	<b>04</b>	<b>03</b>	<b>02</b>	<b>0A</b>	<b>01</b>

Solución:

20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
<b>01</b>	<b>02</b>	<b>03</b>	<b>04</b>	<b>05</b>	<b>06</b>	<b>07</b>	<b>08</b>	<b>09</b>	<b>0A</b>	<b>0B</b>	<b>0C</b>	<b>0D</b>	<b>0E</b>	<b>0F</b>	<b>FF</b>

- Comprobar el funcionamiento de su programa con distintos conjuntos de datos.

## Laboratorio de Microcomputadoras

### Práctica No. 3

### Sistema mínimo microcontrolador PIC16F877

**Objetivo.** Desarrollar la habilidad de interpretación de esquemáticos. Conocer el diagrama del sistema mínimo del microcontrolador, el software de comunicación. Realizar aplicaciones con puertos paralelos en la modalidad de salida; ejecución de un programa en tiempo real.

### Introducción

Para ejecutar un programa en el procesador se debe alambrear el sistema mínimo, siguiendo el diagrama de la figura 3.1.

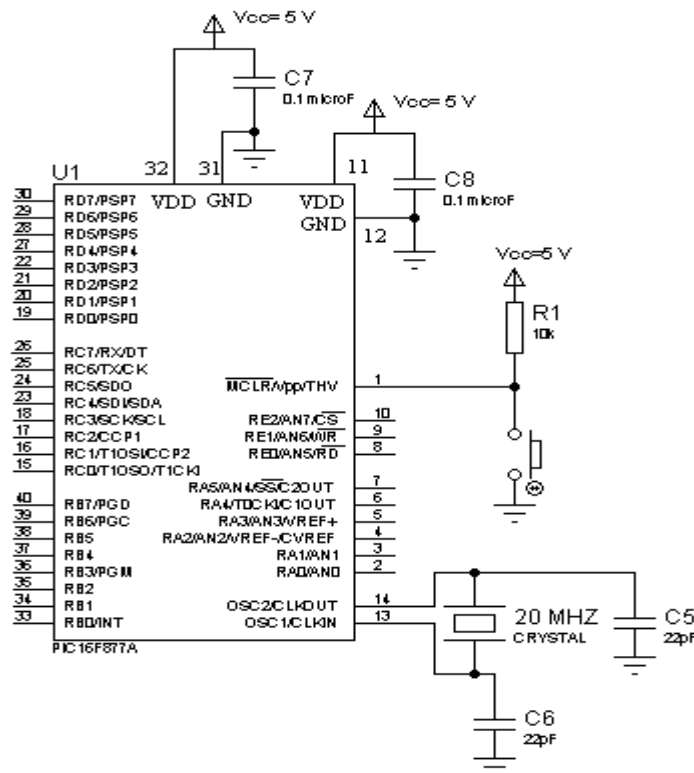


Figura 3.1 Sistema mínimo del microcontrolador PIC16F877(A)

Como se puede apreciar, el sistema requiere de tres módulos imprescindibles:

- Reloj;** formado por un cristal de cuarzo de 20 MHz y dos capacitores de 22 pF, cuyo objetivo es la generación de la frecuencia de operación externa.
- Circuito de reset;** formado por una resistencia y un push button; cuya finalidad es la generación del pulso en bajo para producir un reset en el sistema.
- La alimentación al sistema;** Vdd= 5 V y GND= 0V.





## Puertos Paralelos

El microcontrolador PIC tiene 5 puertos paralelos, denominados A, B, C, D y E, todos ellos se pueden configurar para operar como puerto de salida o entrada.

Puerto	Tamaño	Función	TRISX	PORTX
<b>A</b>	6	E/S	85H	05H
<b>B</b>	8	E/S	86H	06H
<b>C</b>	8	E/S	87H	07H
<b>D</b>	8	E/S	88H	08H
<b>E</b>	3	E/S	89H	09H

Al emplear un puerto paralelo, lo primero que se debe de hacer es configurar su función, esto se realiza en las posiciones de memoria RAM denominados **TRISX** los cuales están ubicados en el banco número 1. Una vez ubicado en este banco se realiza la configuración, bajo la siguiente convención.

'0'	Configura el bit del puerto como salida
'1'	Configura el bit del puerto como entrada

Después que se ha configura todo el puerto, regresar al banco cero para enviar o recibir información a través de los registros de datos **PORTX**; a continuación se presenta las instrucciones que realizan lo anterior:

```

processor 16f877          ; Indica la versión de procesador
include <p16f877.inc>      ; Incluye la librería de la versión del procesador

org 0H                   ; Carga al vector de RESET la dirección de inicio
goto inicio

org 05H                   ; Dirección de inicio del programa del usuario
inicio: BSF STATUS,RP0    ; Cambia la banco 1
      BCF STATUS,RP1
      MOVLW B'00000000'   ; Configura al puerto B como salida (8 bits)
      MOVWF TRISB
      BCF STATUS,RP0      ; Regresa al banco cero
      .....
      .....
end                       ; Directiva de fin de programa

```

## Simulador PROTEUS

Es la herramienta propuesta para resolver y simular el funcionamiento de cada uno de los ejercicios asignados en las subsecuentes practicas, se pueden simular tanto aplicaciones analógicas como digitales; así mismo contiene librerías para trabajar diferentes microcontroladores y plataformas, como es el caso de los PIC's, Arduino, Raspberry Pi entre otras.

Permite la trasferencia y creación de esquemáticos, simular el funcionamiento y además contine soporte para crear tabletas PCB, para construir posteriormente circuitos impresos. La figura 5.4 muestra el IDE, con pantalla inicial.



Figura 3.4 IDE Proteus

De manera inicial se debe seleccionar **NEW PROJECT** para comenzar a desarrollar un proyecto,

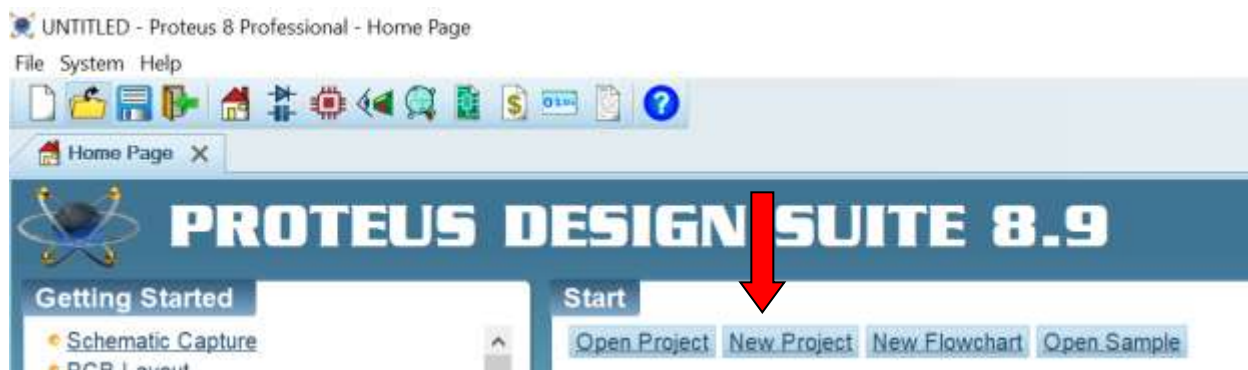
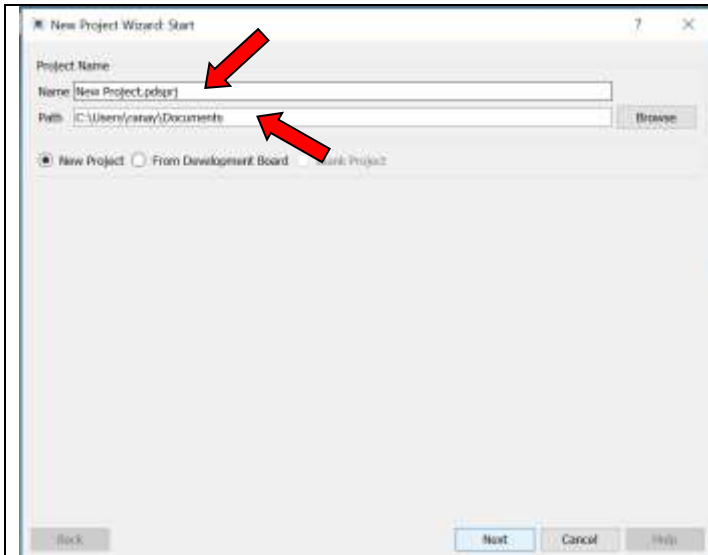


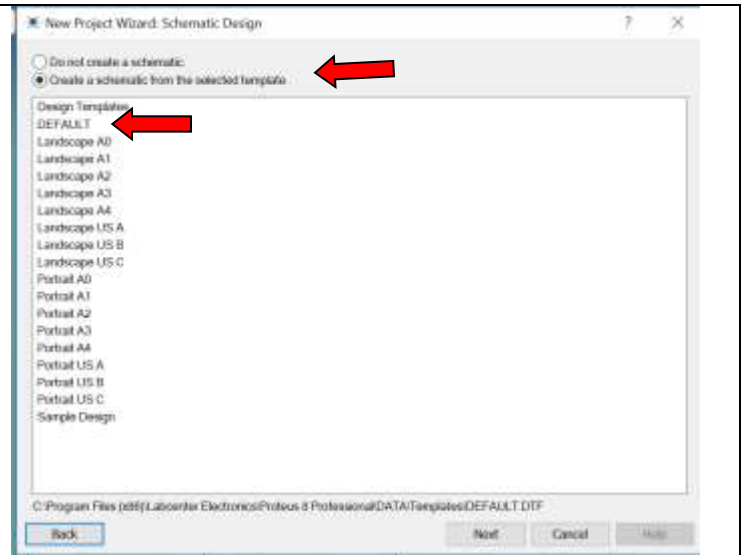
Figura 3.5 Comando para iniciar nuevo proyecto.

A continuación se realiza la configuración de proyecto, de acuerdo al siguiente orden, se muestran los gráficos en la figura 5.6:

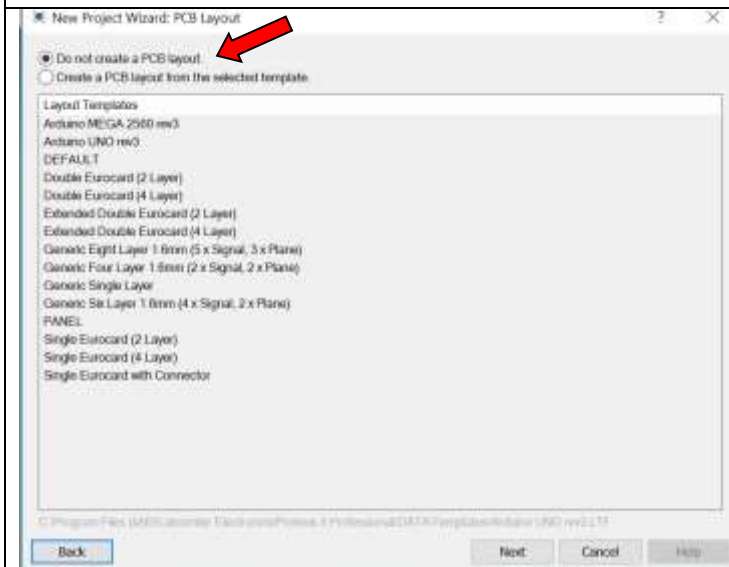
1. Nombre y ubicación del proyecto
2. Plantilla para creación del esquemático (se recomienda comenzar con la opción **DEFAULT**)
3. Creación del PBC LAYOUT (seleccionar **Do not create a PCB Layout**)
4. Agregar firmware, en caso de requerir (para efecto de la practica, seleccionar **No firmware Project**)
5. Mostrará el resumen del proyecto



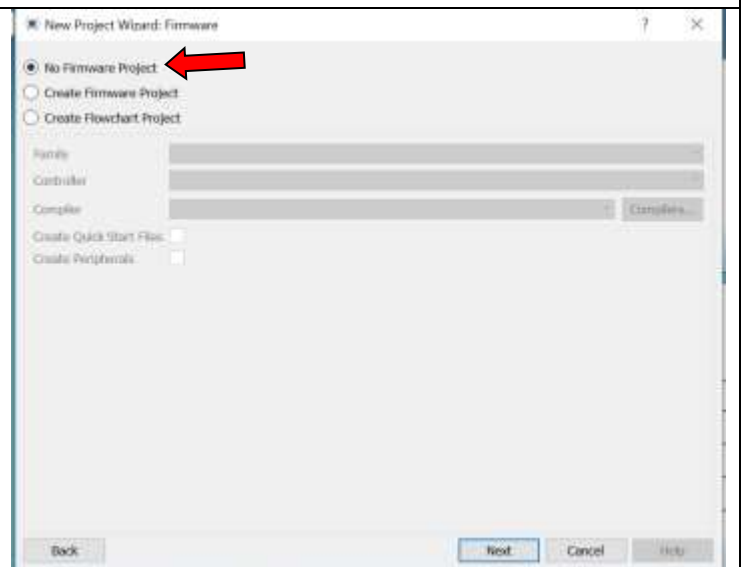
a. Nombre proyecto



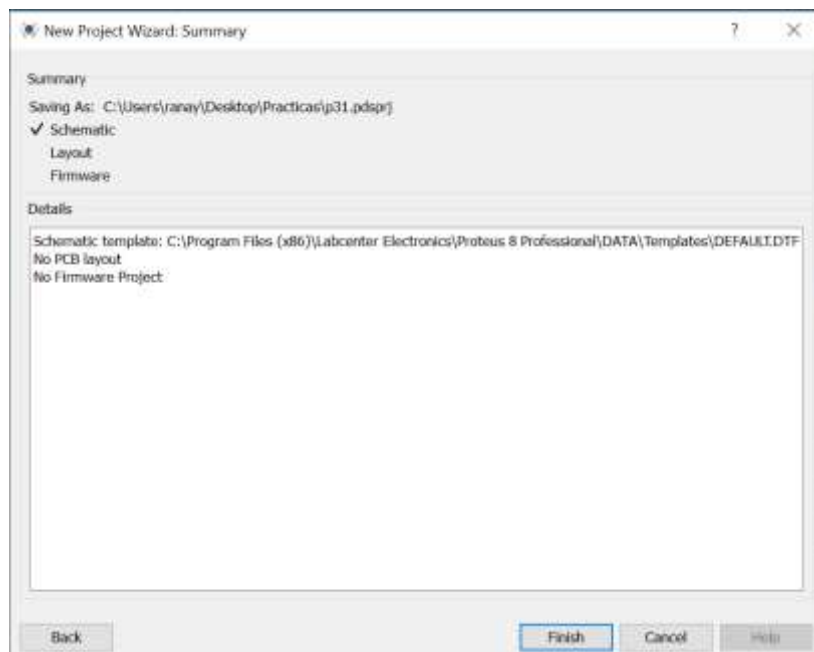
b. Configuración esquemático



c. PCB Layout



d. Selección del Firmware



e. Resumen del proyecto

Figura 3.6 Configuración del proyecto

Una vez hecho lo anterior, aparecerá el IDE de Proteus, con lo que se puede comenzar con la creación del esquemático.

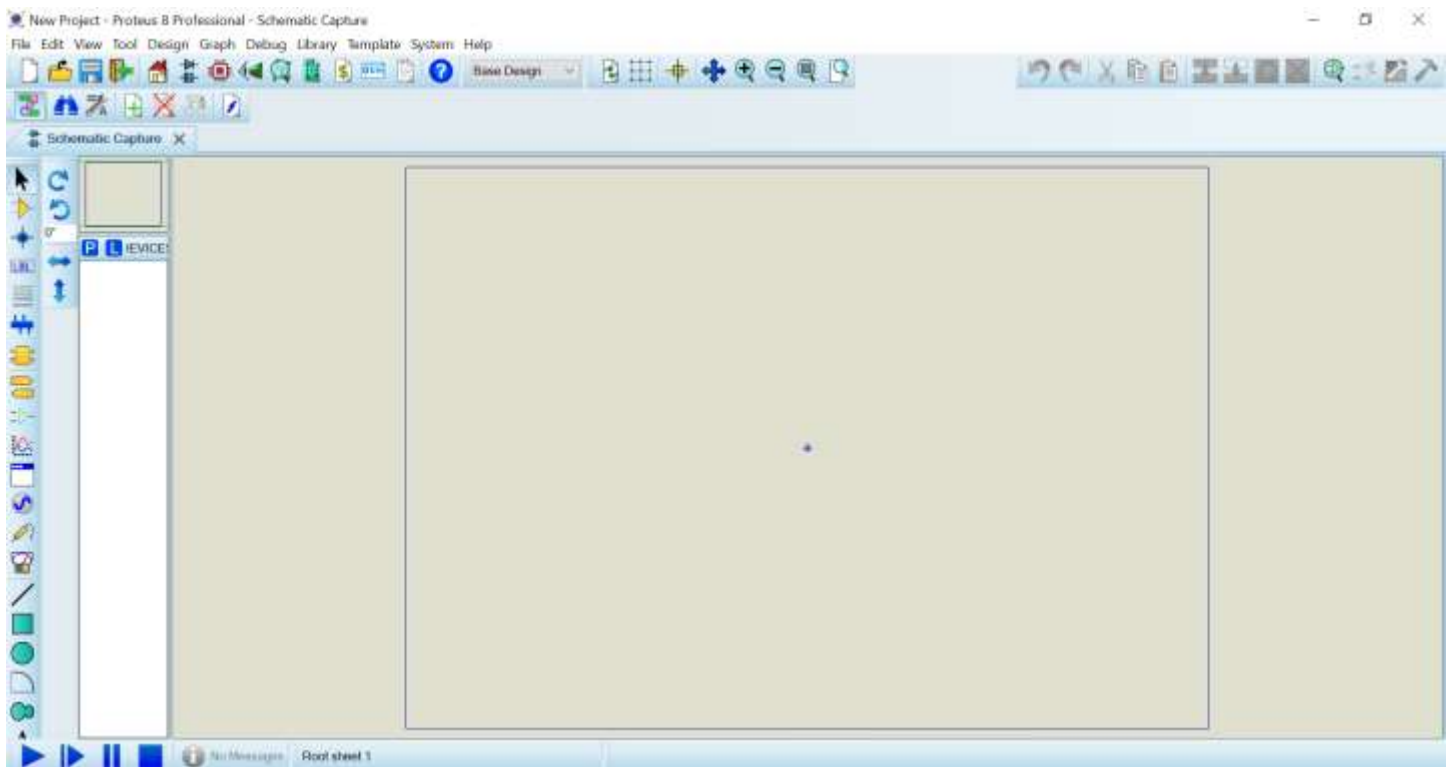


Figura 3.7 Pantalla inicial

En el IDE, se distinguen los comandos para trabajar, la zona de trabajo, los iconos para acceso rápido para seleccionar los componentes, instrumentos, así como la simulación.

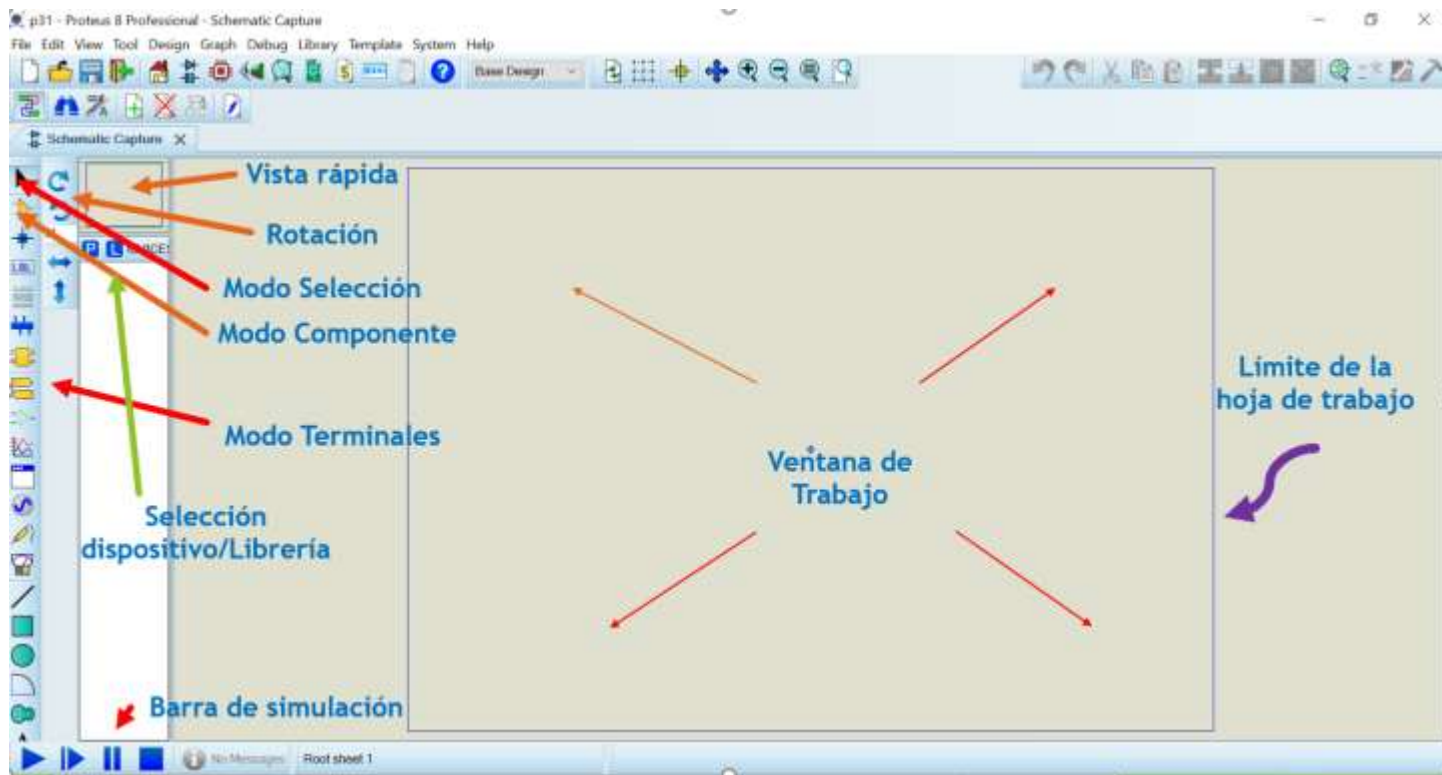


Figura 3.8 Descripción del IDE de Proteus

Para la creación del esquemático, se realizan las siguientes etapas:

1. Seleccionar **componentes** (usar el icono o el comando).
  - a. Presionar P para buscar el componente deseado
  - b. Cada que se seleccionan los componentes, aparecen enlistados
  - c. Una vez con los componentes requeridos, seleccionarlos de la lista, estos aparecerán en la vista rápida; con los comandos de rotación podrá cambiar su orientación
  - d. Colocar el mouse en la ventana de trabajo y dar click izquierdo para ubicarlo en la zona deseada
2. Con el modo **terminales**, podrá agregar Vcc, GND, etiquetas que sirven para unir conexiones sin requerir conexión.
3. Con todos los elementos y la ubicación deseada, se comienzan unir mediante líneas, para hacer esto, debe posicionarse en un extremo del componente (aparece un lápiz de color verde), dar click (izq), al momento de arrastrar el cursor se dibujará la línea de conexión para llevarlo al extremo deseado para repetir la acción inicial; podrá notar que se ha cubierto la unión.
4. Una vez terminado el alambrado, se procederá a cargar el programa en el PIC; seleccionará el programa previamente ensamblado o compilado, presionando doble click sobre el microcontrolador se abrirá la ventana para seleccionar y configurar.
5. Ya con el programa cargado, se procederá a realizar la simulación; usar los iconos para ejecutar, parar, pausar, etc.

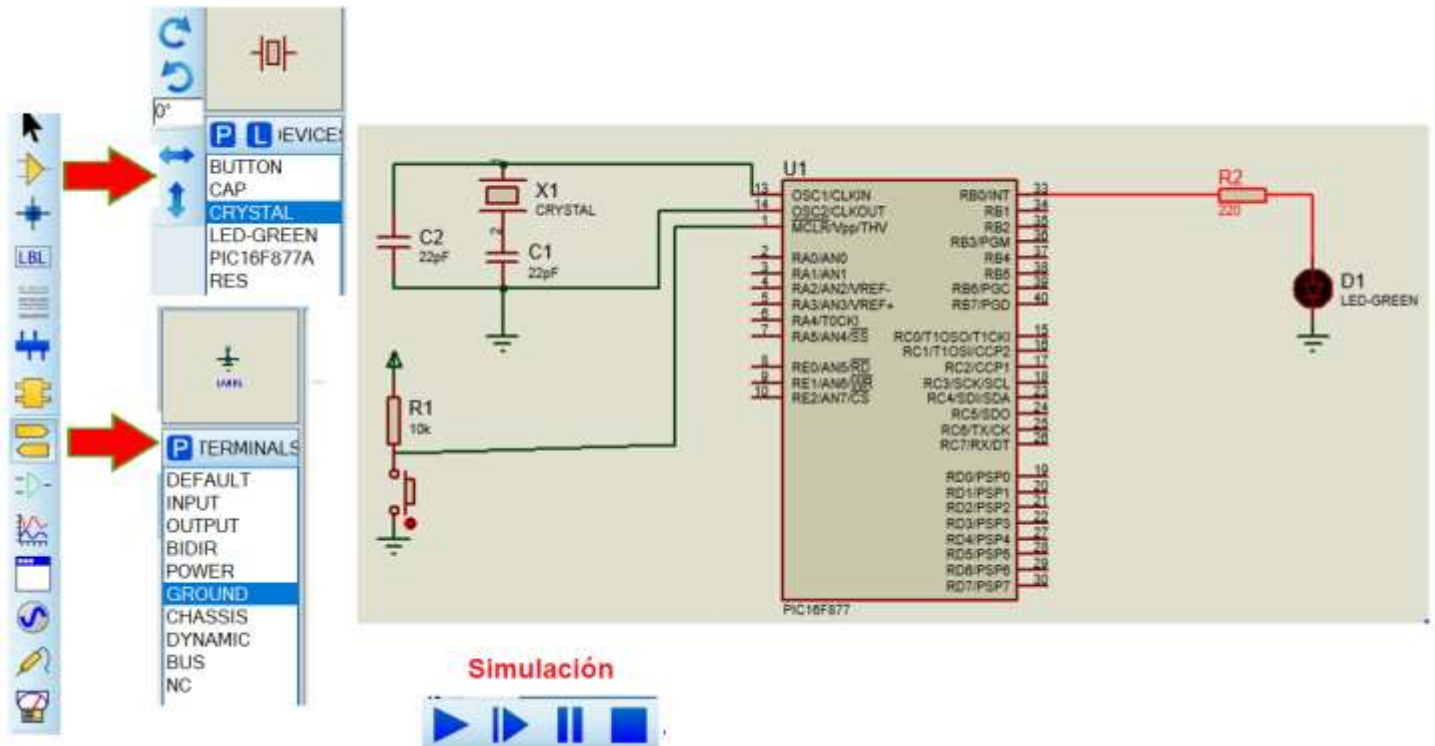


Figura 3.9 Selección de componentes, terminales, creación del esquemático y simulación.

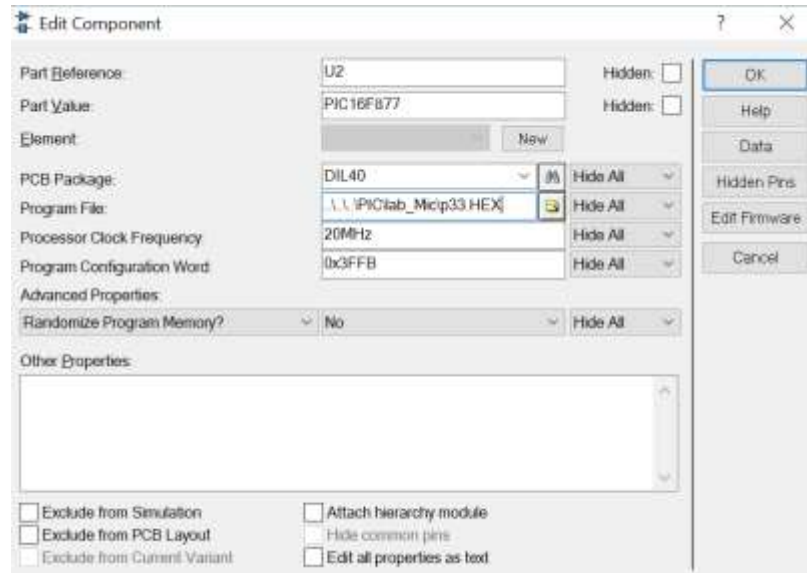


Figura 3.10 Selección del programa



**Desarrollo practica 3.** Para cada uno de los siguientes ejercicios, realizar los programas solicitados y comprobar el funcionamiento de ellos.

El sistema utilizado para esta práctica está diseñado de acuerdo al siguiente diagrama:

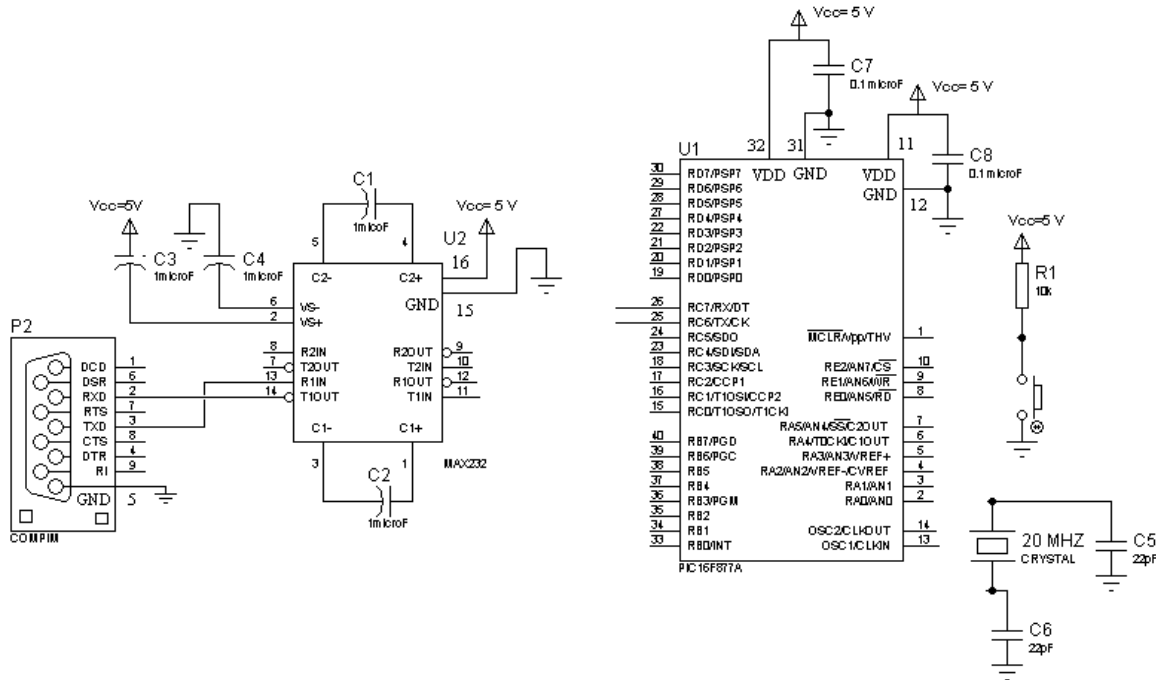


Figura 3.11 Circuito de prueba

- 1.- Revisar a detalle y en concordancia con el circuito 3.2, identificar las conexiones faltantes, discutir con sus compañeros y con su profesor(a) el impacto y función de los mismos.
- 2.- Completar las conexiones faltantes, utilizando jumpers; cerciorar el alambrado correcto.
- 3.- Utilizando PROTEUS, crear un proyecto con los elementos indicados, siguiendo el procedimiento previo.

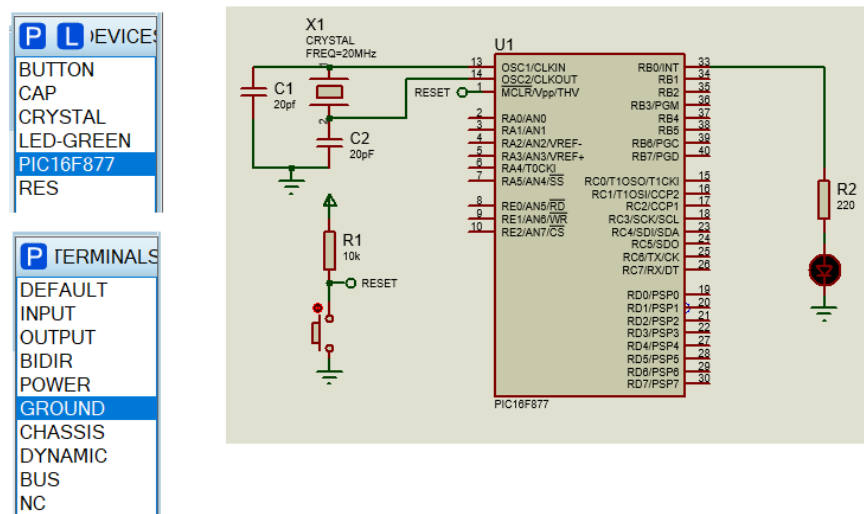


Figura 3.12 Circuito PB0

4.- Escribir, comentar e indicar que hace el siguiente programa.

```
processor 16f877
include <p16f877.inc>

valor1 equ h'21'
valor2 equ h'22'
valor3 equ h'23'
cte1 equ 20h
cte2 equ 50h
cte3 equ 60h

ORG 0
GOTO INICIO

ORG 5
INICIO:BSF STATUS,RPO
BCF STATUS,RP1
MOVLW H'0'
MOVWF TRISB
BCF STATUS,RPO
CLRF PORTB

loop2 BSF PORTB,0
CALL retardo
BCF PORTB,0
CALL retardo
GOTO loop2

retardo MOVLW cte1
MOVWF valor1
tres MOVLW cte2
MOVWF valor2
dos MOVLW cte3
MOVWF valor3
uno DECFSZ valor3
GOTO uno
DECFSZ valor2
GOTO dos
DECFSZ valor1
GOTO tres
RETURN
END
```

5.- Ensamblar y cargar el programa anterior, usando **PROTEUS**; simular el funcionamiento.

6.- En el programa, modifique el valor de **cte1** a **8h**, ensamblar, cargar y simular en **PROTEUS**; que alcanza a apreciar y porque?

7.- Modifique **cte1** a **80h**; ensamblar, cargar y simular en **PROTEUS**, existe algún cambio?

8.- Modificar el programa anterior, para que ahora se actualice el contenido de todos los bits del puerto B y se genere una rutina de retardo de un segundo.

Este programa requiere de 8 salidas conectadas al puerto B, tal como se muestra en la figura 3.13, agregar las resistencias faltantes y simule en **PROTEUS**.



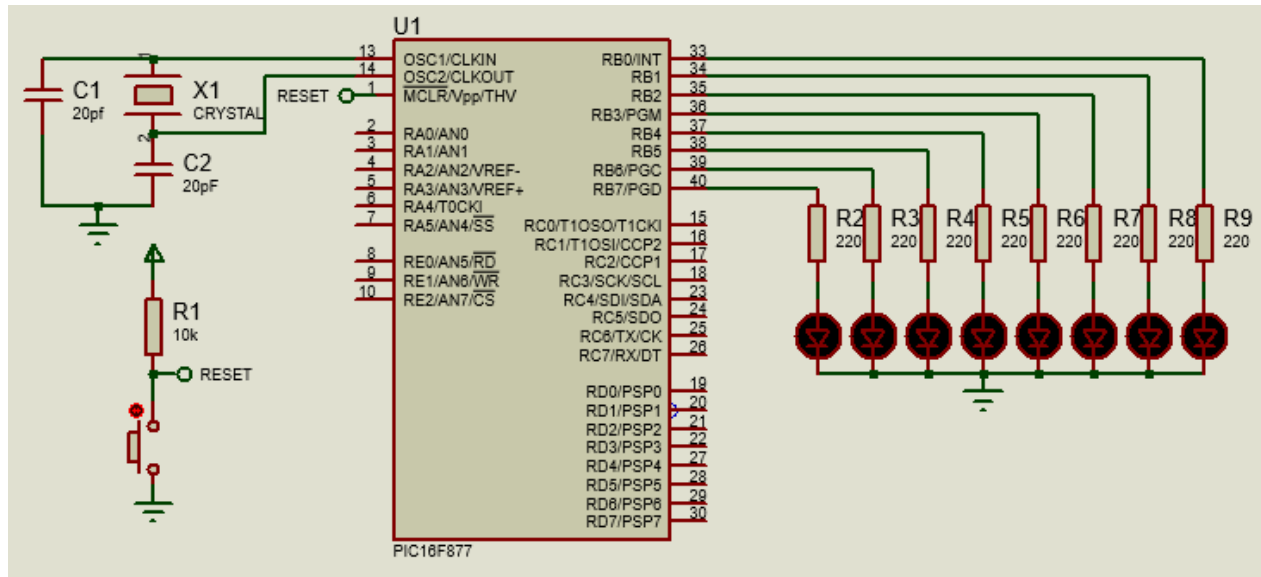


Figura 3.13 Conexión del sistema mínimo al módulo de 8 leds

9.- Realizar un programa que muestre la siguiente secuencia en el puerto B con retardos de  $\frac{1}{2}$  segundo, realizar la simulación en **PROTEUS**.

Secuencia:

H'80'  
H'40'  
H'20'  
H'10'  
H'08'  
H'04'  
H'02'  
H'01'

El circuito empleado es el mismo que en el ejercicio anterior.

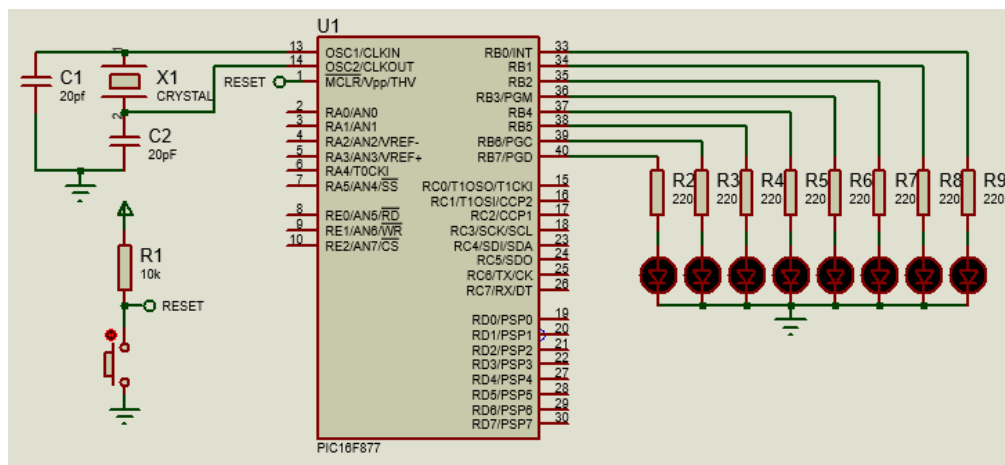
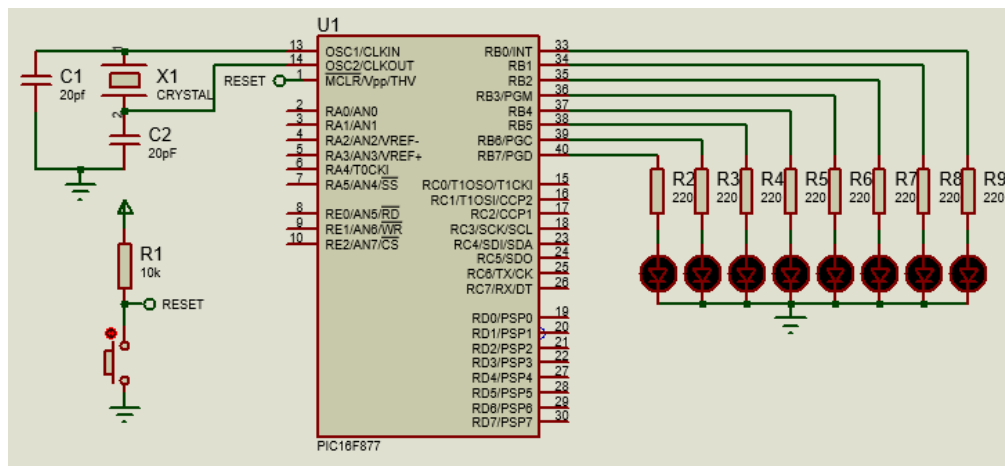


Figura 3.14 Circuito para visualizar el efecto del programa

10.- Realizar un programa que controle el funcionamiento de dos semáforos; cada estado tendrá una duración de 2 segundos, seleccionar los leds de los colores que correspondan y ubicarlos en la posición deseada.



Estado	Salida
1	VI, R2
2	A1, R2
3	R1, V2
4	R1, V2



3.15 Circuito para simular dos semáforos.

**Laboratorio de Microcomputadoras**  
**Práctica No 4**  
**Puertos Paralelos E/S**

---

**Objetivo.** Emplear los puertos paralelos que contiene un microcontrolador para realizar funciones de control, configurando estos como entrada y salida.

**Introducción**

Cuando el microcontrolador PIC será configurado como entrada, se recomienda limpiar el contenido del registro de datos del puerto mediante la instrucción **CLRF PORTX**, esto con la finalidad de iniciar los latches de datos del puerto, con esta instrucción se configurará al puerto de manera correcta.

Además de lo anterior, para el caso del puerto A y E se requiere indicar en el registro **ADCON1** ubicado en el banco 1 que se desea utilizar como E/S digitales, por lo que se escribirá un **06H** o **07H** en dicho registro, para posteriormente cargar el dato de configuración al registro **TRISA** o **TRISE**.

```
PROCESSOR 16F877
INCLUDE <P16F877.INC>
```

```
ORG 0 ;Vector de reset
GOTO INICIO
```

```
INICIO: ORG 5
        CLRF PORTA ; Limpia PORTA
        BSF STATUS,RP0 ; Cambia a banco 1
        BCF STATUS,RP1
        MOVLW 06H ; Define puertos A y E como digitales
        MOVWF ADCON1
        MOVLW H'3F' ; Configura puerto A como entrada
        MOVWF TRISA
        BCF STATUS,RP0 ; Cambia al banco 0
        ....
        ....
        END
```

**Desarrollo.** Para cada uno de los siguientes apartados, realizar los programas solicitados y comprobar el funcionamiento de ellos.

1.- Crear el proyecto en PROTEUS, agregar los componentes requerido, escribir el programa, ensamblar, cargar y simular el funcionamiento; el programa usará la terminal RA0 como fuente de entrada y todo el puerto B como salida.

Valor PA0	Acción puerto B
0	00000000
1	11111111

Tabla 4.1 Control de salidas controladas por un bit

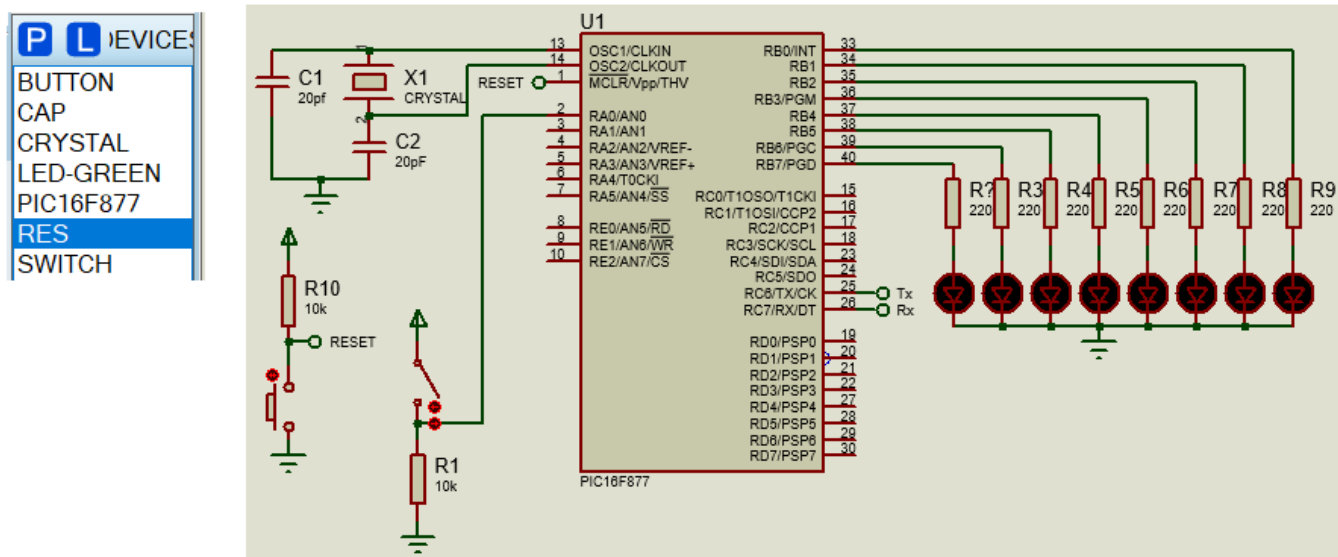


Figura 4.1 Esquemático actividad 1

2.- Escribir un programa, el cuál realice las siguientes acciones de control indicadas, para lo cuál requiere trabajar un puerto de entrada y otro puerto de salida, usar los sugeridos en el ejercicio anterior; generar retardos de ½ seg., agregar los componentes requeridos y simular el funcionamiento.

Usar la configuración de pull up o pull down, para los interruptores.

DATO PUERTO A	ACCION PUERTO B	Ejecución
0x00	Todos los leds apagados	00000000
0x01	Todos los leds encendidos	11111111
0x02	Corrimiento del bit más significativo hacia la derecha	10000000 01000000 00100000 ..... 00000001
0x03	Corrimiento del bit menos significativo hacia la izquierda	00000001 00000010 00000100 ..... 10000000
0x04	Corrimiento del bit más significativo hacia la derecha y a la izquierda	10000000 01000000 ..... 00000001 00000010 ..... 10000000
0x05	Apagar y encender todos los bits.	00000000 11111111

Tabla 4.2 Control de salidas completo

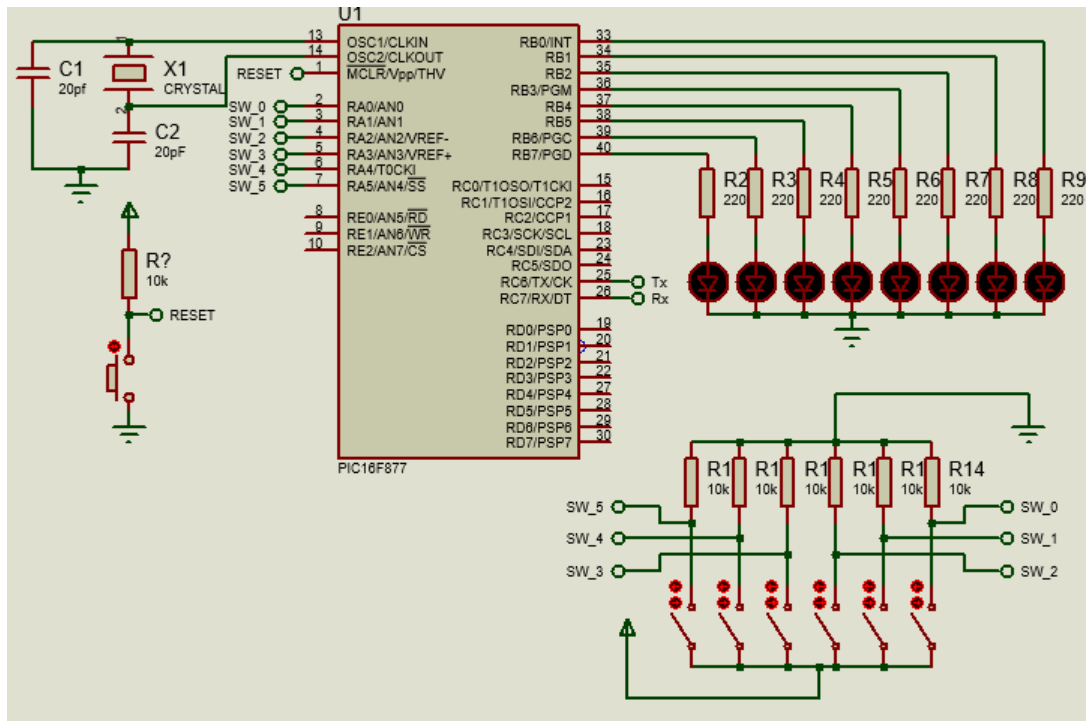


Figura 4.2 Esquemático actividad 2

## Laboratorio de Microcomputadoras

### Practica No. 5

### Control de actuadores

**Objetivo.** Emplear los puertos paralelos que contiene un microcontrolador, para controlar la operación de dos motores de corriente directa, motores a pasos y servomotores.

### Introducción

Entre los actuadores más empleados se encuentran:

- a. Motores de corriente directa
- b. Motores a pasos
- c. Servomotores

En cualquiera de los anteriores se genera un campo magnético producido por la circulación de corriente por sus devanados creando fuerzas de atracción y repulsión.

Un microcontrolador no otorga la corriente requerida para producir el movimiento de rotación en los motores, por lo que se hace indispensable el uso de un amplificador de corriente, que puede ser desde un solo transistor o un arreglo de cuatro transistores o contar con un driver de potencia disponible como el L293, L298, TB6612, entre otros, la mayoría de ellos funcionando de manera parecida.

### Motores de Corriente Directa

Para la práctica se emplea el driver L293, que tiene el siguiente encapsulado:

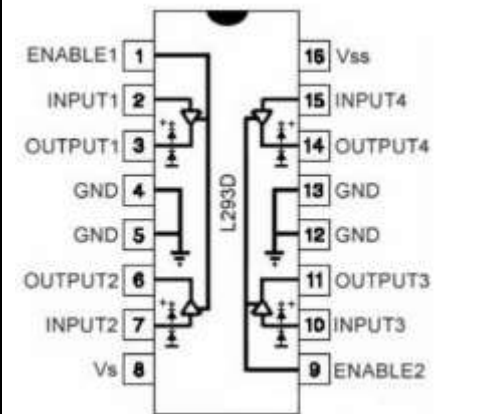
	Terminal	Función
	ENABLE1, ENABLE2	Habilitadores (Izq, Der)
	INPUT1, INPUT2, INPUT3, INPUT4	Señales de control
	OUTPUT1, OUTPUT2, OUTPUT3, OUTPUT4	Salidas, conexión a motores
	GND	0 Volts
	VSS	5 Volts
	VS	Tensión del motor, puede ser desde 0.2 V a 32 V

Figura 5.1 Driver L293

El dispositivo que nos permite entregar la potencia y señales de control a motores de corriente directa es el L293 B/D o en caso de requerir mayor corriente usar el L298; se sugiere para una mayor información revisar la hoja de datos de este circuito.

El L293 tiene dos terminales para alimentación, una de ellas es para el propio dispositivo, el cuál debe ser de 5 volts, y otra para la tensión en los motores la cual puede ser desde 0.2 volts hasta 32 volts (de acuerdo al voltaje de operación del motor), así mismo permite tener el control de la velocidad de rotación de o los motor(es), mediante las terminales EN1 y EN2; por último la dirección de rotación se establece de acuerdo al nivel lógico existente entre las terminales identificadas como DIR1 y DIR2 para un motor, DIR3 y DIR4 para el otro motor.

Por ejemplo si EN=1, Dir1=1 y Dir2=0, el motor girará hacia un sentido y cuando DIR1=0 y DIR2=1, el motor girará en sentido contrario. El motor se mantiene parado cuando EN1=0 o el valor en Dir1= Dir2.

A manera de protección del microcontrolador, se recomienda contar con dos fuentes de alimentación independientes (una para el PIC y otra para el circuito L293B) y contar con una etapa de acoplamiento óptico para tener un mejor desempeño del sistema.

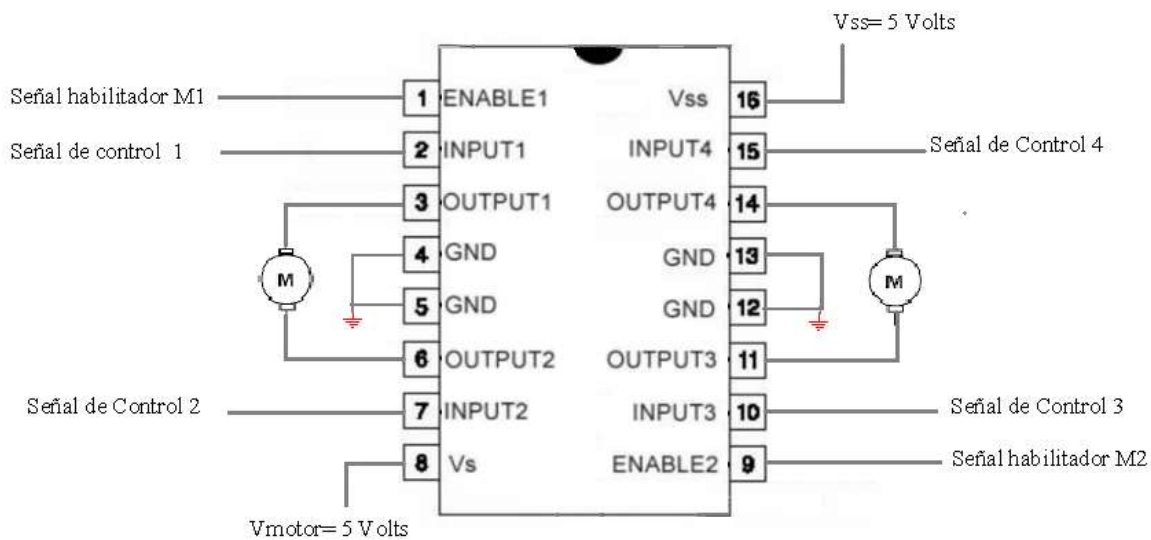


Figura 5.2 Conexión Driver L293 para control de motores de CD.

## Motores a pasos

Existen dos tipos de motores a pasos, los unipolares y los bipolares.

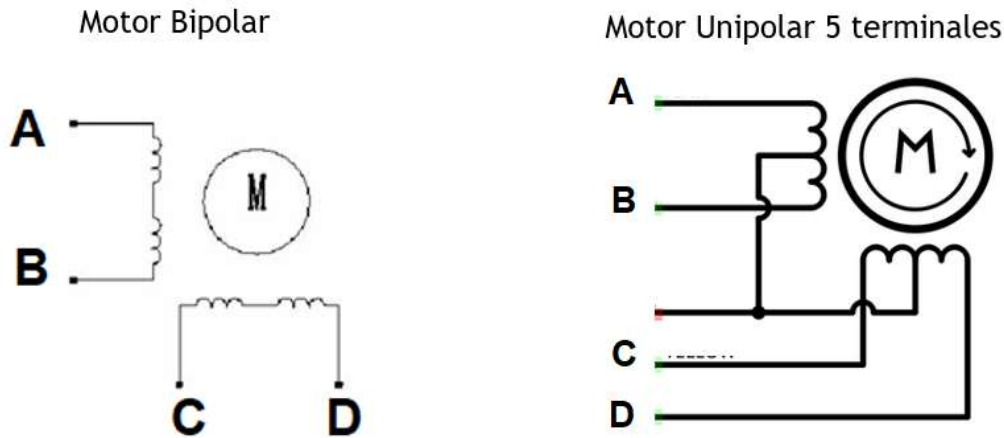


Figura 5.3 Motores a pasos

Para los motores bipolares se emplea regularmente dos puentes H para controlarlos, en este caso el L293B, mientras que, los motores unipolares las bobinas son controladas de manera independiente, con lo que se usan transistores; es recomendable el empleo del driver ULN2003A.

### ULN2003A

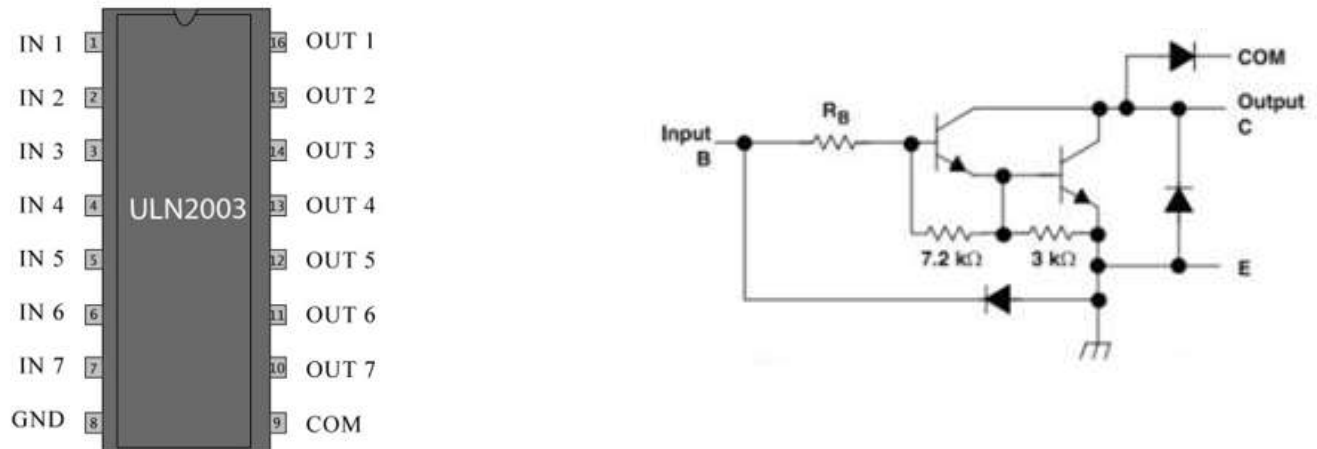


Figura 5.4 Driver UNL2003

El control de paso unipolares se controla mediante tres técnicas: Pasos completos, medio paso y oleada, lo que implica programar la secuencia adecuada.



La técnica recomendada para la practica es la de pasos completos; que se describe en la figura 5.5

PASO	Bobina A	Bobina B	Bobina C	Bobina D	EFEECTO
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

Figura 5.5 Secuencia de pasos completos para motores unipolares

## Servo motores

El servo motor contiene en su encapsulado los mecanismos, que le permiten funcionar sin requerir elementos externos; integra el sistema de control para colocar en la posición deseada, el driver de potencia que amplifica la corriente del microcontrolador y el sistema de reducción en base de engranes para incrementar el torque.

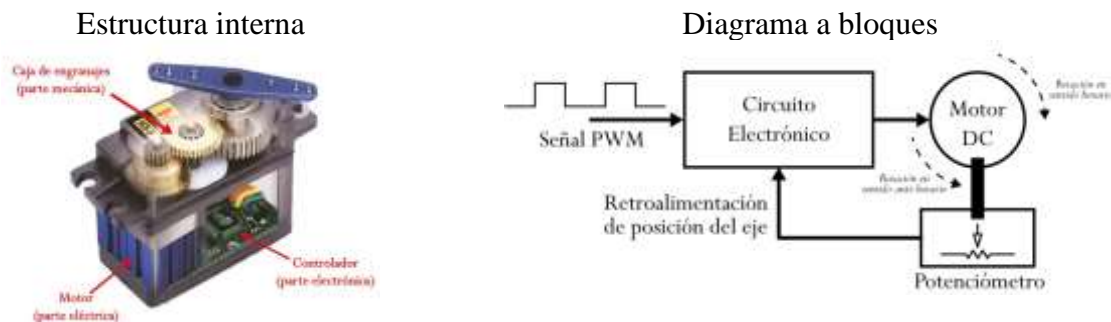


Figura 5.6 Descripción de servomotor

Como se puede ver en la figura 5.5, tiene un cable con tres terminales:

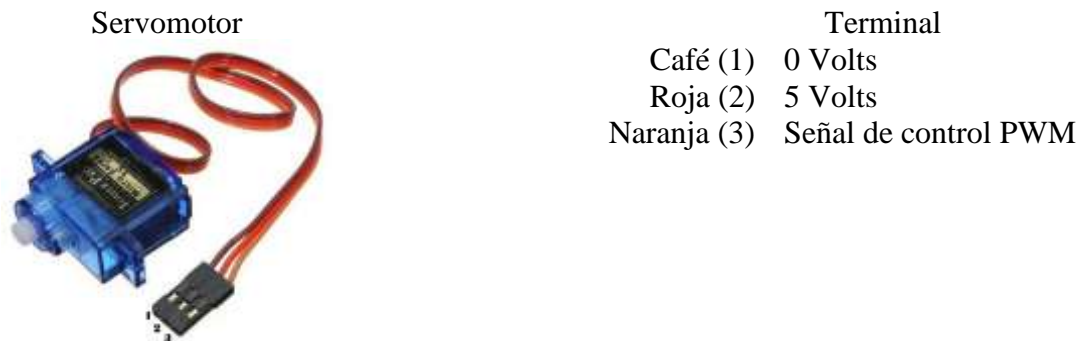


Figura 5.7 Servomotor y sus terminales

Para controlar la posición del cursor se ingresa un pulso en la terminal de entrada al servomotor, debe generar la señal PWM con periodo de 20 ms; esta señal debe modular el pulso en alto para que se encuentre en un tiempo comprendido entre 0.5 ms a 2.5 ms. La posición de 0° a 180°, como se muestra a continuación.

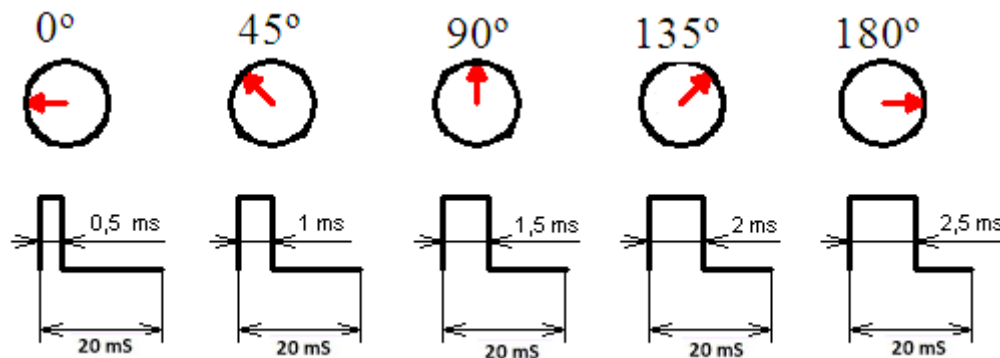


Figura 5.8 Pulsos para control de servomotor

Material a utilizar para la práctica:

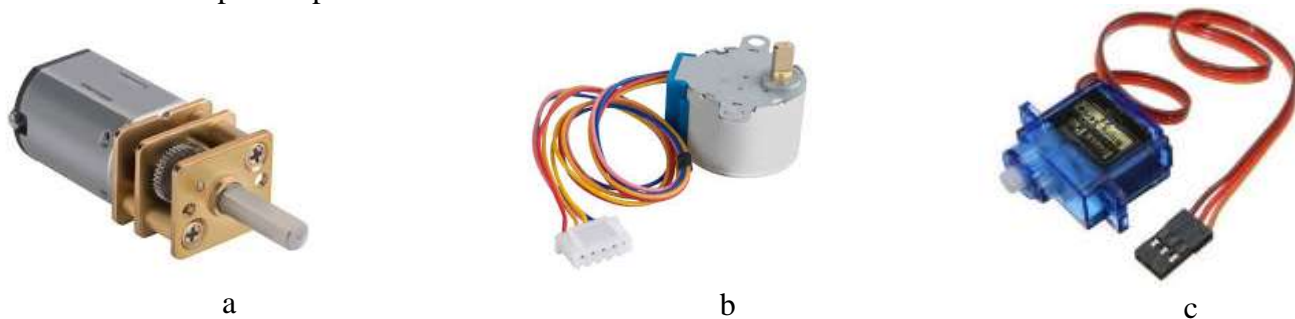


Figura 5.9 Moto-reductor, motor a pasos y servomotor

**Desarrollo.** Utilizando el circuito de potencia de motores de corriente directa y el sistema de desarrollo del microcontrolador PIC, realizar los programas solicitados.

1.- Considerando la asignación de terminales asignadas en la figura 5.1; realizar el programa que ejecute el control indicado en la tabla 5.1.

Nota: Las tierras de los ambos circuitos están conectados entre sí.

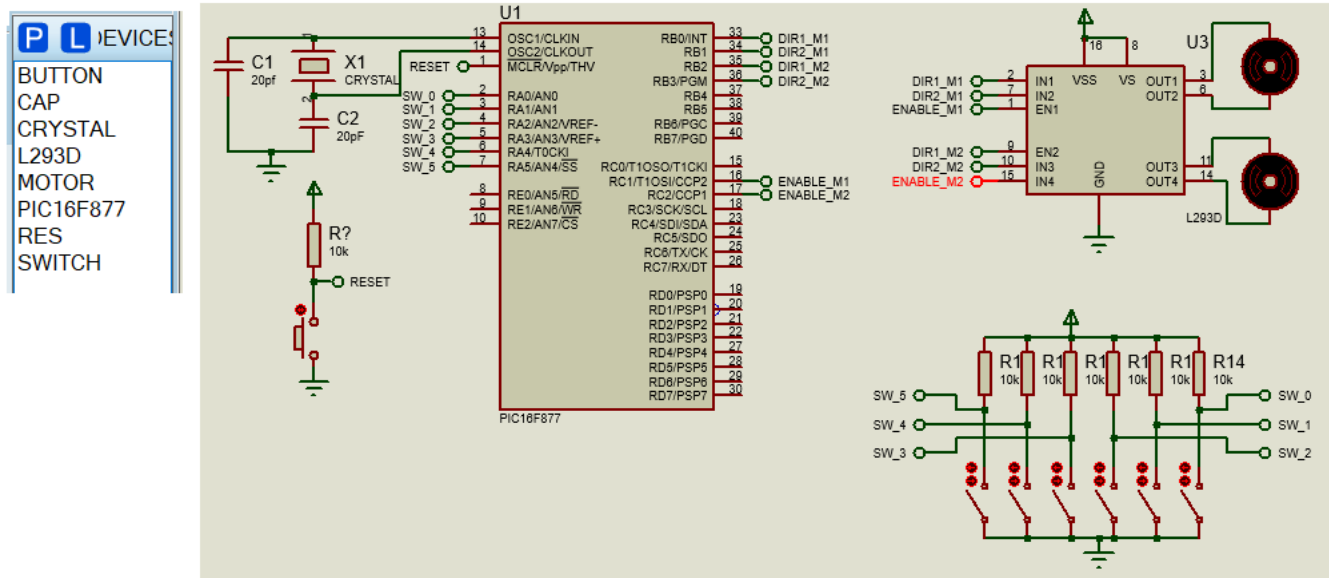


Figura 5.10 Circuito control de motores de CD

La asignación de las terminales queda de la siguiente manera:

MOTOR2		
RC2	RB3	RB2
ENABLE_M2	DIR1_M2	DIR2_M2

MOTOR1		
RC1	RB1	RB0
ENABLE_M1	DIR1_M1	DIR2_M1

DATO Puerto Paralelo	ACCION	
	MOTOR M1	MOTOR M2
0x00	PARO	PARO
0x01	PARO	HORARIO
0x02	PARO	ANTI-HORARIO
0x03	HORARIO	PARO
0x04	ANTI-HORARIO	PARO
0x05	HORARIO	HORARIO
0x06	ANTI-HORARIO	ANTI-HORARIO
0x07	HORARIO	ANTI-HORARIO
0x08	ANTI-HORARIO	HORARIO

Tabla 5.1 Operación de motores de corriente directa

2.- Realizar un programa que controle la cantidad de pasos que debe dar un motor, así como el sentido de giro; agregar los componentes requeridos en PROTEUS, simular el funcionamiento.

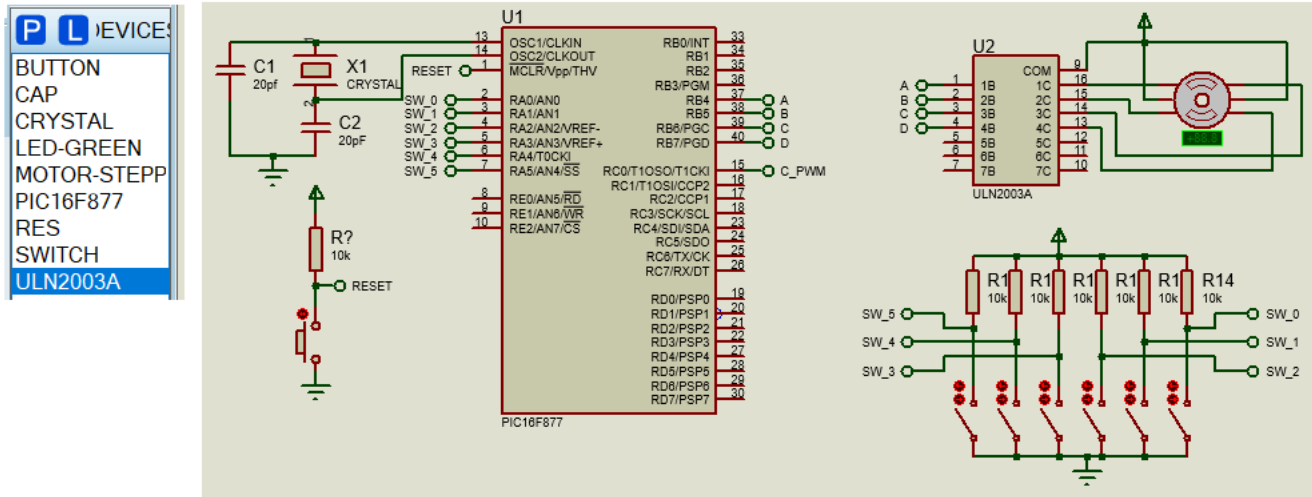


Figura 5.11 Conexión de motor a pasos

Dato Puerto Paralelo	Motor a pasos
0x00	Motor en paro
0x01	Gira en sentido horario
0x02	Gira en sentido anti horario
0x03	Gira cinco vueltas en sentido horario
0x04	Gira 10 vueltas en sentido anti horario

Tabla 5.2 Control del motor a pasos

3.- Utilizando un servo motor realizar el control mostrado en la tabla No. 5.3, agregar los componentes requeridos del proyecto en PROTEUS, simular el funcionamiento.

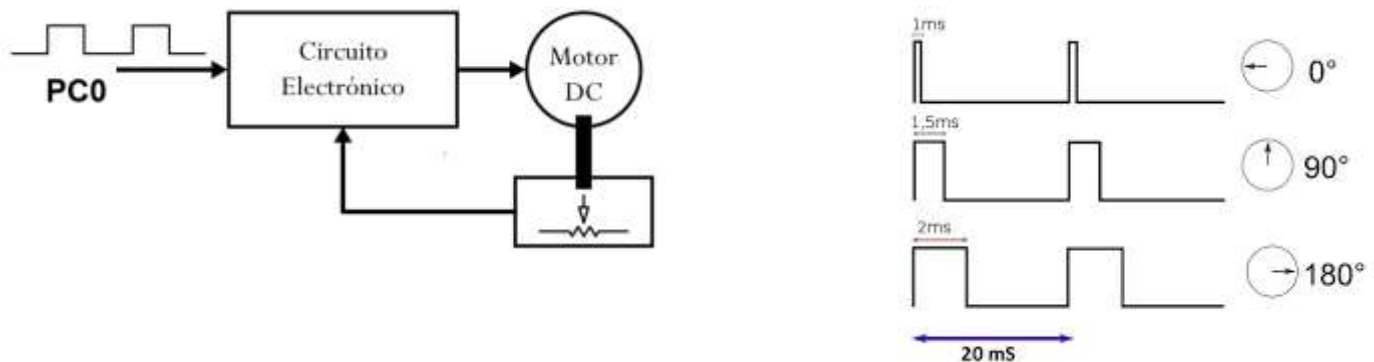


Figura 5.12 Funcionamiento y conexión del servo




SW2	SW1	SW0	Posición Servo	Representación
1	0	0	Izquierda	 0°
0	1	0	Central	 90°
0	0	1	Derecha	 180°

Tabla 5.3 Funcionamiento del servo motor

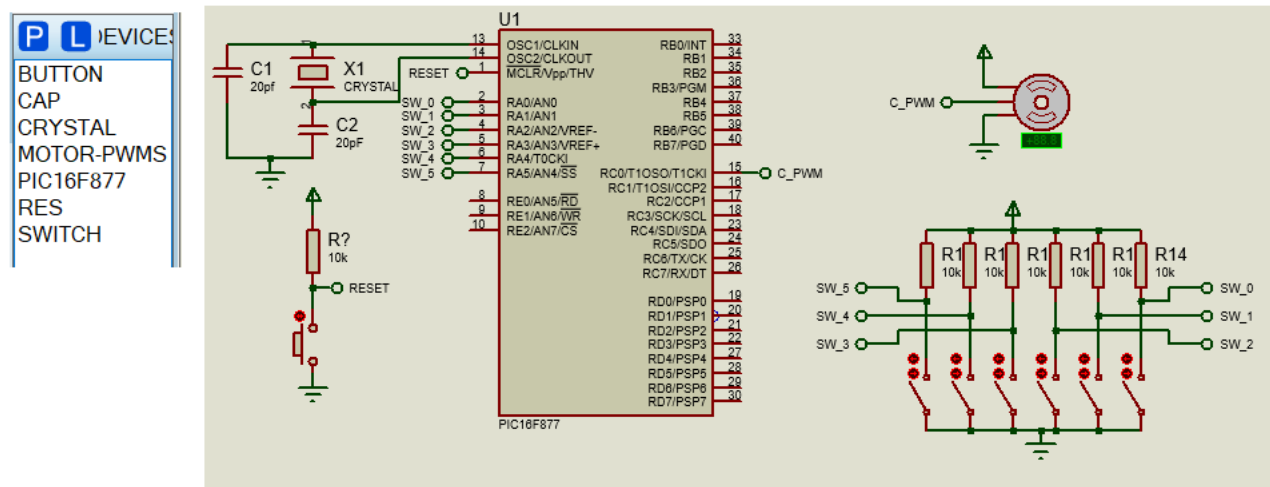


Figura 5.13 Circuito para funcionamiento del servo

**Laboratorio de Microcomputadoras**  
**Práctica No. 6**  
**Convertidor Analógico/Digital**

**Objetivo.** Familiarizar al alumno con el uso y aplicación del Convertidor Analógico/Digital de un microcontrolador.

**Introducción**

El microcontrolador PIC16F877 tiene 8 posibles canales de entrada por los cuales se pueden procesar señales analógicas de 10 bits de resolución.

Los registros involucrados para este periférico son los mostrados a continuación, la dirección y banco donde están ubicados se pueden consultar en la información dada en la práctica uno.

Registro de configuración ADCON0 - **0x1F**

<b>ADCON0</b>	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
---------------	-------	-------	------	------	------	---------	---	------

ADCS1:ADCS0      Seleccionan la frecuencia de reloj  
 CHS2-0          Selección del canal de entrada  
 GO/DONE        Si GO/DONE=1; inicia el proceso de conversión  
                     Si GO/DONE=0; terminó la conversión  
 ADON            Enciende al convertidor A/D

Registro de configuración ADCON1 - **0x9F**

<b>ADCON1</b>	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
---------------	------	---	---	---	-------	-------	-------	-------

ADFM            El resultado de la conversión se almacena en los registros ADRESH:ADRESL, formando un dato de 10 bits, pudiendo ser reflejado de la siguiente manera:

Si ADFM=1; el resultado es justificado en el registro ADRESH (los seis bits más significativos de este registro valen cero).

Si ADFM=0; el resultado es justificado en el registro ADRESL (Los seis bits menos significativos de este registro valen cero)

PCFG3-0        Configura a los puertos paralelos A y E como entradas al convertidor A/D; en el caso de utilizar ambos puertos como entradas analógicas, se configuran estas banderas en cero.

Registro de resultados parte alta ADRESH - **0x1E**

**ADRESH**

--	--	--	--	--	--	--	--

Parte alta del resultado de la conversión

Registro de resultados parte baja ADRESL - **0x9E**

**ADRESL**

--	--	--	--	--	--	--	--

Parte baja del resultado de la conversión

El algoritmo a emplear para el uso del convertidor A/D, con resolución de 8 bits:

1. Ubicado en el banco cero, limpiar el puerto A, usando CLRF PORTA.
2. Cambiar al banco uno.
3. Configurar el puerto A como entradas analógicas, escribir 00H al registro ADCON1.
4. Regresar al banco 0.
5. Realizar la configuración de la fuente de reloj, el canal de entrada y prender al convertidor A/D, en el registro ADCON0.
6. Iniciar la conversión colocando un '1' a la bandera GO/DONE#.
7. Generar un tiempo de retardo de 20 microsegundos.
8. Esperar a que GO/DONE# sea igual a cero, lo que indica que ha concluido el proceso de conversión.
9. Lee el resultado de la conversión del registro ADRESH.

**Desarrollo.** Realizar los programas solicitados y comprobar su funcionamiento.

1.- Empleando el canal de su elección del convertido A/D, realizar un programa en el cuál, de acuerdo a una entrada analógica que se ingrese por este canal, se represente el resultado de la conversión en un puerto paralelo utilizar el arreglo de leds para ver la salida, como se muestra en la figura 6.1.

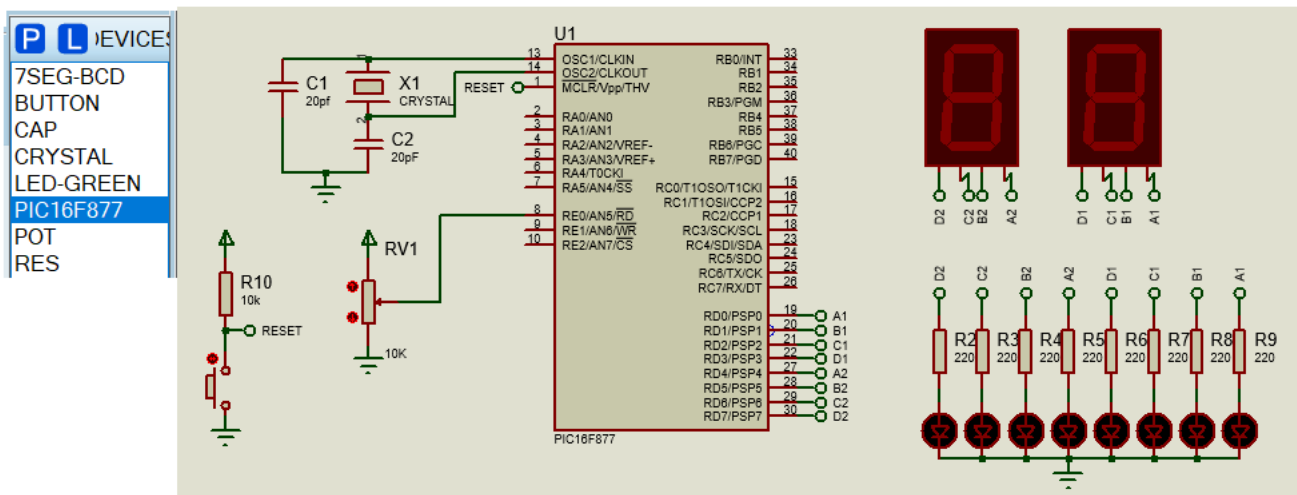


Figura 6.1 Circuito con lectura de una señal analógica



Tabla 6.1  
Donde  $V_{cc} = 5$  volts

Señal	PD2	PD1	PD0
Ve1>Ve2 y Ve3	0	0	1
Ve2>Ve1 y Ve3	0	1	1
Ve3>Ve1 y Ve2	1	1	1

Laboratorio Microcomputadoras 48



Circuito empleado para este ejercicio.

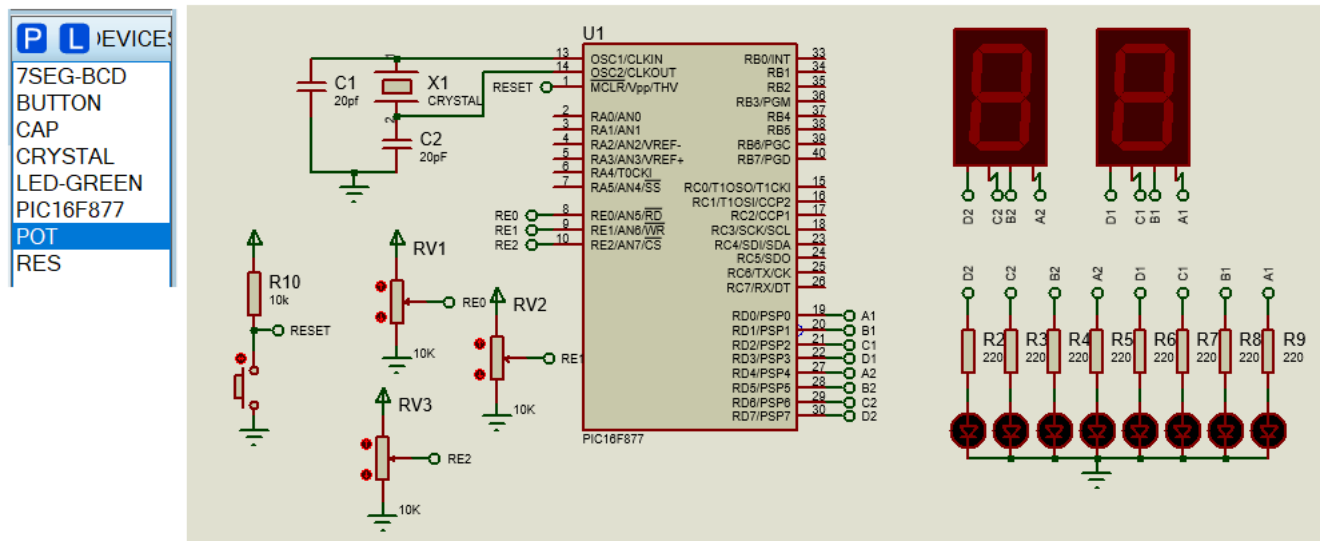


Figura 6.3 Tres señales analógicas

## Laboratorio de Microcomputadoras

## Práctica No. 7

## Puerto Serie SCI (Asíncrono)

**Objetivo.** Familiarizar al alumno en el uso de una Interfaz de Comunicación Serie Asíncrona de un microcontrolador.

**Introducción**

El microcontrolador PIC16F877 contiene un módulo USART, el cuál permite la comunicación de tipo Asíncrona, con el uso de los pines RC6 y RC7 del puerto C, la tasa de transferencia (BAUD RATE) se configura por software, dentro de una gama amplia de valores, además de contar con banderas que indican la terminación, ya sea de la transmisión o la recepción de datos.

Registros ocupados en la comunicación serie:

**Registro generador de Baud Rate - 0x99**

<b>SPBRG</b>	7						0
--------------	---	--	--	--	--	--	---

Con este registro se configura la velocidad de comunicación utilizando una expresión matemática para encontrar un valor de 8 bits que será cargado en el, la tasa de transferencia y fórmula dependerá de valor que sea cargado la bandera **BRGH** del registro **TXSTA**.

Si XTAL=20MHZ		
Baud Rate [BAUDS]	BRGH=0 SPBRG	BRGH=1 SPBRG
1200	255	-
2400	129	-
4800	64	
9600	32	129
19200	15	64
38400	8	32

**Registro usado en el módulo transmisor - 0x98**

<b>TXSTA</b>	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D
--------------	------	-----	------	------	---	------	------	------

Donde:

CSRC	Bit de selección del reloj, aplicable solo en modo de comunicación síncrona
TX9	Habilita el 9° bit de transmisión
TXEN	Activa la transmisión
SYNC	Selección del modo de comunicación a emplear SYNC=0 Comunicación asíncrona SYNC=1 Comunicación síncrona

BRGH	Bit de selección de baudios BRGH=0 Baja velocidad BRGH=1 Alta velocidad
TRMT	Estado del registro de corrimiento de transmisión, indica que se ha transmitido el dato si esta bandera es igual a uno.
TX9D	9° bit de datos a transmitir

### Registro del módulo receptor - 0x18

RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
-------	------	-----	------	------	-------	------	------	------

Donde:

SPEN	Habilita el puerto serie SPEN=0 Deshabilitado SPEN=1 Habilitado
RX9	Habilita el 9° bit de recepción
SREN	Configura la recepción sencilla, aplicable solo para comunicación síncrona
CREN	Configura la recepción continua en modo de comunicación asíncrona
ADDEN, FERR, OERR	Indicadores de posibles errores en la recepción de datos
RX9D	9° bit de dato

### Registro de banderas de recepción y transmisión completa 0x0C

PIR1			RCIF	TXIF				
------	--	--	------	------	--	--	--	--

RCIF	Bandera de recepción completa RCIF=0 Recepción en proceso RCIF=1 Recepción completa; indica que es posible leer el registro de recepción RCREG
TXIF	Bandera de transmisión completa TXIF=0 Recepción en proceso TXIF=1 Recepción completa; indica que es posible escribir otro dato al registro TXREG

Registro de transmisión 0x19

TXREG								
-------	--	--	--	--	--	--	--	--

Registro de recepción 0x1A

RCREG								
-------	--	--	--	--	--	--	--	--

Algoritmo de empleo del módulo USAR en la modalidad Asíncrona utilizando transmisor y receptor en el mismo programa.

1. Cambiar al banco uno
2. Configura la bandera BRGH para seleccionar alta o baja velocidad
3. Cargar el valor correspondiente a la velocidad requerida (consultar los valores del data sheet)
4. Configurar el modo asíncrono SYNC=0 del registro TXSTA
5. Habilita la transmisión TXEN=1 del registro TXSTA
6. Regresar al banco cero
7. Habilita la recepción de datos CREN=1 del registro RCSTA
8. Habilita el puerto serie SPEN del registro RCSTA
9. Realizar la operación deseada por el programa
  - a. Transmisión: Escribir el dato al registro TXREG y esperar a la transmisión del mismo, esperar a que TRMT=1 en el registro TXSTA (considerar que este registro está ubicado en el banco uno)
  - b. Recepción: Esperar hasta que la bandera RCIF del registro PIR=1, indicador de recepción completa (tomar en cuenta que este registro esta ubicado en el banco cero)

**Desarrollo.** Realizar los siguientes apartados:

1.- Escribir, comentar y ensamblar el siguiente código.

	processor 16f877	RECIBE:	BTFSS PIR1,RCIF
	include<p16f877.inc>		GOTO RECIBE
	ORG 0		MOVF RCREG,W
	GOTO inicio		MOVWF TXREG
INICIO:	ORG 5	TRASMITE:	BSF STATUS,RP0
	BSF STATUS,RP0		BTFSS TXSTA,TRMT
	BCF STATUS,RP1		GOTO TRASMITE
	BSF TXSTA,BRGH		BCF STATUS,RP0
	MOVLW D'129'		GOTO RECIBE
	MOVWF SPBRG		
	BCF TXSTA,SYNC		END
	BSF TXSTA,TXEN		
	BCF STATUS,RP0		
	BSF RCSTA,SPEN		
	BSF RCSTA,CREN		

2.- Crear el proyecto en PROTEUS, incluyendo los componentes propuestos, así como la terminal virtual para comprobar el funcionamiento del programa.

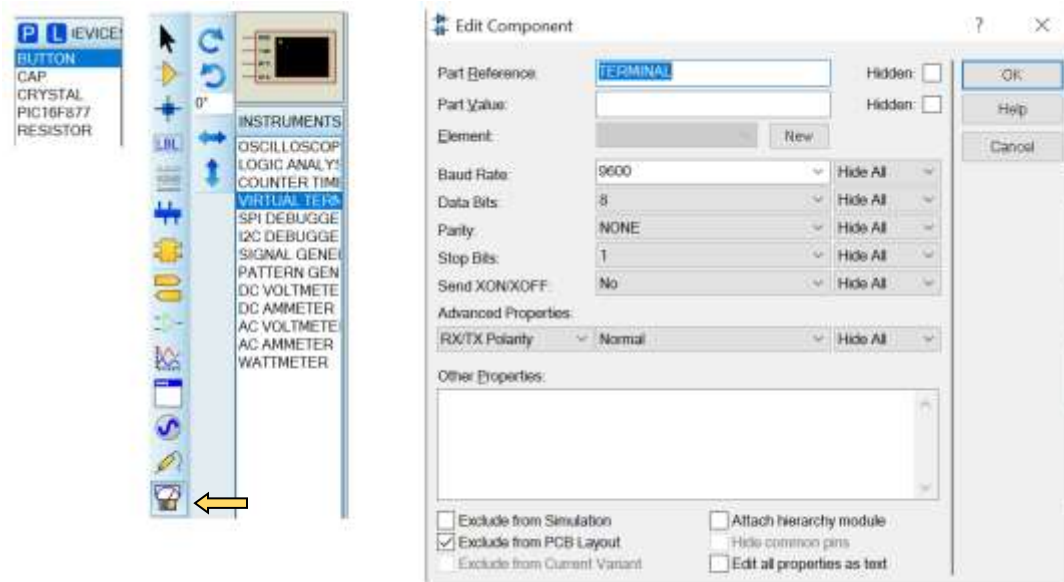


Figura 7.1 Configuración de la terminal virtual en proteus

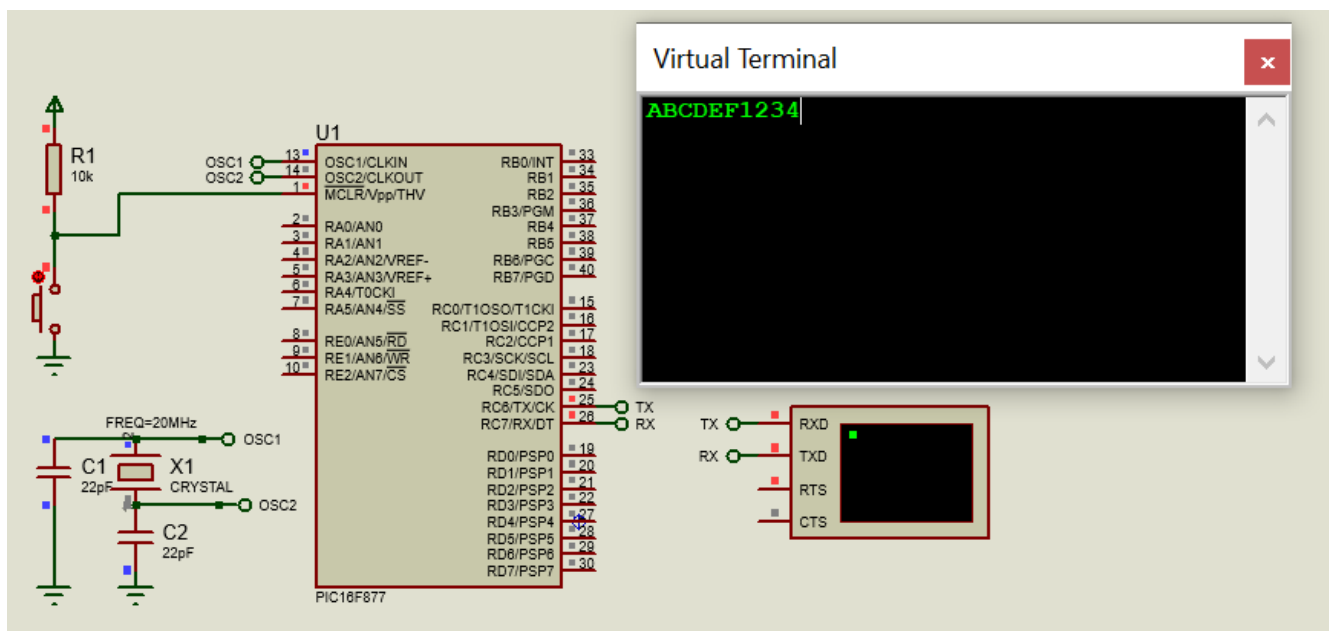


Figura 7.2 Recepción y transmisión de datos por puerto serie asíncrono

3.- Realizar un programa que despliegue la siguiente cadena en una terminal.

**HOLA UNAM**

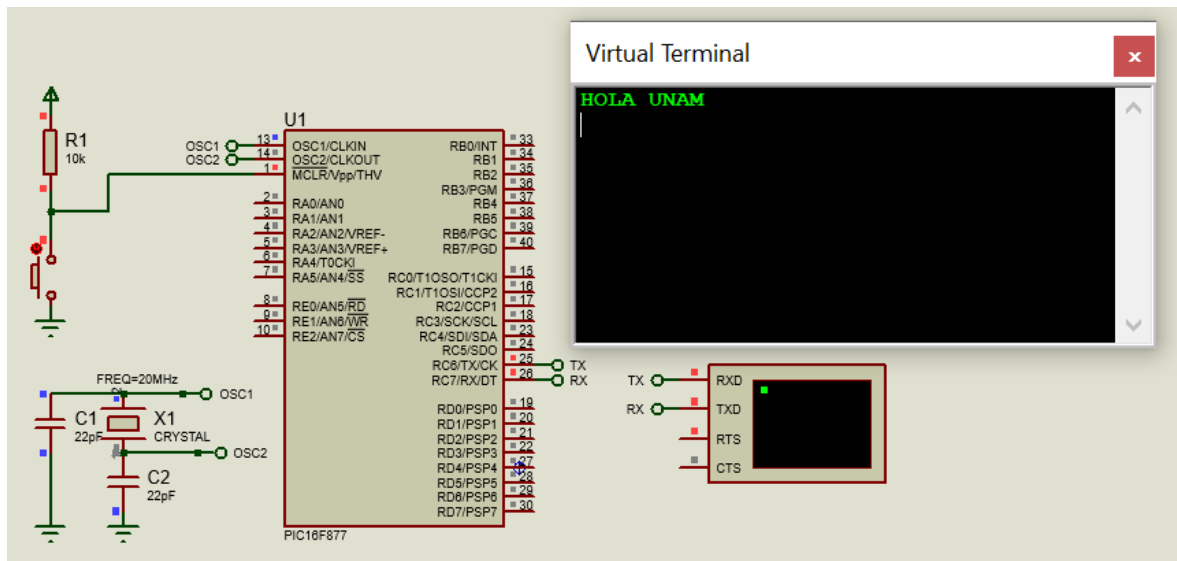


Figura 7.3 Despliegue de una cadena de caracteres

4.- Realizar un programa que ejecute el control indicado; el dato proviene a través del puerto serie:

DATO	ACCION
Puerto Serie	Terminal 0 del puerto B
'0'	0
'1'	1

Tabla 7.1 Control para activar y desactivar una señal

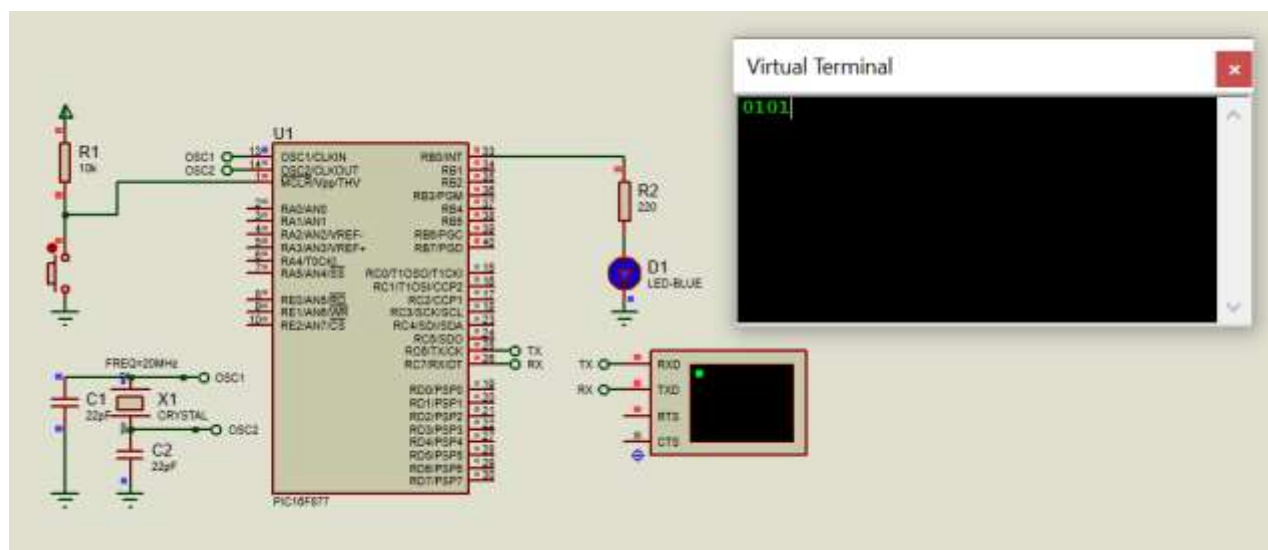


Figura 7.4 Control de un led a través del puerto serie

5.- Realizar un programa que ejecute el control indicado; la secuencia será reconocida cada que sea recibido el comando, usar retardos de ½ segundo entre cada estado generado:

DATO Puerto Serie	ACCION Salida Puerto D
'D' ó 'd'	10000000
	01000000
	00100000
	00010000
	00001000
	00000100
	00000010
	00000001
'I' ó 'i'	00000001
	00000010
	00000100
	00001000
	00010000
	00100000
	01000000
	10000000

Tabla 7.2 Secuencia de control

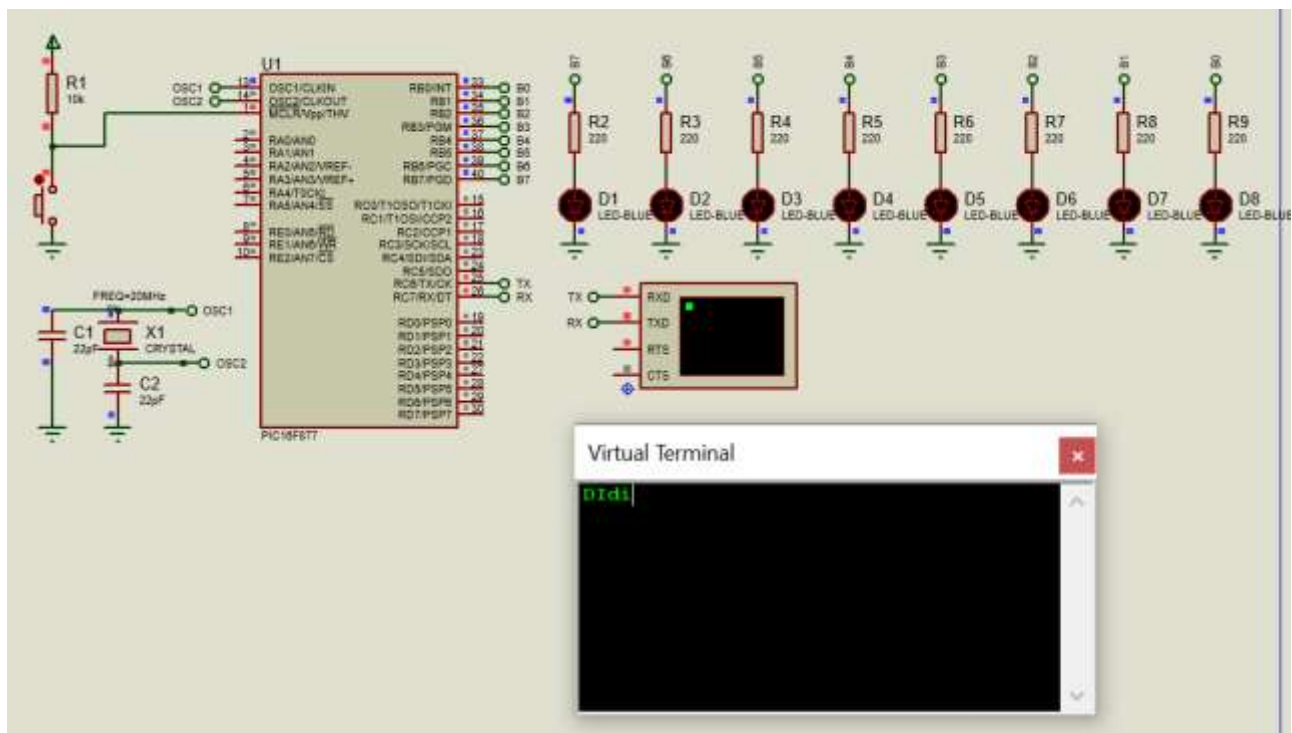


Figura 7.5 Control de secuencias a través del puerto serie

6.- Realizar un programa que permita controlar el funcionamiento de dos motores de corriente directa a través del puerto serie, considerar la tabla 7.3 y 7.4 para la asignación de terminales y la acciones de control respectivamente.

MOTOR2		
PC2	PB3	PB2
ENABLE_M2	DIR1_M2	DIR2_M2

MOTOR1		
PC1	PB1	PB0
ENABLE_M1	DIR1_M1	DIR2_M1

Tabla 7.3. Asignación de señales de control de los motores de CD.

Comando Puerto serie	ACCION	
	MOTOR M1	MOTOR M2
'S'	PARO	PARO
'A'	DERECHA	DERECHA
'T'	IZQUIERDA	IZQUIERDA
'D'	DERECHA	IZQUIERDA
'I'	IZQUIERDA	DERECHA

Tabla 7.4 Control de motores, comunicación serie

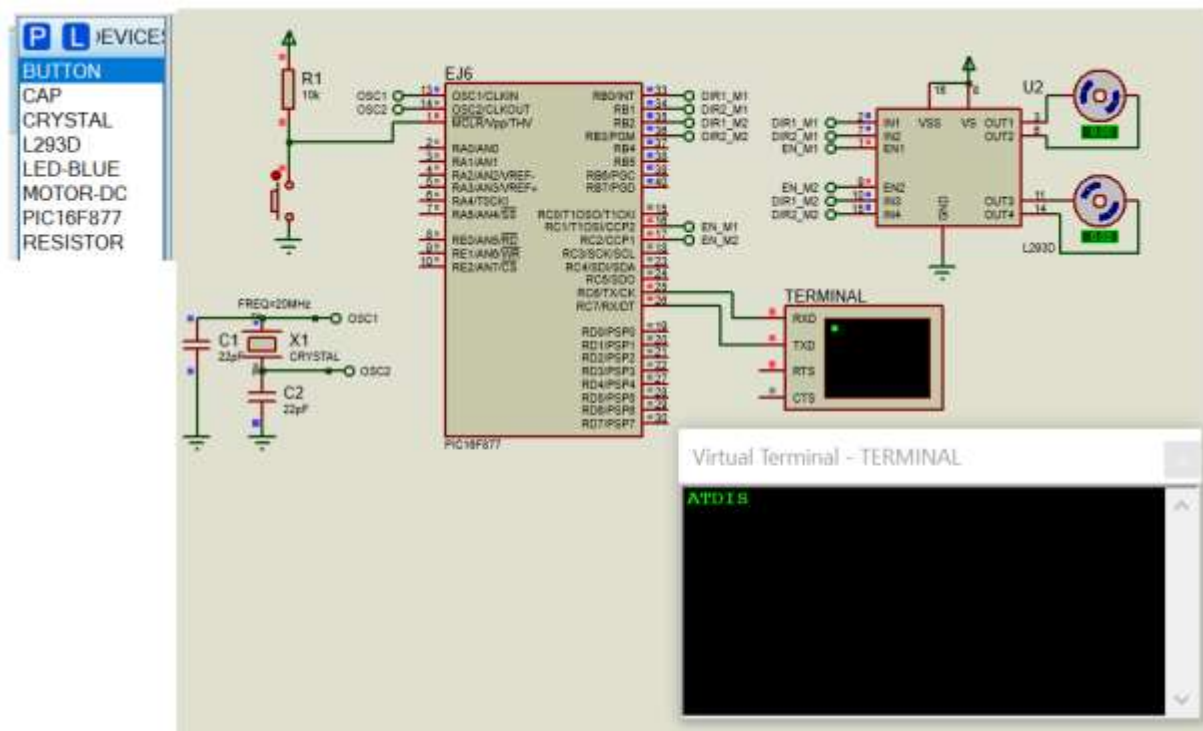


Figura 7.6 Circuito para control de motores a través del puerto serie



7.- Descargar la aplicación practica7.apk e instalar en su dispositivo móvil (Android), realizar un programa para el microcontrolador, de manera que reciba el comando a través del puerto serie, con conexión inalámbrica (bluetooth), usar la misma configuración del ejercicio anterior.

Notas importantes:

- a. Deberá descargar la librería del módulo bluetooth para ser usado en proteus (figura 7.7).
- b. Ubicar los archivos descargados en las carpetas Library y Models contenidos en el directorio de instalación de lab center (figura 7.8).
- c. Agregar el bluetooth a su proyecto y configurar (figura 7.9).
- d. Tendrá dos opciones para comunicarse vía bluetooth a su proyecto:
  - a. Usando una terminal virtual
  - b. Usando el Bluetooth de su computadora
- e. Para vincularse con el dispositivo Bluetooth deberá comprobar su identificador.

BluetoothTEP.IDX

BluetoothTEP.LIB

Figura 7.7 Librería Bluetooth para proteus

WinToUSB (C:) > Archivos de programa (x86) > Labcenter Electronics > Proteus 8 Professional > BIN  
WinToUSB (C:) > Archivos de programa (x86) > Labcenter Electronics > Proteus 8 Professional > DATA > LIBRARY

Figura 7.8 Ruta de instalación de la librería

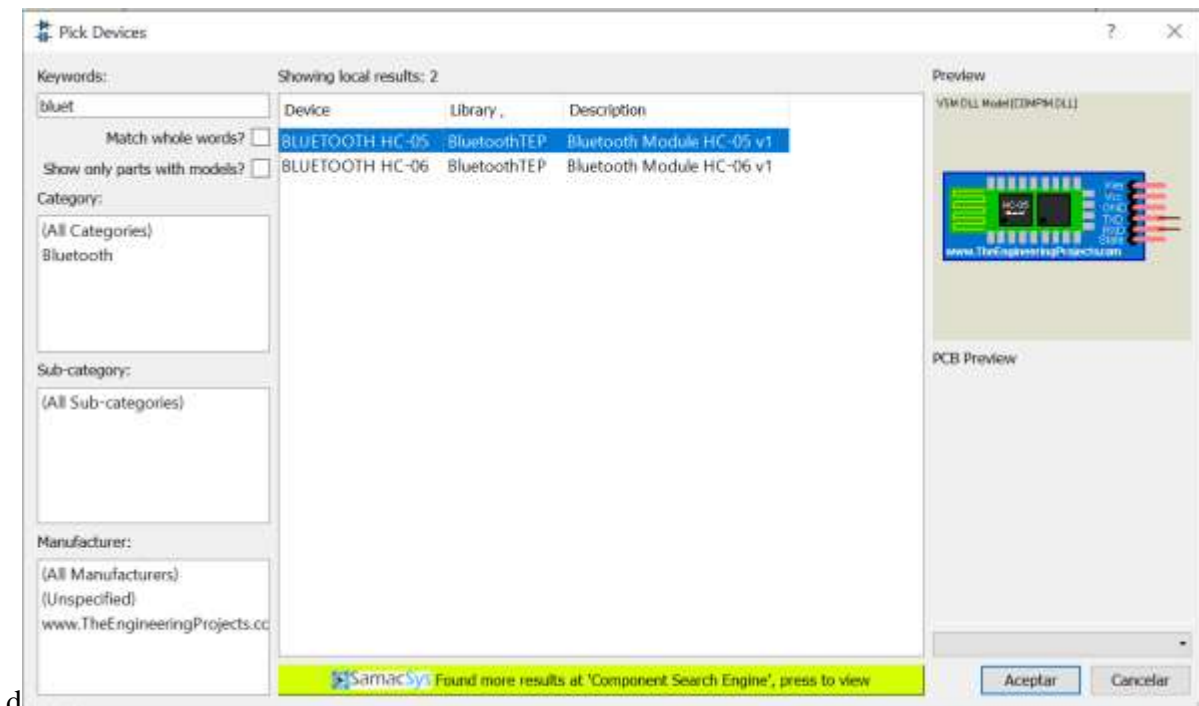


Figura 7.9 Selección de bluetooth en proteus

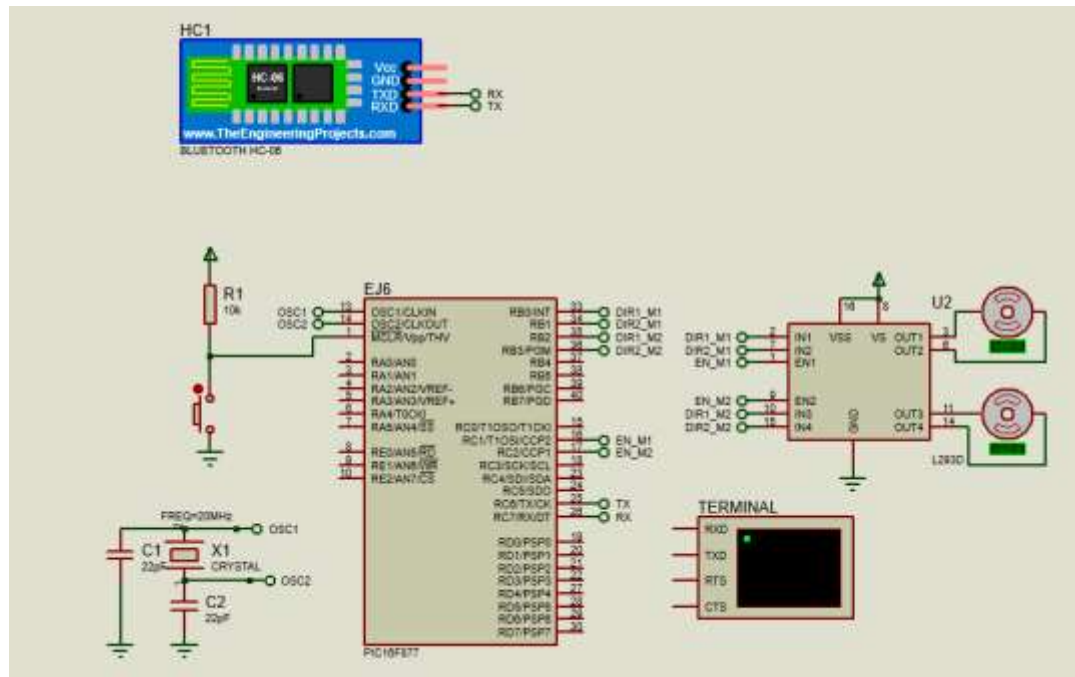


Figura 7.10 Comunicación serie usando módulo bluetooth

8.- Utilizado el termómetro LM35, mostrar la temperatura del ambiente en la terminal de la computadora. La variación del sensor de temperatura es de 100 mV/°C.

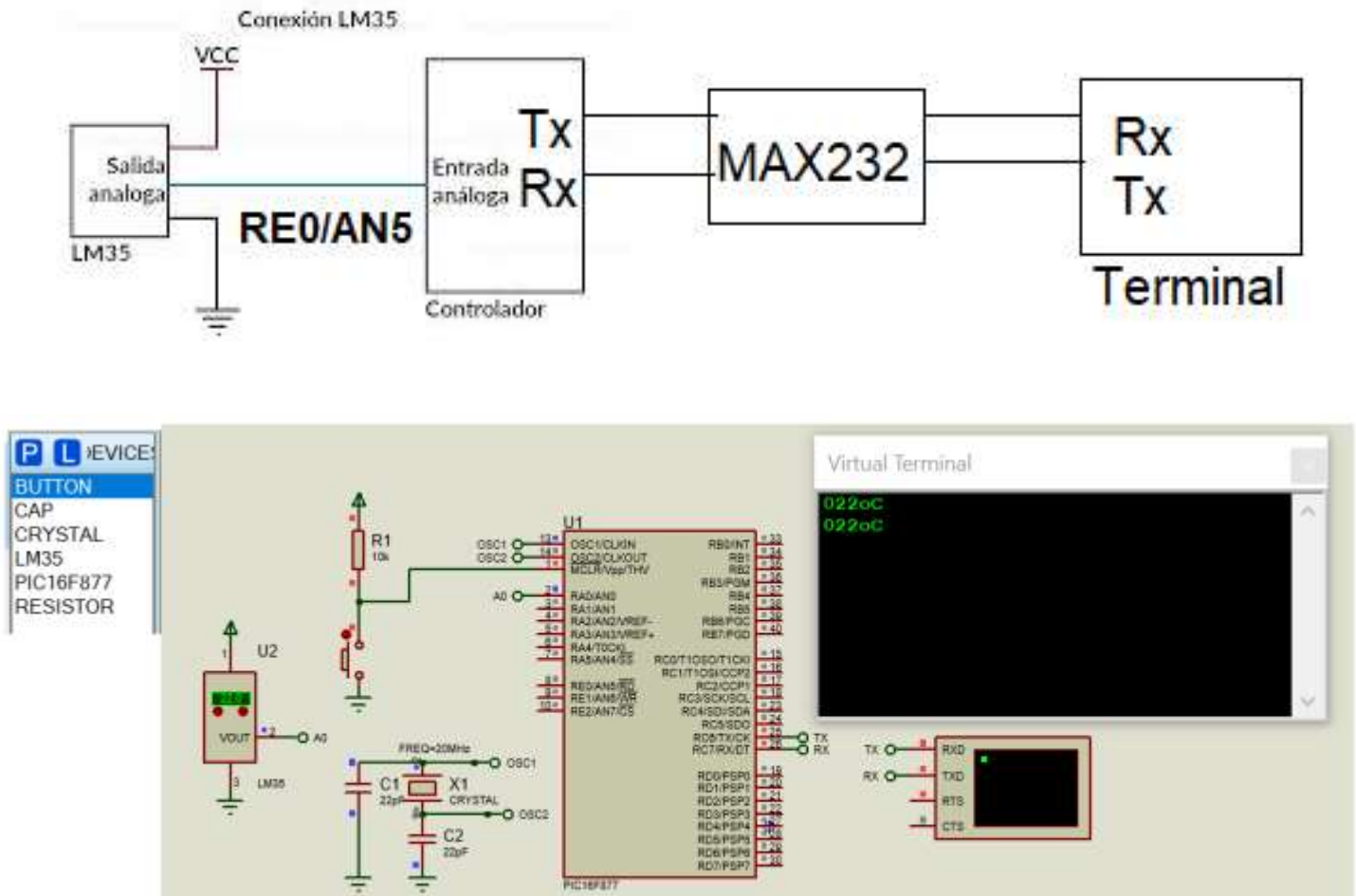


Figura 7.11 Circuito detección y transmisión de la temperatura a través del puerto serie asíncrono.

## Laboratorio de Microcomputadoras

### Práctica No. 8

### Programación en C

### Puertos Paralelos E/S, Puerto Serie

**Objetivo.** Realización de programas a través de programación en C y empleo del puerto serie para visualización y control.

#### Actividades previas

Instalar el compilador de C para el microcontrolador PIC (PIC C Compiler)

#### Introducción

El IDE (Entorno de desarrollo Integrado) del compilador contiene los comandos necesarios para crear un proyecto y compilarlo para posteriormente comprobar el funcionamiento en un sistema real o empleando el simulador Proteus.

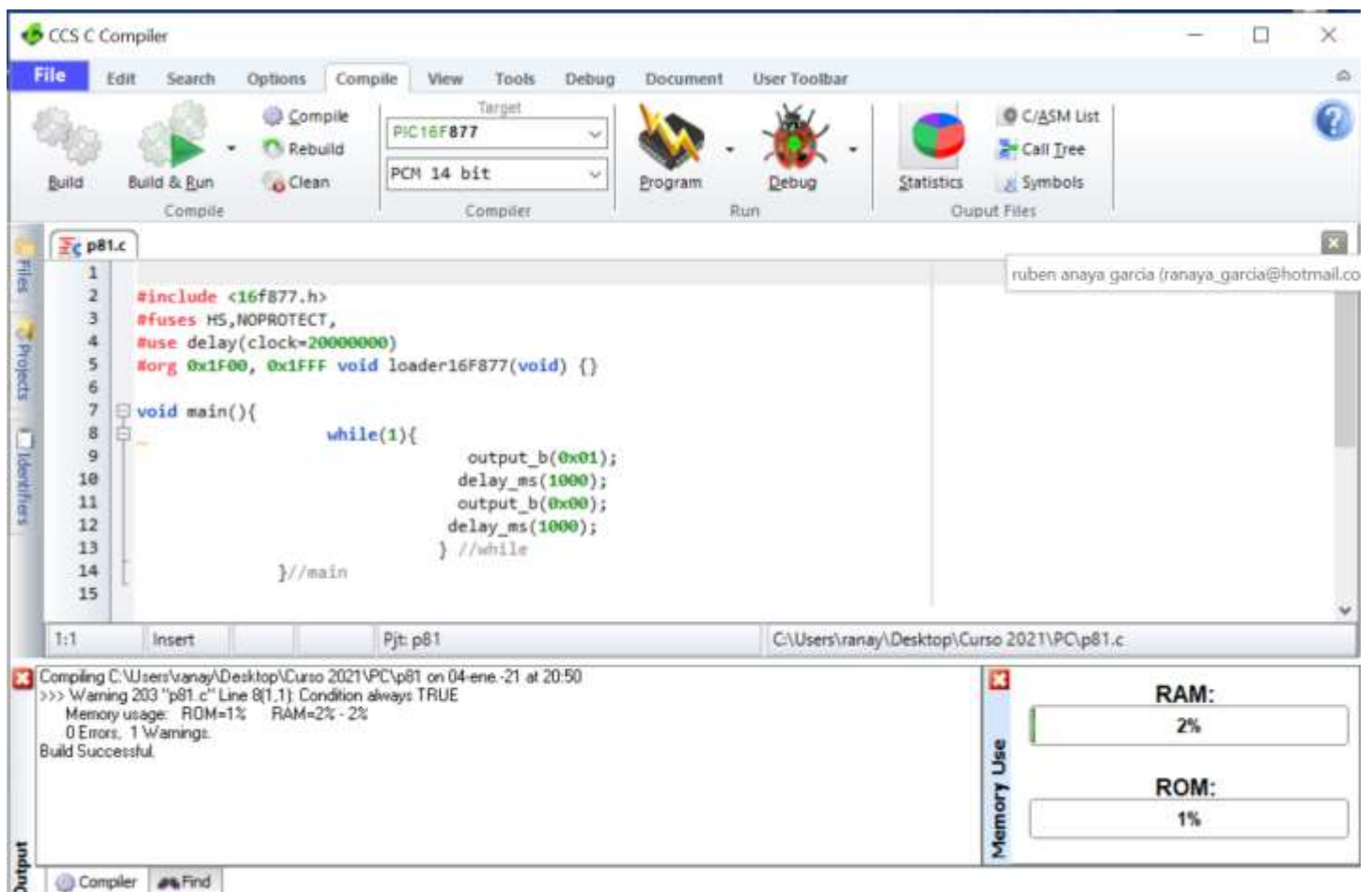


Figura 8.1 IDE del PIC C Compiler

El compilador dispone de una gran variedad de funciones para cada uno de los periféricos del microcontrolador; se recomienda consultar la ayuda del compilador para obtener detalles del uso de las mismas.

Para acceder a la ayuda, seleccionar el icono **?**, ubicado en la parte superior derecha (seleccionar **Built in functions**, para acceder a la ayuda de las funciones)

a. Funciones disponibles para manejo de terminales digitales:

<b>DISCRETE I/O</b>	<a href="#">get tris x()</a>	<a href="#">output X()</a>	<a href="#">output drive()</a>
	<a href="#">input()</a>	<a href="#">output bit()</a>	<a href="#">output low()</a>
	<a href="#">input state()</a>	<a href="#">input change x()</a>	<a href="#">output toggle()</a>
	<a href="#">set tris x()</a>	<a href="#">output float()</a>	
	<a href="#">input x()</a>	<a href="#">output high()</a>	
	<a href="#">port x pullups()</a>		

b. Funciones para manejo del puerto serie asíncrono:

<b>RS232 I/O</b>	<a href="#">assert()</a>	<a href="#">getch()</a>	<a href="#">putc()</a>
	<a href="#">fgetc()</a>	<a href="#">getchar()</a>	<a href="#">putchar()</a>
	<a href="#">fgets()</a>	<a href="#">gets()</a>	<a href="#">puts()</a>
	<a href="#">fprintf()</a>	<a href="#">kbhit()</a>	<a href="#">setup_uart()</a>
	<a href="#">fgetc()</a>	<a href="#">perror()</a>	<a href="#">set_uart_speed()</a>
	<a href="#">fputs()</a>	<a href="#">getc()</a>	<a href="#">printf()</a>
	<a href="#">putc()</a>	<a href="#">scanf()</a>	-

**Desarrollo.** Realizar las siguientes actividades.

1.- Escribir, comentar, compilar el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento, usando el simulador de Proteus.

```
#include <16f877.h>
#fuses HS,NOPROTECT,
#use delay(clock=20000000)
#org 0x1F00, 0x1FFF void loader16F877(void) { }

void main(){
    while(1){
        output_b(0x01);
        delay_ms(1000);
        output_b(0x00);
        delay_ms(1000);
    } //while
} //main
```

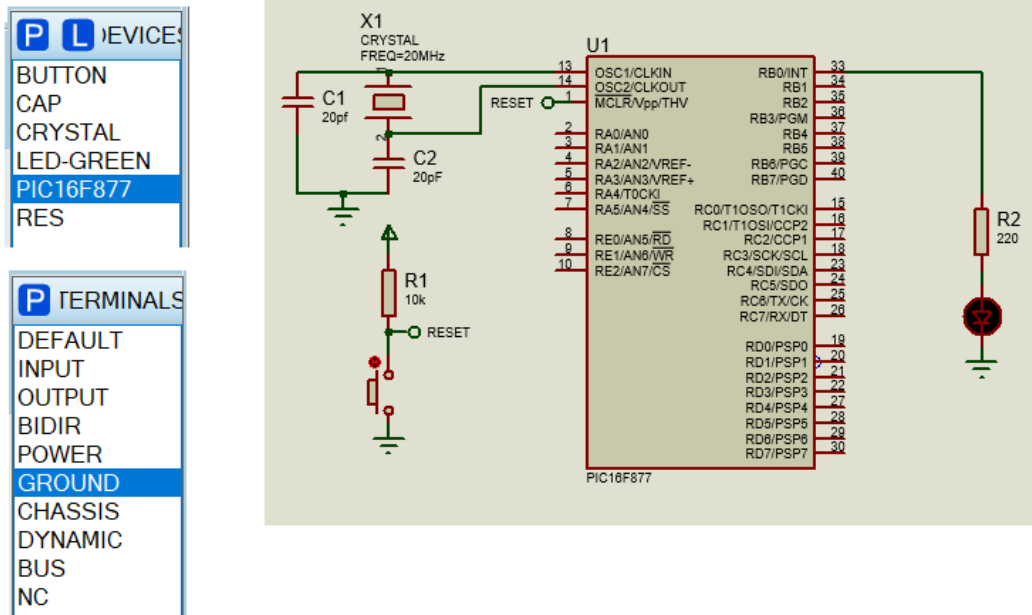


Figura 8.2 Circuito a implementar para la actividad 1

2.- Modificar el programa para que active y desactive todos los bits del puerto B.

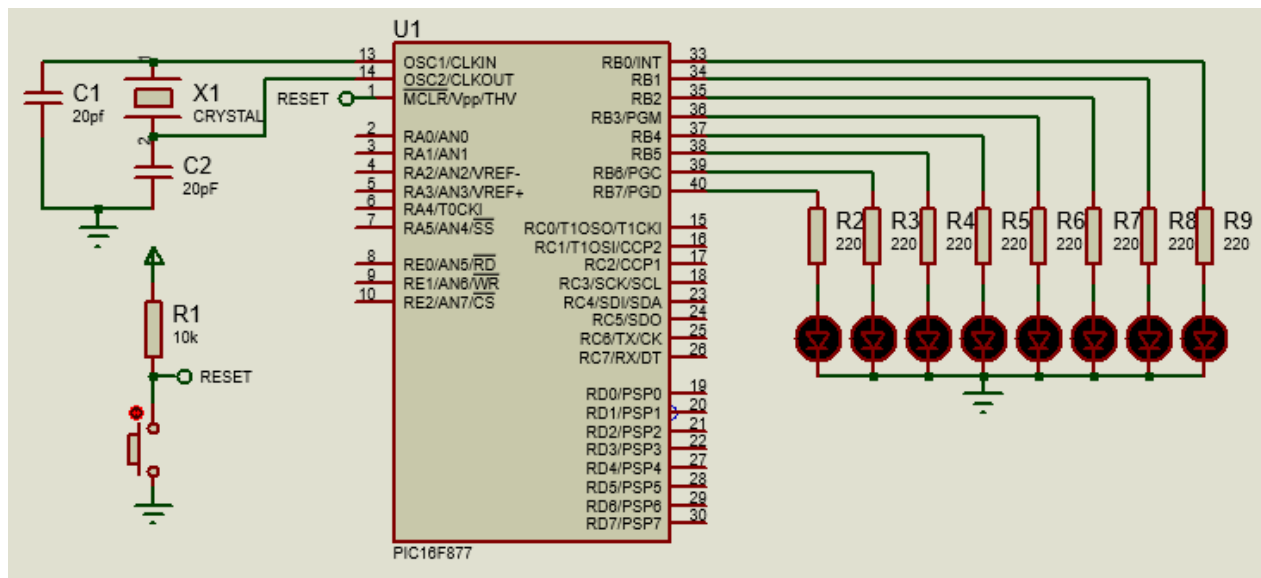


Figura 8.3 Circuito a implementar para la actividad 2

3.- Escribir, comentar, compilar el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

```
#include <16f877.h>
#fuses HS,NOPROTECT,
#use delay(clock=20000000)
#org 0x1F00, 0x1FFF void loader16F877(void) {}

int var1;

void main(){
    while(1){
        var1=input_a();
        output_b(var1);
    }//while
}//main
```

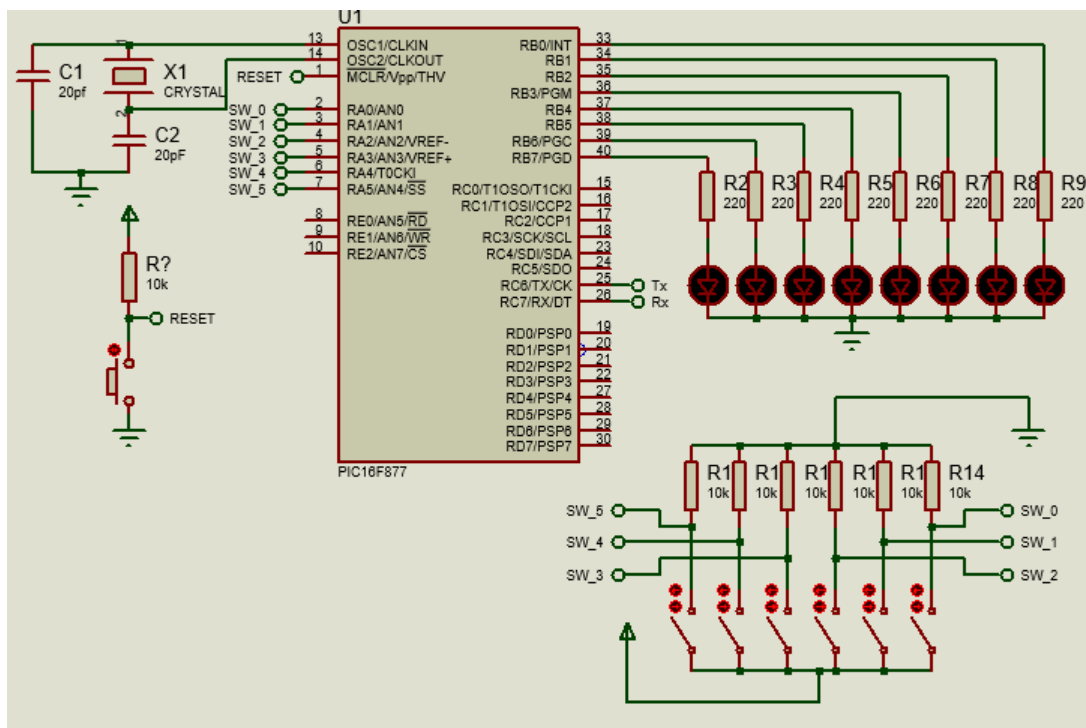


Figura 8.4 Circuito a implementar para la actividad 3



4.- Escribir, comentar, compilar, el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

```
#include <16f877.h>
#fuses HS,NOPROTECT,
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#org 0x1F00, 0x1FFF void loader16F877(void) { }

void main(){
    while(1){
        output_b(0xff); //
        printf(" Todos los bits encendidos \n\r");
        delay_ms(1000);
        output_b(0x00);
        printf(" Todos los leds apagados \n\r");
        delay_ms(1000);
    } //while
} //main
```

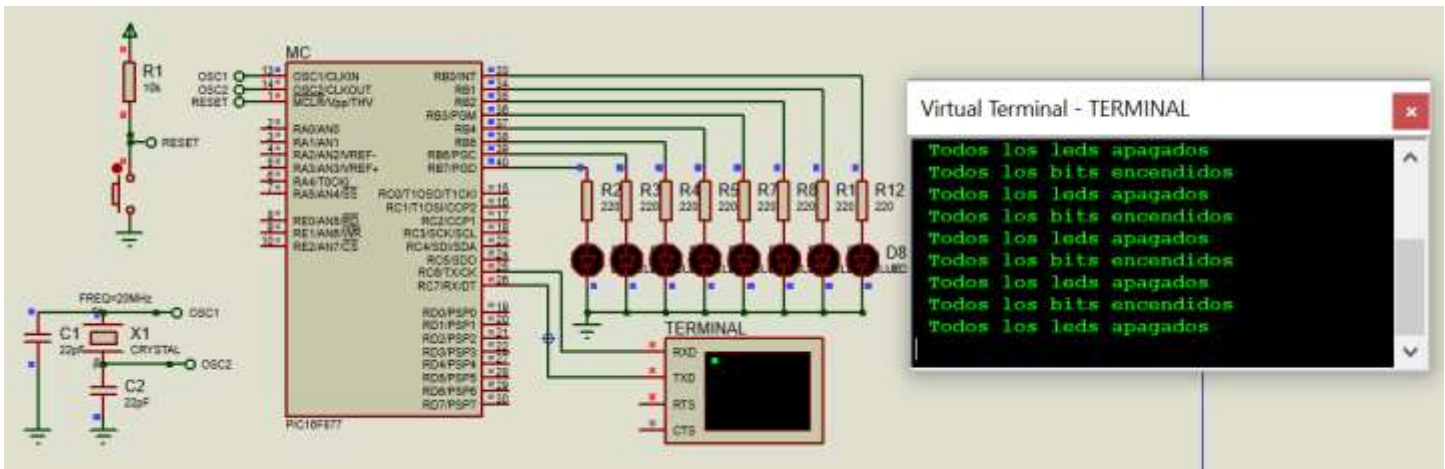


Figura 8.5 Circuito a implementar para la actividad 4

5.- Escribir, comentar, compilar, el siguiente programa usando el ambiente del PIC C Compiler y comprobar el funcionamiento.

**Nota:** La biblioteca lcd.c asigna las terminales para uso del LCD, en la plataforma usada se ha conectado al puerto D; también permite usar el puerto B para las señales de control; en este caso agregar previo a incluir la librería `#define use_portb_lcd true`.



```

#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#include <lcd.c>

void main() {

    lcd_init();

    while( TRUE ) {
        lcd_gotoxy(1,1);
        printf(lcd_putc," UNAM \n ");
        lcd_gotoxy(1,2);
        printf(lcd_putc," FI \n ");
        delay_ms(300);
    }
}

```

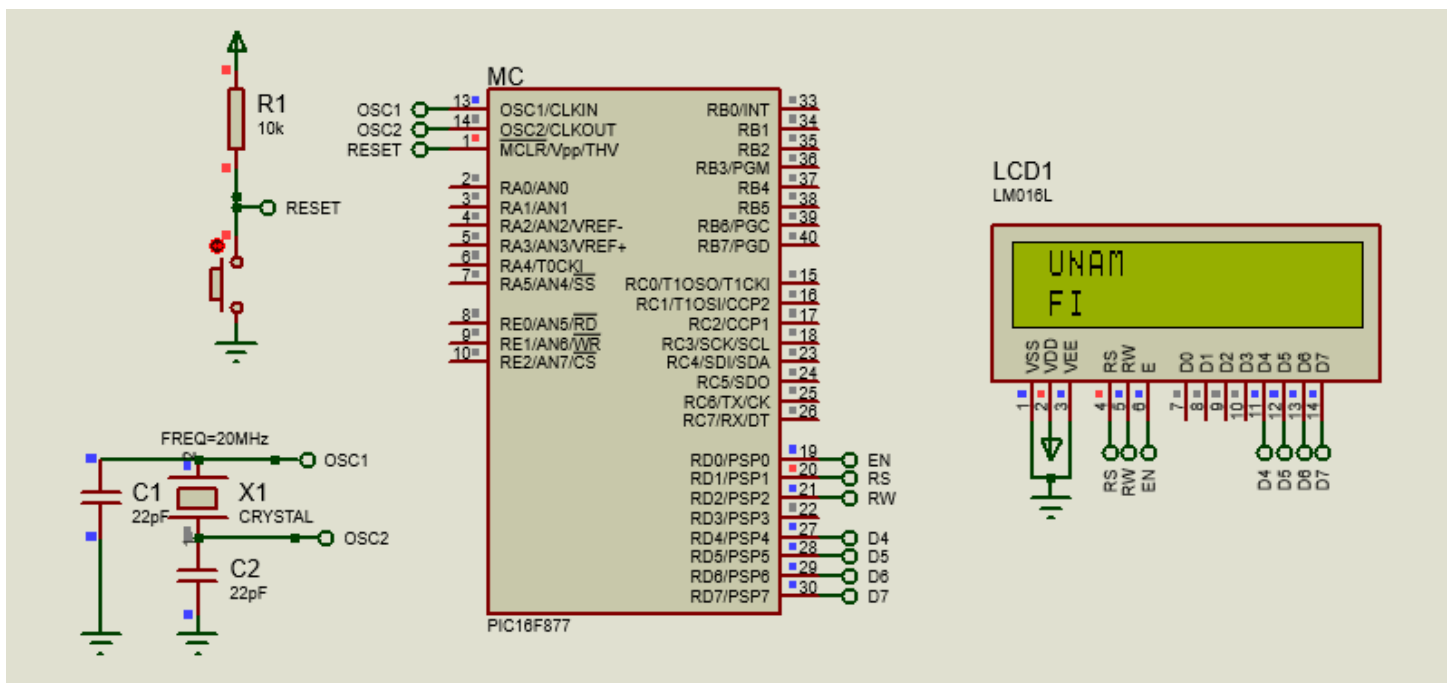


Figura 8.6 Circuito a implementar para la actividad 5

6.- Realizar un programa empleando el compilador de C, para ejecutar las acciones mostradas en la siguiente tabla, estas son controladas a través del puerto serie; usar retardos de ½ segundos.

DATO	ACCION Puerto B	Ejecución
0	Todos los bits apagados	00000000
1	Todos los bits encendidos	11111111
2	Corrimiento del bit más significativo hacia la derecha	10000000 ..... 00000001
3	Corrimiento del bit menos significativo hacia la izquierda	00000001 ..... 10000000
4	Corrimiento del bit más significativo hacia la derecha y a la izquierda	10000000 ..... 00000001 ..... 10000000
5	Apagar y encender todos los bits.	00000000 11111111

Tabla 8.1 Control a través del puerto serie

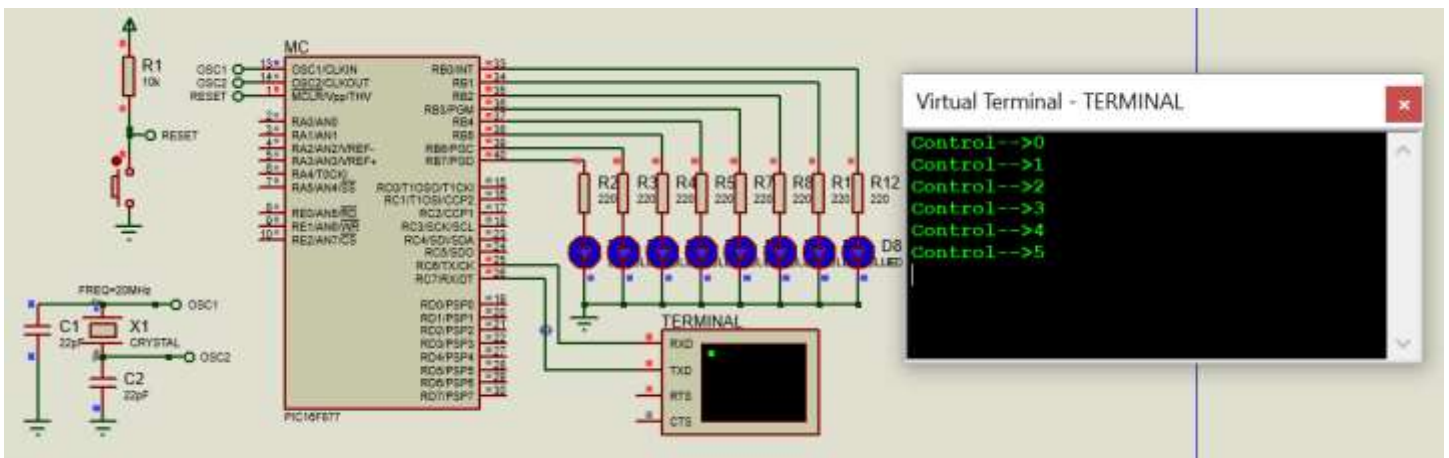


Figura 8.7 Proyecto de control

7.- Realizar un programa que muestre en un Display de Cristal Líquido, la cantidad de veces que se ha presionado un interruptor, el cual se conectará en la terminal A0.

La información a mostrar tendrá el siguiente formato:

- Primer línea y 5 columna; la cuenta en decimal
- Segunda línea y 5 columna; la cuenta en hexadecimal

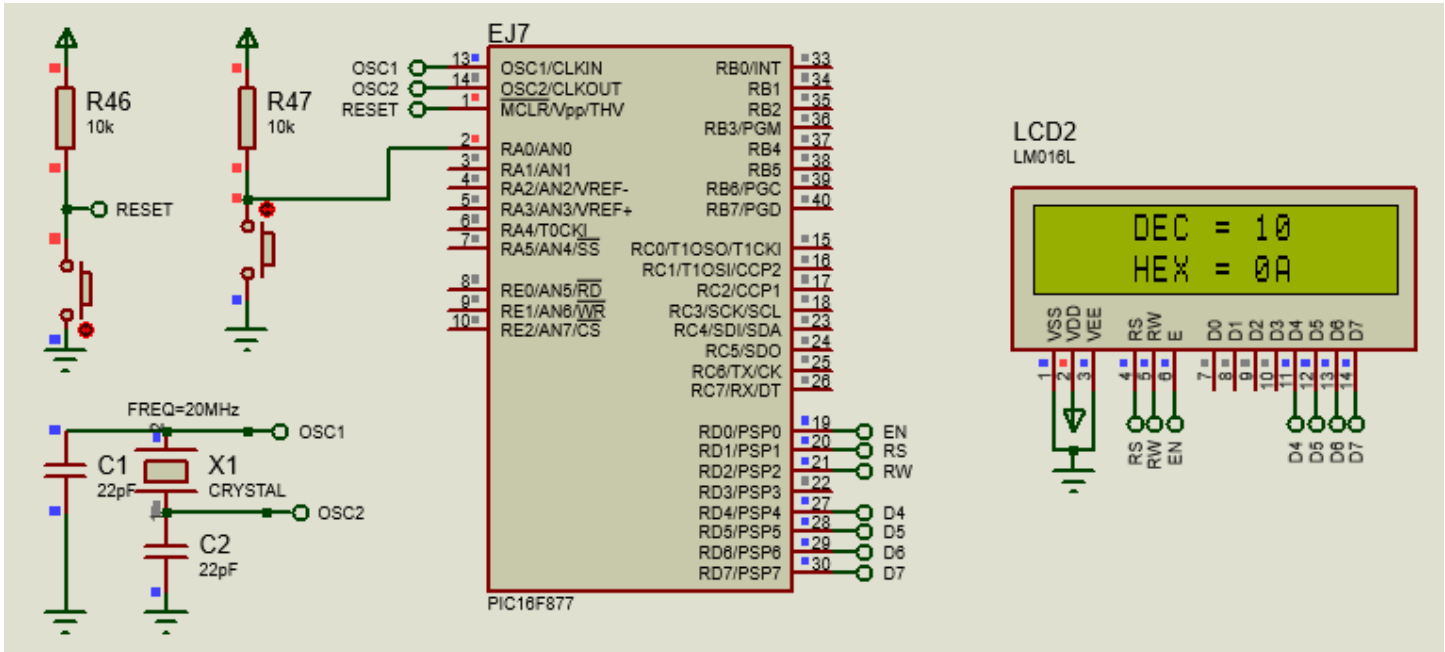


Figura 8.7 Esquemático contador de eventos

**Laboratorio de Microcomputadoras**  
**Práctica No. 9**  
**Programación en C**  
**Comunicación serie síncrona, I2C**

**Objetivo:** El alumno experimentará y reforzará sus conocimientos sobre la comunicación serie síncrona en la modalidad de protocolo I2C, usará el circuito PCF8574 como expensor de puertos, conectado como esclavo para controlar diversos dispositivos.

### Introducción

El bus I2C o I<sup>2</sup>C (Inter-Integrated Circuit) fue desarrollado por Philips, su propósito original fue conectar un microcontrolador de un televisor a diversos periféricos. Se ha adoptado como protocolo de comunicación serial que permite transferencia de datos entre un microcontrolador y dispositivos externos. La tasa de transferencia estándar es de 100Kbps y la más alta puede alcanzar los 3.4 Mbps.

Utiliza dos líneas para la transferencia y recepción de datos:

- SDA** (Serial **D**Ata Line); línea de datos bidireccional.
- SCL** (Serial **C**Lock Line); línea de sincronización ó señal de reloj.

Los dispositivos conectados a estas líneas son de tipo “colector abierto”, por lo tanto debe de conectar resistencias de *pull-up* de 10 K $\Omega$ , así como tener tierras comunes para establecer las mismas referencias entre todos los dispositivos.

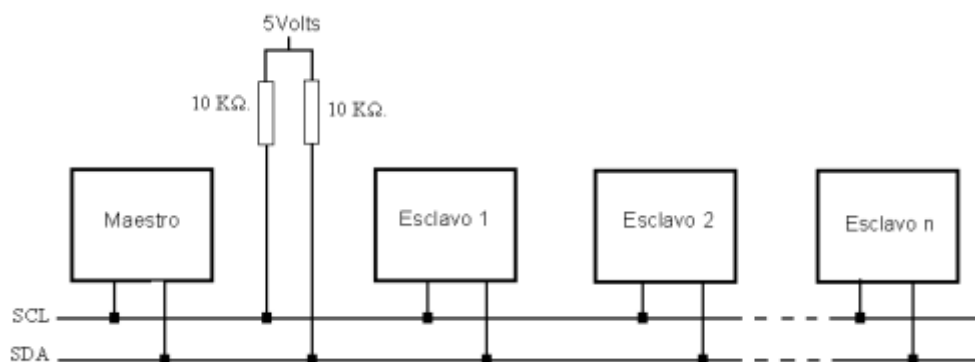


Figura 9.1 Comunicación I2C

Se puede establecer comunicación entre un maestro y uno o varios esclavos, así como en modo multi-maestro. Cada uno de los dispositivos tiene una dirección única, en general el protocolo consta de las siguientes etapas:

- Señal de inicio **START**
- Selección del dispositivo esclavo **ADDRESS**
- Indica acción a realizar lectura o escritura **R/W**
- Respuesta del esclavo **ACK**
- Envío/recepción de información **DATA**
- Señal de paro **STOP**

S	ADDRESS	R/W	ACK	DATA	A	P
	Maestro		Esclavo	Maestro	Esclavo	Maestro

Si R/W= '0' el esclavo escribirá; por lo tanto, el maestro envía información.

Si R/W= '1' el esclavo será leído; por lo tanto, el maestro solicita información.

### Transferencia de datos del maestro al esclavo

El maestro escribirá información al esclavo, la transferencia de datos se describe a continuación:

Transferencia de un dato:

S	ADDRESS	0	ACK	DATA	A	P
	Maestro		Esclavo	Maestro	Esclavo	Maestro

El algoritmo a programar será:

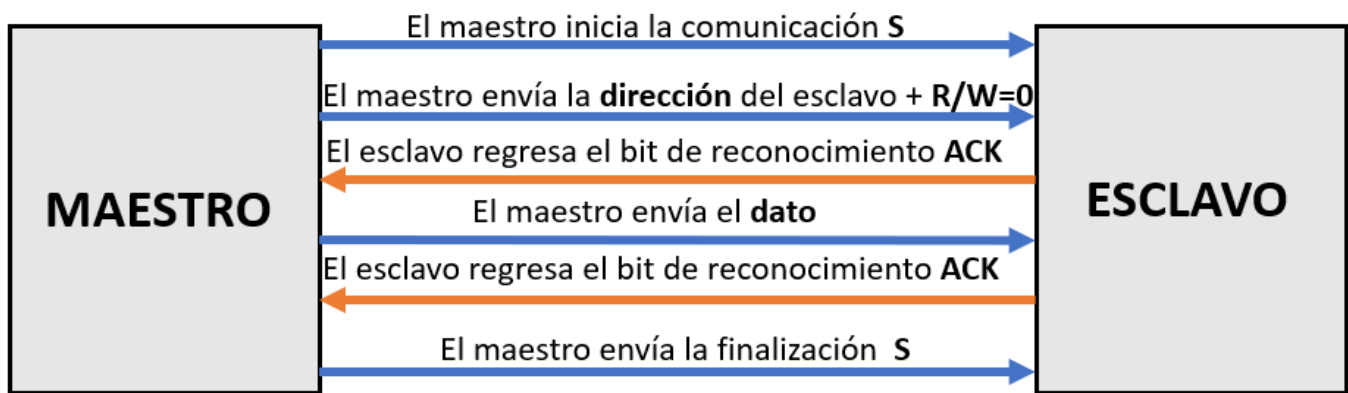


Figura 9.2 Algoritmo I2C

El maestro podrá continuar transferir información, previo al envía del bit de paro S.

Transferencia múltiple:

S	DIR_ESCLAVO	0	A	Dato	A	Dato	A		Dato	A	P
	Maestro		E	M	E	M	E		M	E	M

### Lectura de datos del esclavo

En este caso, el maestro solicitará información al esclavo; el maestro responderá con el bit de reconocimiento cada que reciba información del esclavo a excepción del último dato, en el cuál responderá con un NACK.

Lectura de un dato:

S	ADDRESS	1	ACK	DATA	N	P
Maestro			Esclavo		Maestro	

Lectura múltiple de datos:

S	ADDRESS	1	A	Dato	A	Dato	A	Dato	N	P
Maestro				E	M	E	M	E		M

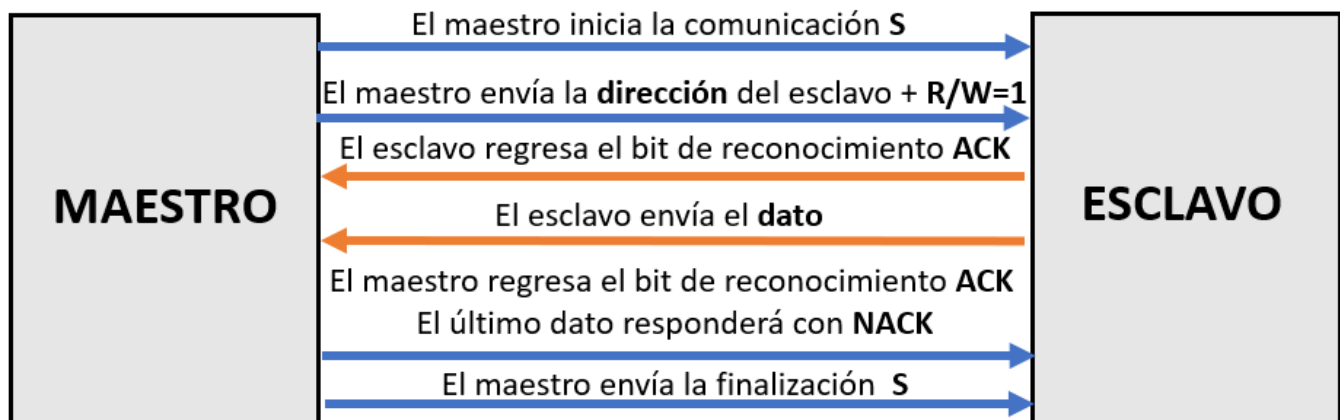


Figura 9.3 Algoritmo para lectura de datos del maestro

La dirección del esclavo podrá ser de 7 0 10 bits, es importante consultar la hoja técnica del dispositivo con el que se desee establecer la comunicación; en la práctica se usará el formato descrito previamente (dirección de 7 bits).

## Comunicación I2C en el PIC

El módulo MSSP del PIC16F877(A) tiene seis registros para la operación del I2C. Asigna en las terminales: **RC3/SCK/SCL** para la señal de reloj y **RC4/SDI/SDA** para los datos; estos son:

- Registro de control SSPCON
- Registro de control 2 SSPCON2
- Registro de estado SSPSTAT
- Registro de transmisión / recepción SSPBUF
- Registro de corrimiento SSPSR
- Registro de dirección SSPADD

Se describen brevemente, consultar la hoja técnica para mayor información.

### Registro de estado 94h

SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
---------	-----	-----	-----	---	---	-----	----	----

Define el tipo de velocidad en la comunicación:

- Estándar
- Alta
- Contiene banderas del estado que guarda la comunicación serie síncrona

### Registro de control 14h

SPBRG	WCOL	SSPOV	SSPEN	CLKP	SSPM3	SSPM2	SSPM1	SSPM0
-------	------	-------	-------	------	-------	-------	-------	-------

Registro de configuración:

- Habilita la comunicación serie asíncrona
- Define la configuración I2C
- Configura la función del microcontrolador, ya sea maestro o esclavo
- Define el formato de direccionamiento (7 o 10 bits)

### Registro de control 2 91h

SPBRG	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
-------	------	---------	-------	-------	------	-----	------	-----

Registro de configuración:

- Contiene las banderas que seleccionan la acción que será realizada
  - Habilita la condición START, STOP, START repetido
  - Habilita la recepción de datos en modo maestro
  - Habilita los bits de reconocimiento ACK

### Registro Transmisión/Recepción de datos 13h

SSPBUF	7						0
--------	---	--	--	--	--	--	---

Registro en el cual se envía o recibe el dato, de acuerdo a la acción realizada

### Registro configuración de dirección/velocidad 93h

SSPADD	7						0
--------	---	--	--	--	--	--	---

- a. En modo esclavo
  - a. Almacena la dirección del esclavo
- b. En modo maestro
  - a. Define la velocidad de transmisión que será usada en SCL

Cálculo de la velocidad:

$$f_{SCL} = f_{osc} / 4 \times (SSPADD + 1)$$

$f_{osc}$  : Frecuencia del microcontrolador

**SSPADD:** 7 bits menos significativos del registro

### Funciones y recursos del compilador de C para empleo del protocolo I2C.

Directiva

#use I2C (configuraciones)

Las configuraciones más empleadas son:

MASTER	Configura modo Maestro
SLAVE	Configura modo Esclavo
SCL=PIN	Define el pin a usar para SCL
SDA=PIN	Define el pin a usar para SDA
ADDRESS=dirección	Especifica la dirección del esclavo
FAST	Configura velocidad alta
SLOW	Define velocidad baja
FORCE_HW	Emplea funciones por hardware

Ejemplos:

- a. Configura modo maestro, asigna C4 para SDA y C3 para SCL, así como velocidad baja.
  - a. #use I2C(MASTER,SDA=PIN\_C4, SCL=PIN\_C3, SLOW)



- b. Configura modo esclavo, asigna C4 para SDA y C3 para SCL, velocidad baja, con dirección 0xa0.
  - a. #USE I2C(SLAVE, SDA=PIN\_C4, SCL=PIN\_SDA, SLOW, ADDRESS=0X0A)

Funciones disponibles:

- a. Inicia la comunicación I2C, enviando el bit S
  - a. I2C\_START();
- b. Escribe un dato o la dirección del esclavo en el bus; cuando se use para direccionar al esclavo, los 7 bits más significativos indicarán la dirección establecida y el ultimo bit indicará la modalidad de escritura o lectura: A6,A5,A4,A3,A2,A1,A0,**R/W**. Por ejemplo; para un dispositivo cuya dirección esté especificada en su hoja técnica como: 1010000**R/W**:
  - a. I2C\_WRITE(ADDRESS);
    - i. Cuando sigue a una función I2C\_START()
      - 1. Modo escritura
        - a. I2C\_WRITE(0XA0);
      - 2. Modo lectura
        - a. I2C\_WRITE(0XA1);
    - ii. Cuando sigue a una función I2C\_WRITE
      - 1. Modo escritura
        - a. I2C\_WRITE(DATO);
- c. Finalizará la comunicación; enviará el bit de paro
  - a. I2C\_STOP();
- d. El maestro realiza una petición al esclavo (modo lectura)
  - a. Lectura de datos:
    - i. Cuando reciba un dato, regresa un ACK
      - 1. dato=I2C\_READ();
    - ii. Cuando reciba el último dato, regresa un NACK
      - 1. dato=I2C\_READ(0);
- e. El esclavo recibe peticiones del maestro (modo escritura)
  - a. Regresa un TRUE en caso de haber recibido un dato
    - i. I2C\_POLL();
  - b. Lectura del dato recibido
    - i. dato=I2C\_READ();

## Expansor de puertos PCF8574

### Descripción

Este dispositivo es un expansor de puertos que permite comunicarse vía I2C con un microcontrolador maestro.

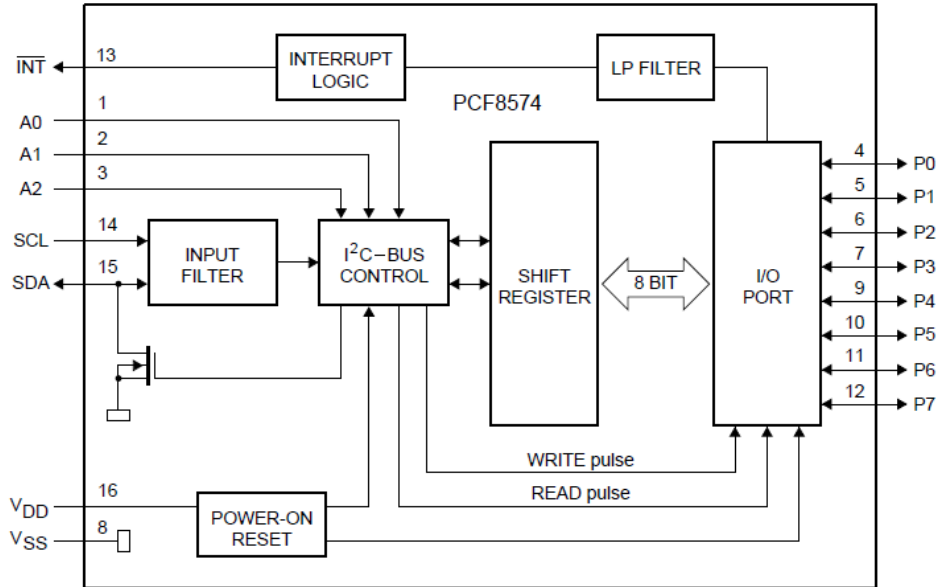
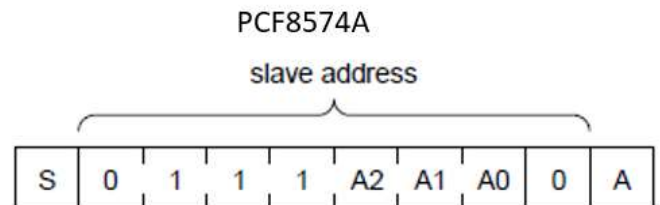
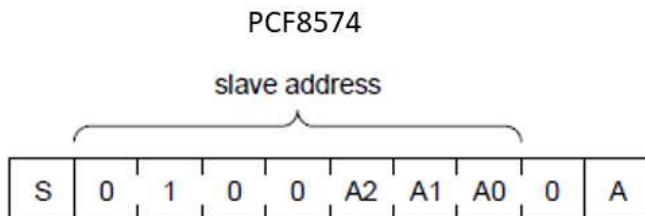


Figura 9.4 Diagrama general del PCF8574

Este circuito tiene las siguientes terminales:

- Terminal de datos SDA
- Terminal de reloj SCL
- Terminal de interrupciones INT, para generar una petición de interrupción cuando suceda una transición en el expansor de puertos
- VDD y VSS para alimentación de circuito 5V y 0V respectivamente
- Tres terminales para configurar la parte baja de la dirección que se asignará A2, A1 y A0.
- Ocho terminales bidireccionables P7...P0

La dirección de memoria es configurada colocando los niveles lógicos A0, A1 y A2 para conformar la dirección de 7 bits; existen dos modelos de este circuito: PCF8574 y PCF8574A, cuya única diferencia radica en los valores fijos de los bits más significativos.



En la práctica es empleado el modelo PCF8574 y la dirección configurada se indica en la actividad correspondiente.

## Desarrollo

1.- El objetivo del siguiente programa será para mayor comprensión de la comunicación I2C y la programación en C, por lo que se pide analizarlo y comentarlo para su reporte; observar en el circuito la conexión de A2, A1 y A0 para generar la dirección del esclavo, así como su uso en el programa.

```
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use i2c(MASTER, SDA=PIN_C4, SCL=PIN_C3,SLOW, NOFORCE_SW)
int contador=0;

void escribir_i2c(){
    i2c_start();
    i2c_write(0x42);
    i2c_write(contador);
    i2c_stop();
}

void main()
{
    while(true)
    {
        escribir_i2c();
        delay_ms(500);
        contador++;
    }
}
```

El circuito se describe a continuación: Considerar que se usa un decodificador BCD-7SEG.

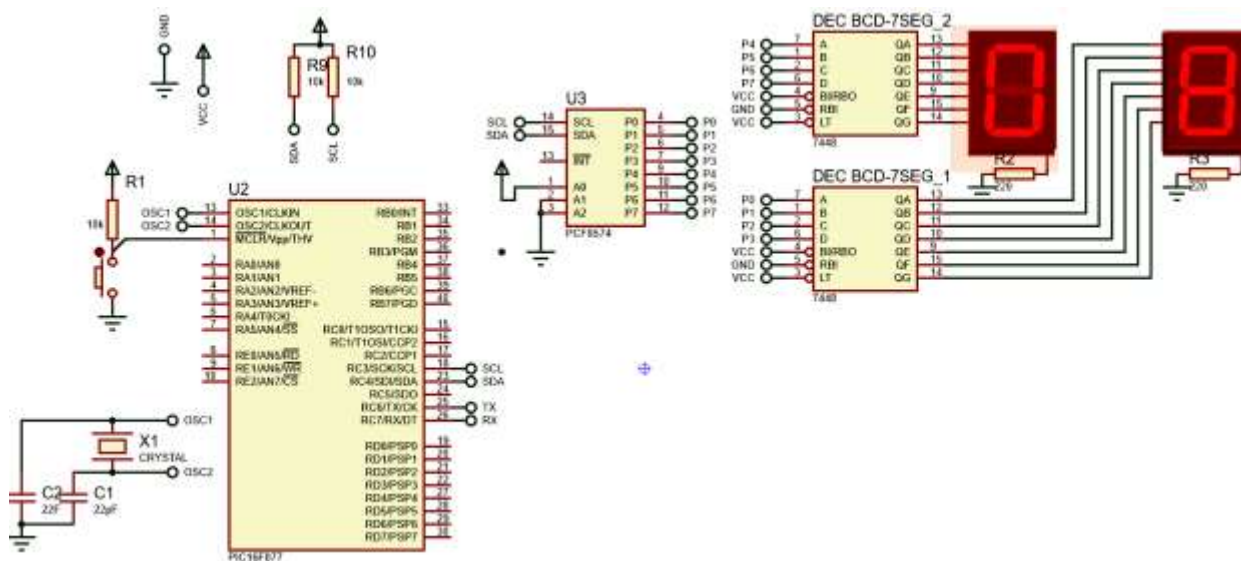


Figura 9.5 Circuito actividad 1; un maestro y un esclavo

Proteus tiene un componente llamado 7SEG-BCD el cuál tiene integrado un decodificador que permite desplegar números en formato BCD y hexadecimal, se deja la libertad de utilizar el componente deseado, en esta actividad y las subsecuentes, el proyecto queda de la siguiente manera (figura 9.6).

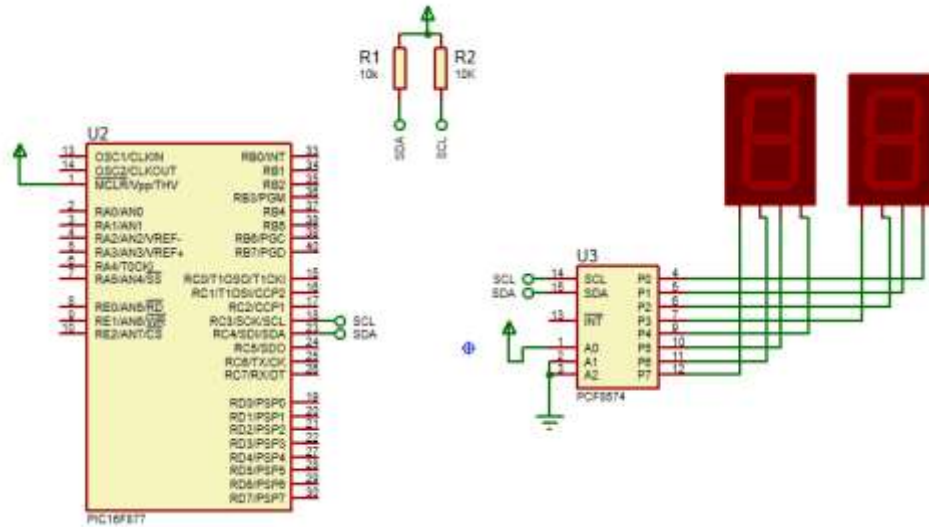


Figura 9.6 Uso de 7SEG-BCD

2.- Realizar la modificación al programa para que también muestre el contador en el puerto B.

Nota: Considerar que se usará un decodificador BCD-7SEG; por lo que deberá modificar el programa para ver la cuenta continua.

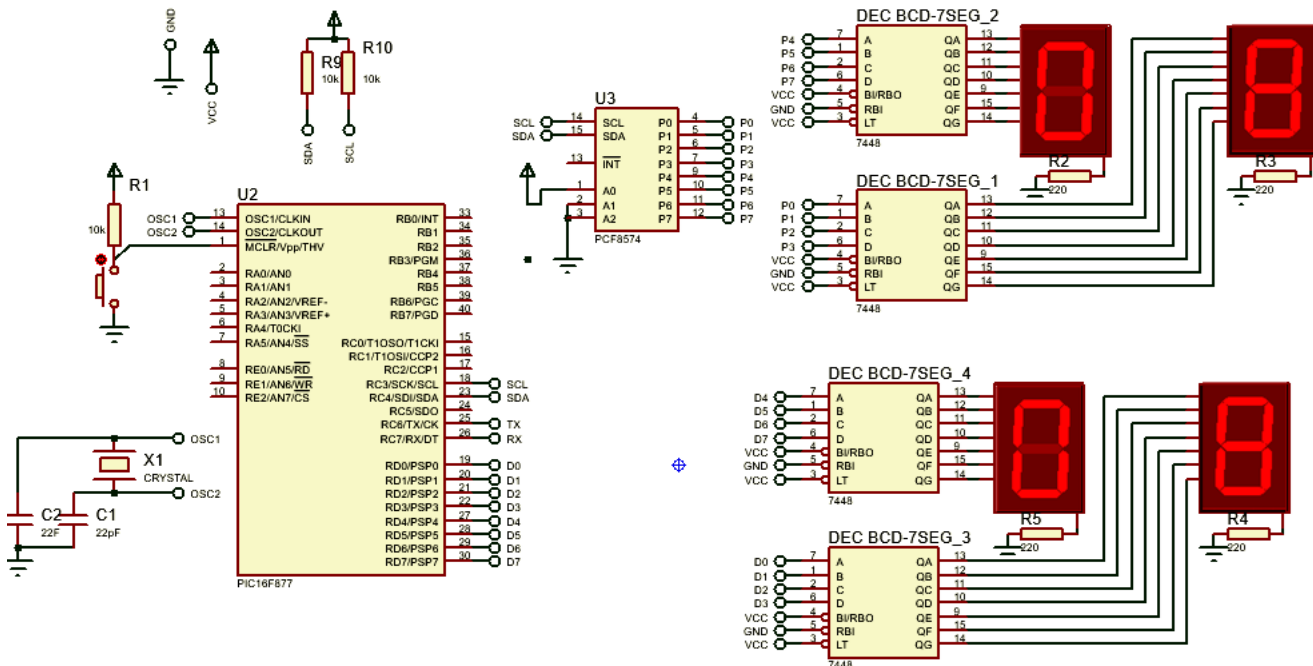


Figura 9.7 Circuito actividad 2; un maestro y un esclavo

3.- Realizar las modificaciones necesarias para que además de lo resuelto en el ejercicio previo, muestre el contador en un display LCD que funcionará como esclavo I2C.

Consideraciones:

- Debe incluir la librería `i2c_LCD.c` a su programa; esta librería contiene el protocolo de comunicación I2C para uso del Display de Cristal Líquido LCD.
- La biblioteca `I2C_LCD` permite emplear con los mismos nombres las funciones empleadas para el LCD en formato paralelo.
- Es necesario incluir la función que configura la forma de comunicación I2C del LCD.
  - `lcd_init(DIRECCION_ESCLAVO,COLUMNAS,RENGLONES);`
    - DIRECCION\_ESCLAVO:** es la dirección configurada por los valores fijos de fábrica y los definidos por hardware del módulo PCF8574 que controla al LCD; ubicar en el esquemático la configuración, para obtener la dirección.
    - COLUMNAS:** es la cantidad de columnas de LCD
    - RENGLONES:** es la cantidad de filas disponibles en el LCD
    - En la práctica se usará LCD de 16x2
- Uso DEC BCD-7SEG

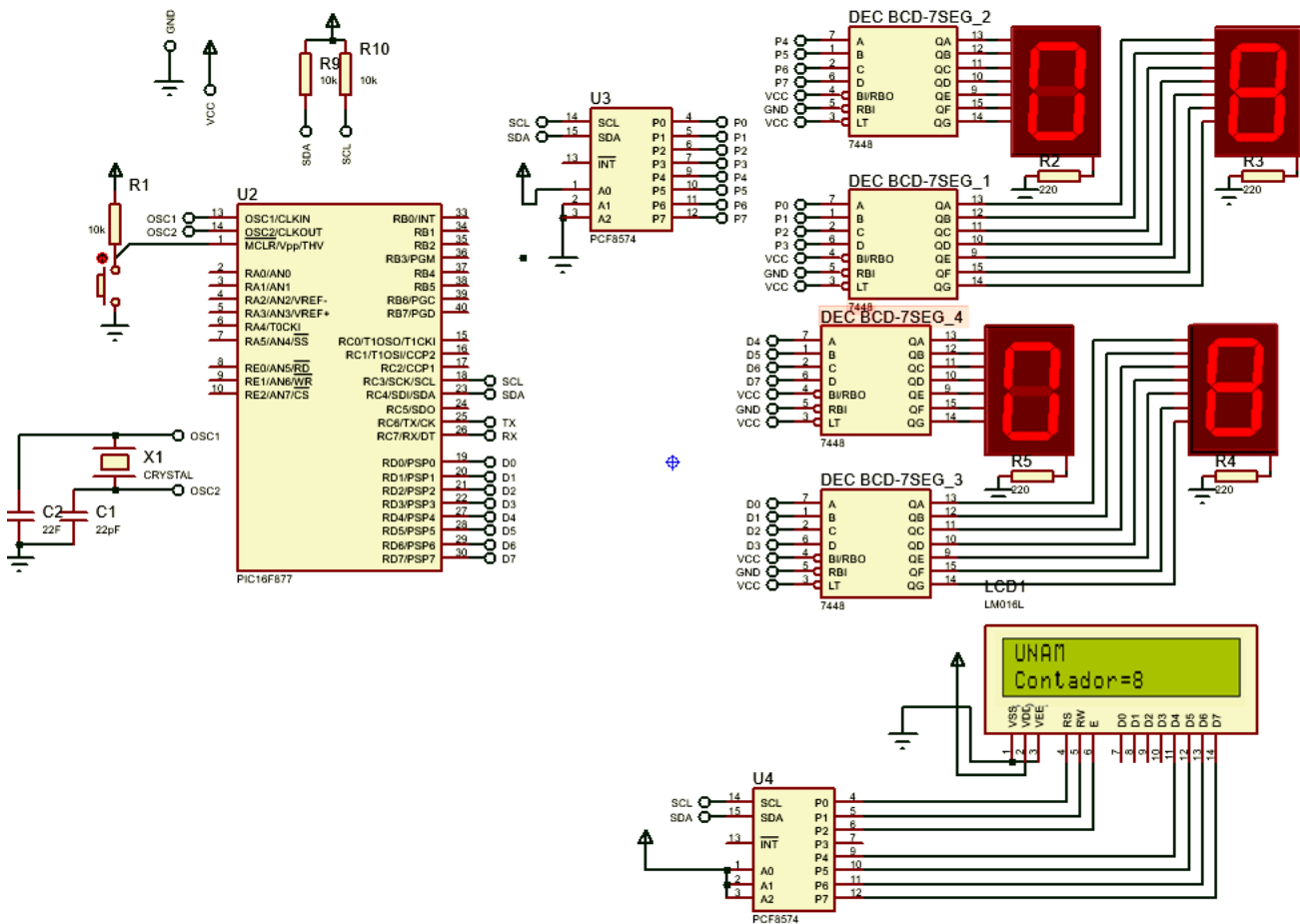


Figura 9.8 Circuito actividad 3; un maestro y dos esclavos

4.- Realizar un programa de tal forma que obtenga la lectura de la entrada generada por otro dispositivo esclavo y los muestre en los tres periféricos usados en la actividad 3.

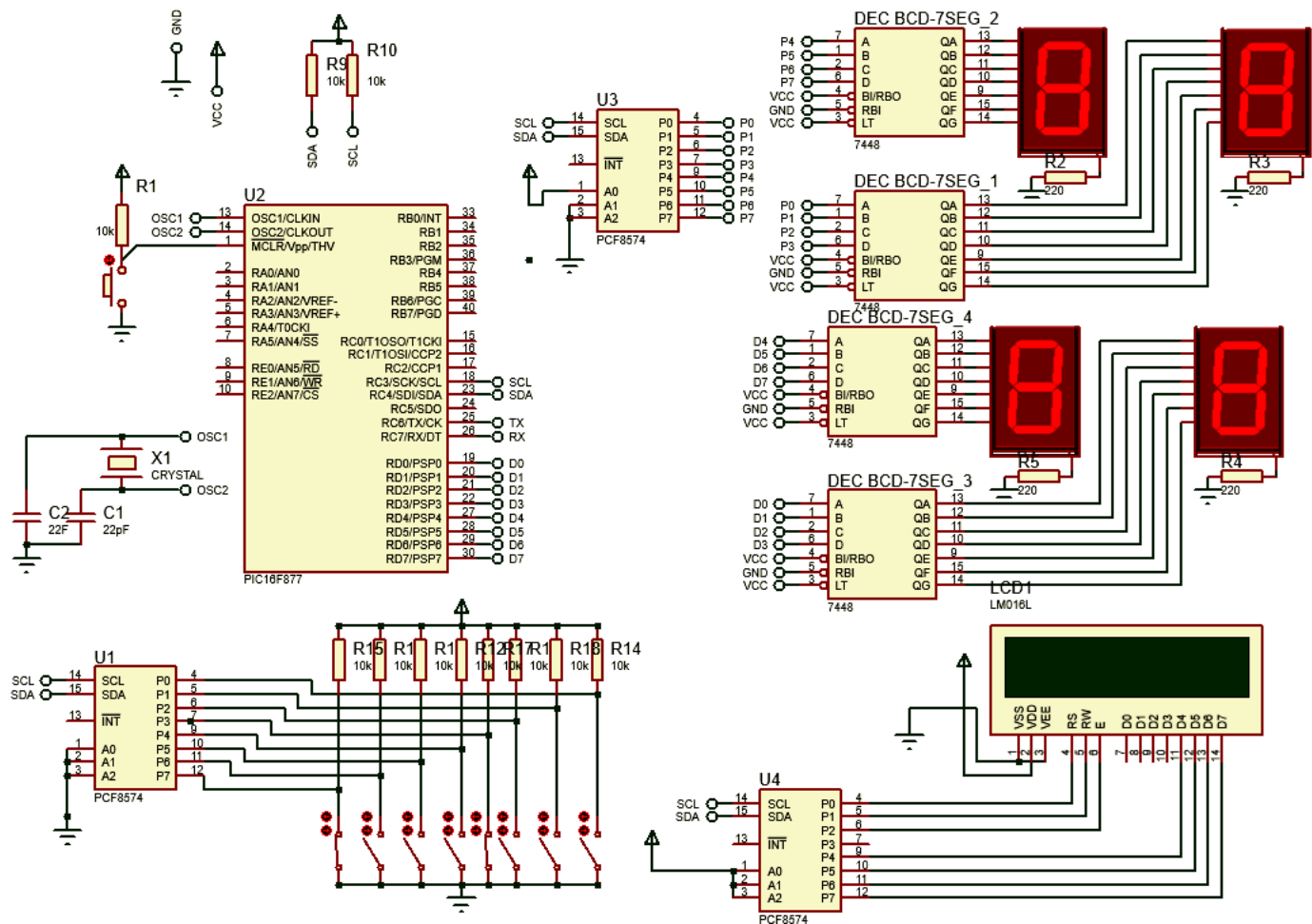


Figura 9.9 Circuito actividad 4; un maestro y tres esclavos

---

**Laboratorio de Microcomputadoras**  
**Práctica No. 10**  
**Programación C**  
**Convertidor A/D, Temporizadores e Interrupciones**

---

**Objetivo.** Realización de programas usando programación en lenguaje C, utilización del puerto serie, convertidor analógico digital e introducción a aplicaciones con interrupciones.

## Introducción

### Convertidor A/D

Tal como se estudió y experimentó en la practica 6, el PIC16F877(A) contiene un convertidor A/D de 10 bits, dispone de 8 canales ubicados en los puertos A y E.

Las funciones y directivas a utilizar son:

- a. `#device ADC=Tamaño del resultado;` agregar después de `#include <16f877.h>`
  - a. `#device ADC=8` // resolución de 8 bits
  - b. `#device ADC=10` // resolución de 10 bits
- b. `Setup_port_a(ALL_ANALOG);` // Define el puerto A como analógico
- c. `Setup_adc(ADC_CLOCK_INTERNAL);` // Define frecuencia de muestreo del convertidor A/D
- d. `Set_adc_channel(num);` // Configura el canal a usar
- e. `Delay_us(20);` // Retardo de 20  $\mu$ s
- f. `Read_adc();` // Obtener el resultado de la conversión

### Interrupciones

Una interrupción es la solicitud al procesador para dejar momentáneamente la ejecución secuencial del programa para atender una petición de interrupción, una vez que se ha reconocido, deberá almacenar el entorno del procesador en la pila, para ser recuperado previo a salir de la rutina de interrupción, para continuar con las instrucciones pendientes. El procesador tiene 14 fuentes de interrupción, estas son:

- a. Desbordamiento del TIMER0
- b. Detección de flanco de RB0
- c. Cambio de nivel de RB4-RB7
- d. Desbordamiento del TIMER1
- e. TIMER2
- f. Módulo CCP1 (Captura y comparación)
- g. Módulo CCP2 (Captura y comparación)
- h. Escritura en memoria EEPROM
- i. Comunicación Serial Síncrona
- j. Colisión del bus en la Comunicación Serial Síncrona
- k. Recepción de datos en la Comunicación Serial Asíncrona
- l. Transferencia de datos en la Comunicación Serial Asíncrona
- m. Terminación completa del convertidor A/D
- n. Transferencia paralela de datos

Para ser atendida una petición de interrupción se debe cumplir con los siguientes requisitos (usando el compilador de C):

- a. Habilitar interrupciones particulares
- b. Habilitar interrupciones generales
- c. Definir la rutina de interrupción

Funciones en C para control de interrupciones:

- a. Habilitar interrupciones particulares
  - enable\_interrups(FUENTE)
- b. Habilitar interrupciones generales
  - enable\_interrups(GLOBAL)
- c. Definir la rutina de interrupción; incluir previo a la función principal (main)

```
#int_evento // Fuente de interrupción
Función_de_interrupción(){
    //código de la rutina de interrupción
}
```

- Fuentes de interrupción usadas en esta práctica
  - #INT\_RB ; //Cambio de nivel de los cuatro bits más significativos del puerto B
  - #INT\_RTCC; //Desbordamiento del TIMER0
  - #INT\_RDA; //Recepción de datos por el puerto serie asíncrona
  - #INT\_EXT; // Detección de flanco por RB0

## TIMER0

El TIMER0 puede ser empleado para trabajar bajo dos modalidades:

- a. Temporizador
- b. Acumulador de pulsos ocurridos en RA4

Como temporizador incrementa el registro de 8 bits **TMR0**, cada  $x$  cantidad de ciclos de reloj, establecido como **pre\_divisor** (desde 2 hasta 256), cuando este registro pasa de 0xFF a 0x00 se prende la bandera **TOIF**; con lo que se genera la petición de interrupción.

- set\_timer0(0) ;inicia el timer0 en 0x00
- setup\_counters(RTCC\_INTERNAL,RTCC\_DIV\_256);
  - Configura el tiempo de activación de TOIF (**TTOIF**)
    - **TTOIF**=**TINT**(255)(256)
      - **TINT** : es el periodo de oscilación interna (0.2  $\mu$ s ; para oscilador de 20 MHz)
      - 255 es el pre-escalador del TIMER0
      - 256 la cantidad de pulsos requeridos para generar el desbordamiento del TIMER0, a partir del valor inicial establecido en la función set\_timer0(0)



- Con la configuración de la función descrita y sustituyendo valores
  - **T<sub>TOIF</sub>**= 13.10 ms

- `enable_interrupts(INT_RTCC);` habilita la interrupción TIMER0

Declaración de la interrupción del TIMER 0:

```
#int_rtcc
clock_isr(){
    //código de la rutina
}
```

### **Interrupción por cambio de nivel de los pines RB4 a RB7**

Se puede generar la petición de interrupción cuando cualesquiera de los pines RB4, RB5, RB6 o RB7 cambia de nivel; ya sea de alto a bajo o de bajo a alto.

- Funciones para uso de PB4 al PB7
  - `enable_interrupts(INT_RB);` //Habilita interrupción por cambio de nivel en los cuatro bits más significativos del puerto B

Declaración de interrupción por cambio de nivel RB4-RB7

```
#int_rb
port_rb(){
    //código de la rutina de interrupción
}
```

### **Interrupción por detección de flanco de RB0**

Cuando ocurre un flanco ya sea de subida o de bajada previamente definida, el procesador podrá generar la petición de interrupción.

Funciones empleadas:

`ext_int_edge(flanco);` // Configura el flanco a detectar; donde flanco= H\_TO\_L o L\_TO\_H

Definición de la rutina de interrupción por detección de flanco de RB0

```
#int_ext
detecta_rb0(){
    //código de la rutina de interrupción
}
```

### **Interrupción por recepción de datos del puerto serie asíncrono**

Configuración de la comunicación asíncrona:

```
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
```

Definición de la rutina de interrupción por recepción de datos por el puerto serie asíncrono.

```
#int_rda
recepción_serie(){
    //código de la rutina de interrupción
}
```

Finalmente, el programa que empleará las interrupciones tendrá la siguiente forma:

```
#include <16f877.h>
#device adc=8 //en caso de emplear el conv. A/D indica resolución de 8 bits
...           //configuración general
...
           //declaración de variables
#int_rtcc // rutina de interrupción del timer0
clock_isr(){
    //código de la rutina de interrupción
}

main()
{
    set_timer0(0); // Inicia TIMER0 en 00H
    setup_counters(RTCC_INTERNAL,RTCC_DIV_256); //Fuente de reloj y pre-divisor
    enable_interrupts(INT_RTCC); //Habilita interrupción por TIMER0
    enable_interrupts(GLOBAL); //Habilita interrupciones generales

    while(1){
        //programa principal
    }
}
```

**Desarrollo.** Realizar los siguientes ejercicios.

1.- Escribir, comentar, indicar que hace; comprobar el funcionamiento del siguiente programa en el simulador de Proteus.

```
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
```

```
int contador;
```

```
#int_EXT
ext_int()
{
    contador++;
    output_d(contador);
}
```

```
void main() {
    ext_int_edge(L_TO_H);
    enable_interrupts(INT_EXT);
    enable_interrupts(GLOBAL);
    output_d(0x00);
    while( TRUE ) {}
```

```
}
```

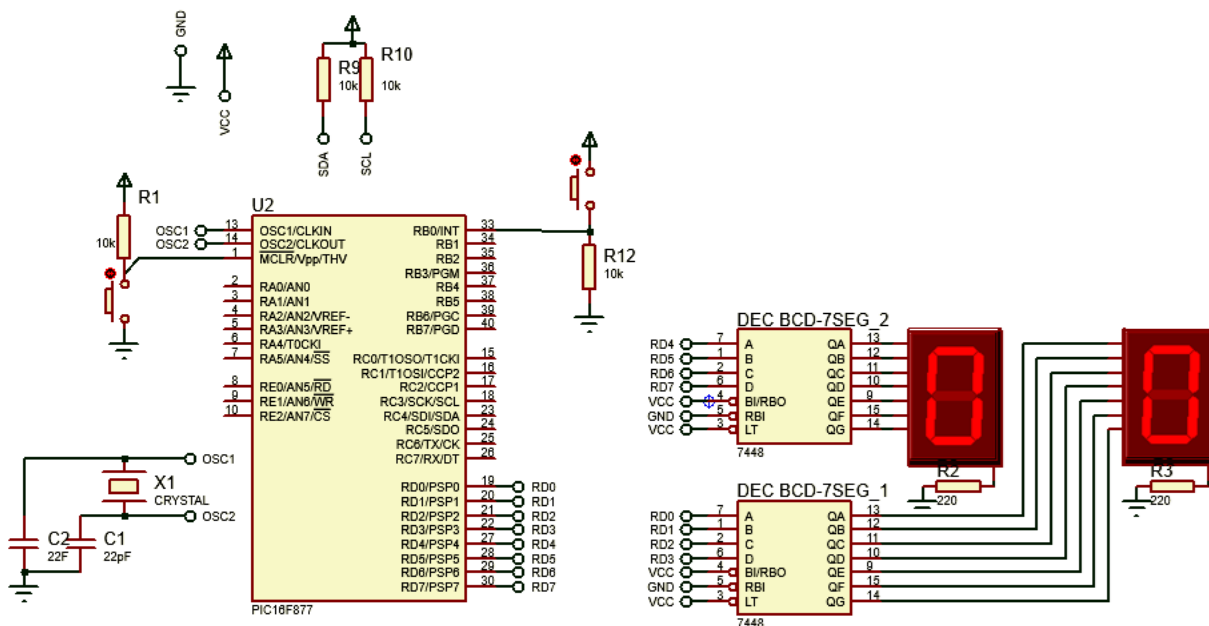


Figura 10.1 Circuito actividad 1

2.- Escribir un programa, el cual obtenga una señal analógica a través del canal de su elección; el resultado de la conversión deberá ser desplegado en tres diferentes dispositivos, de acuerdo a la tabla 9.1.

El resultado debe ser desplegado de acuerdo a:

Periférico	Formato del despliegue	Dispositivo	Formato del despliegue
Puerto paralelo	Decimal	Disp. 7SEG	Visible 00 - 99
I2C	Voltaje	LCD	Vin= 5.00 V
Puerto serie	Decimal, hexadecimal	Terminal	Decimal=1023, Hexadecimal=0x3FF

Tabla 10.1 Formatos de resultados y periféricos

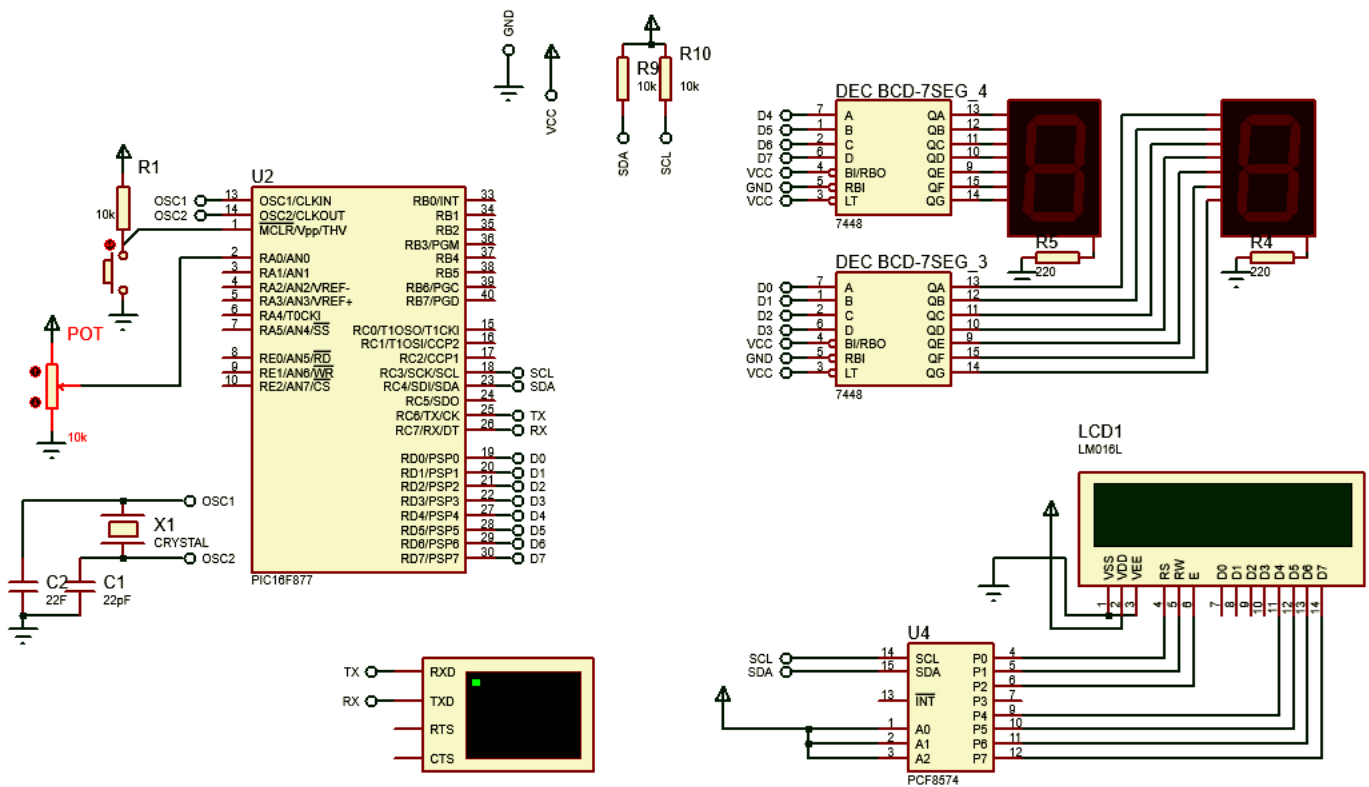


Figura 10.2 Circuito actividad 2; Entrada analógica, display 7 SEG, LCD I2C, Terminal.

3.- Utilizando la interrupción del TIMER0, realizar un programa que transmita el resultado de la conversión cada 10 segundos, usar el mismo formato del ejercicio anterior.

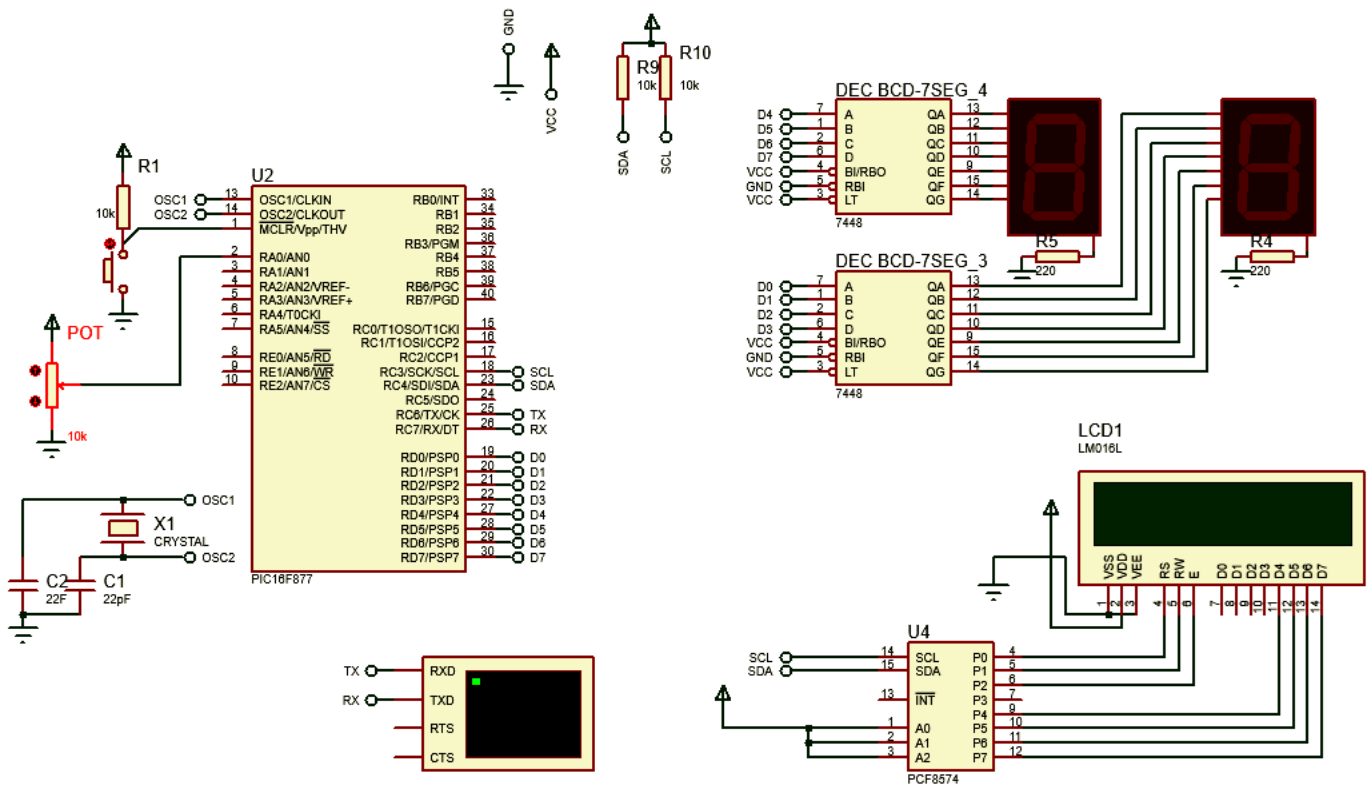


Figura 10.3 Circuito actividad 3.

4.- Realizar un programa que muestre un contador binario de 8 bits en un puerto paralelo (usar leds y retardos de 250 ms), cada 10 segundos muestre el voltaje de la señal que ingrese en el canal 0 del convertidor A/D en el LCD y cada 25 segundos despliegue en la terminal los nombres, número de cuenta, grupo de teoría y laboratorio del o los integrantes del equipo.

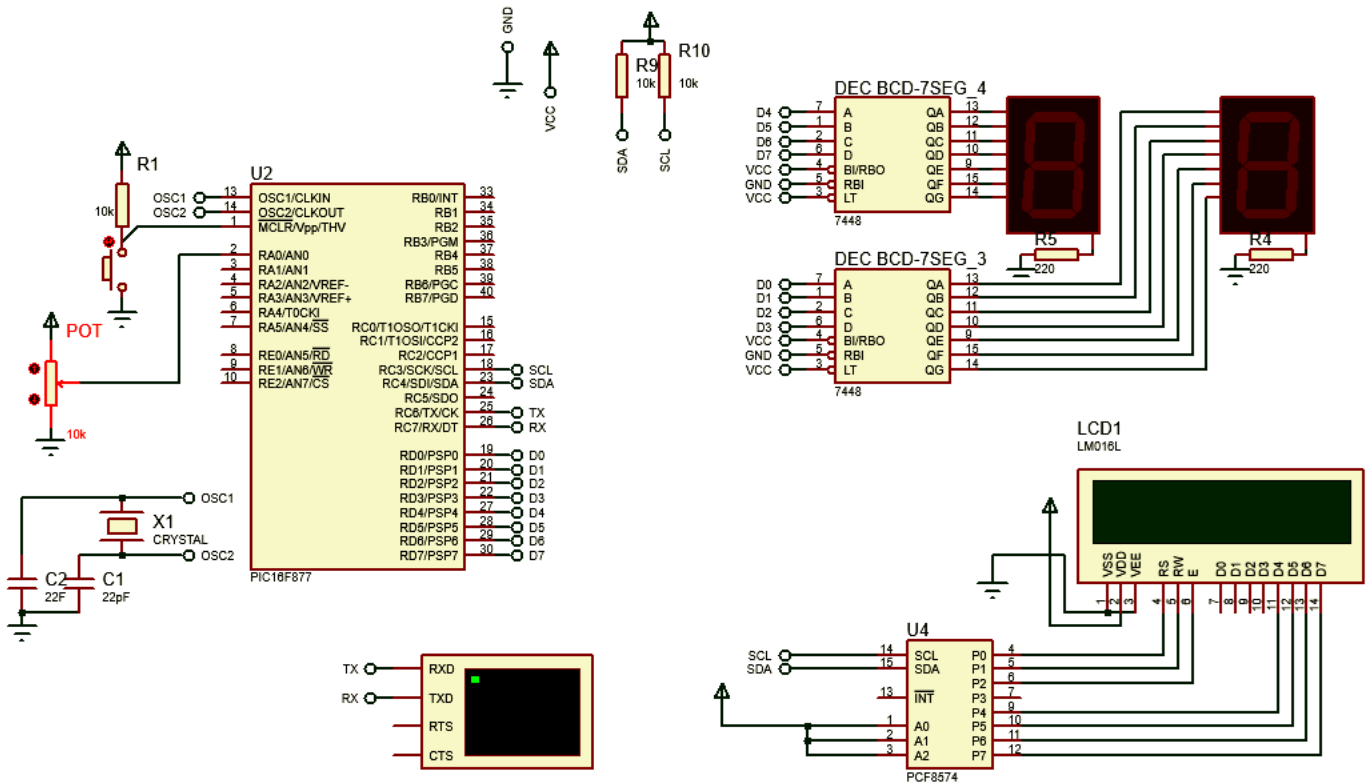
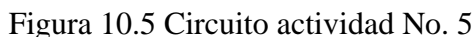


Figura 10.4 Circuito actividad 4

Tabla 10.2 Acciones actividad 5



---

**Laboratorio de Microcomputadoras**  
**Práctica No. 11**  
**Plataforma Arduino**

---

Objetivo: El alumno aprenderá a utilizar las características de la plataforma Arduino, realizará la programación haciendo uso de diversos recursos

### Introducción

Arduino es una plataforma de código abierto basado en software y hardware fácil de usar, permite percibir entradas y generar salidas a través de sus terminales tanto en formato digital como analógico.

Existen una gran variedad de versiones de Arduino, con características propias asociadas al microcontrolador usado en la plataforma, todas ellas emplean el mismo IDE, el software, funciones y métodos ; algunas de las versiones más empleadas son:

Arduino Nano  
Arduino Micro

Arduino Esplora  
Arduino Mega

Arduino Leonardo  
Arduino UNO

La práctica utilizará la plataforma Arduino UNO, que emplea al microcontrolador ATMEGA328; contiene las siguientes terminales, algunas de ellas con función alterna, su uso es de manera independiente.

- 14 Terminales digitales de Entrada o Salida (0, 1, 2, ..., 13)
- 6 Terminales para señales analógicas: A0, A1, A2, A3, A4, A5; con opción de ser usadas con digital
- 6 Terminales con función PWM (salidas analógicas)
- Terminales Tx y Rx
- Terminales I2C SDA y SCL
- 32 K de memoria Flash
- 0.5 K de bootloader Arduino
- 2 K de memoria SRAM
- 1 K de EEPROM

La plataforma y distribución de la plataforma se muestra en las gráficas siguientes:



Figura 11.1 Plataforma Arduino UNO



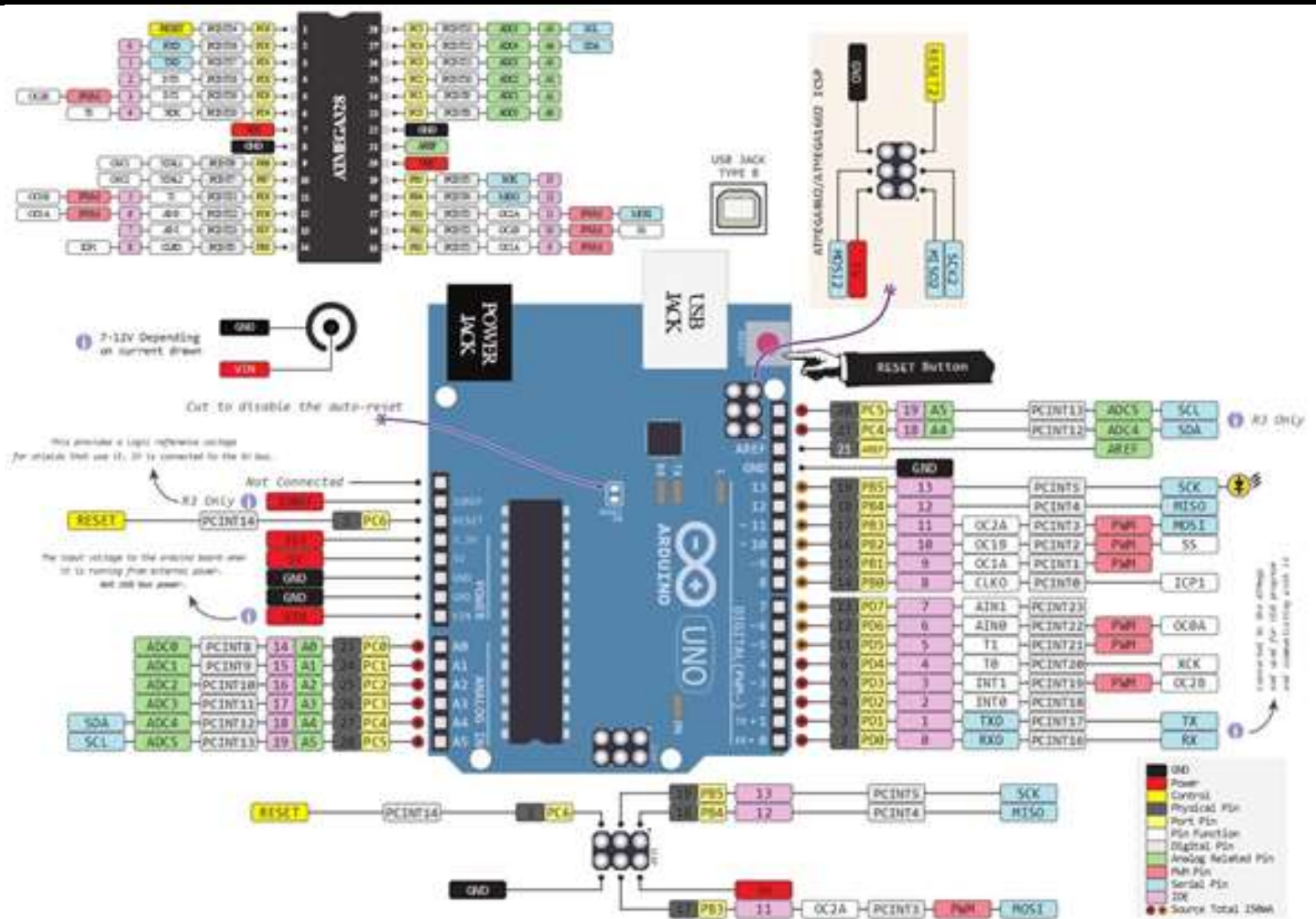


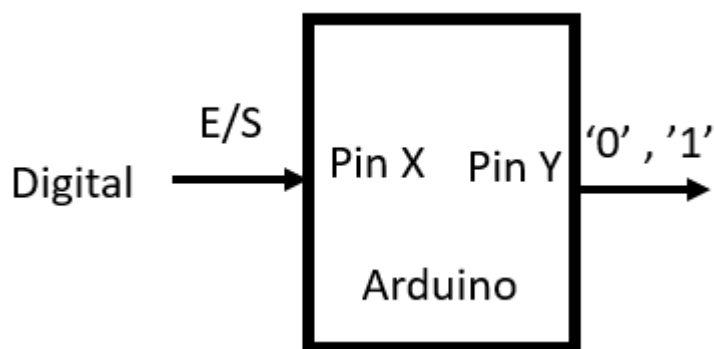
Figura 11.2 Microcontrolador ATMEGA328 en Arduino UNO

La programación se realiza empleando el IDE conocido como Arduino, el cuál puede ser descargado de manera libre del sitio oficial.

La ventaja de emplear Arduino radica en la gran cantidad de funciones, métodos, bibliotecas y ejemplos de aplicación disponibles por los desarrolladores así como por la comunidad que aporta a nivel mundial.

A continuación se presenta la información relacionado con los periféricos empleados con anterioridad, así como las librerías de algunos componentes más usados; se deja al alumno la iniciativa de consultar la ayuda de Arduino para obtener detalles o mayor información.

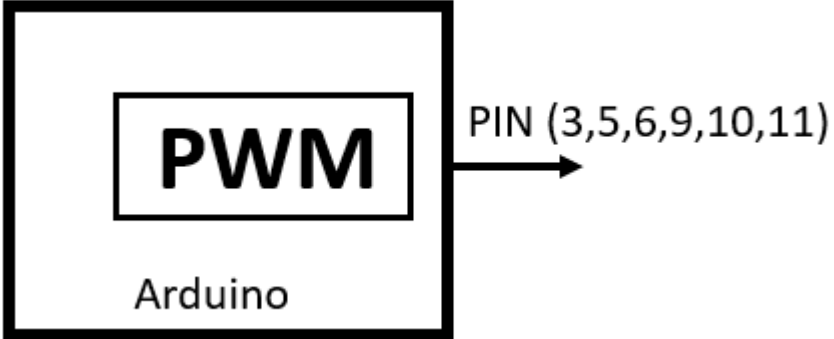
Entradas y salidas Digitales	Descripción
pinMode(pin, función);	Configura el PIN como entrada o salida:  PIN= 1,2,...13; puede también usar 14 a 19 Función: <ul style="list-style-type: none"> <li>• INPUT; entrada</li> <li>• OUTPUT; salida</li> </ul>
digitalWrite(pin,valor)	Escribe un nivel lógico en el pin indicado: Valor: <ul style="list-style-type: none"> <li>• HIGH; Nivel alto ('1')</li> <li>• LOW; nivel bajo ('0')</li> </ul>
digitalRead(pin)	Obtiene el estado lógico del pin; regresa: <ul style="list-style-type: none"> <li>• TRUE; cuando sea '1'</li> <li>• FALSE; cuando sea '0'</li> </ul>



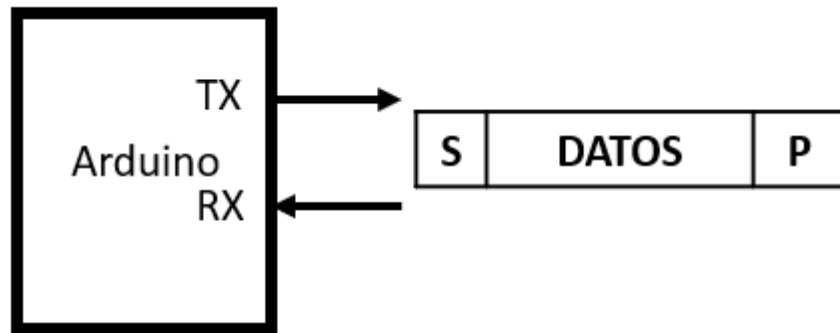
Entradas Analógicas	Descripción
analogRead(pin);	Lectura de la conversión de la entrada indicada.

```

graph LR
    Vi[Vi] --> AX[AX]
    subgraph Arduino
        subgraph Conversion
            AX
        end
    end
  
```

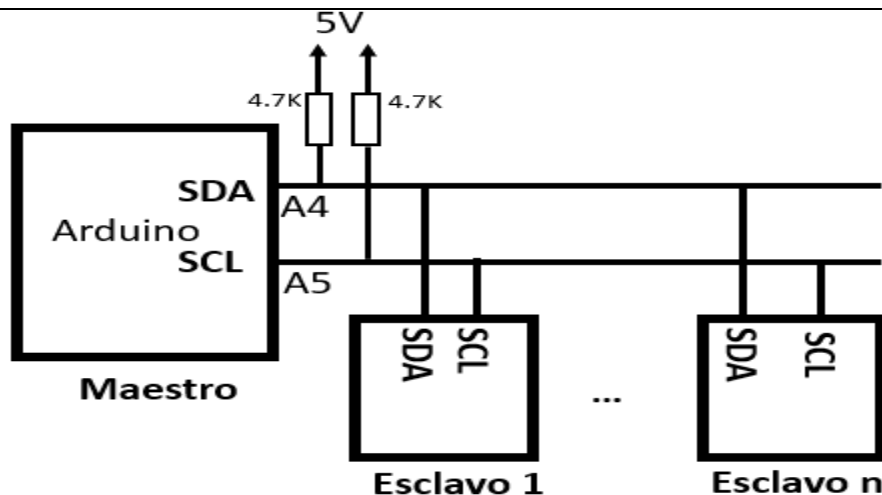
Salidas Analógicas	Descripción
analogWrite(pin,valor)	<p>Genera una señal PWM a través del pin indicado de 495Hz; valor controla el ciclo de trabajo de la señal.</p> <ul style="list-style-type: none"> <li>• Pin=3,5,6,9,10 y 11 para Arduino UNO</li> <li>• Valor = [0 .... 255]</li> </ul>
	

Comunicación Serie Asíncrona	Descripción
Serial.begin(BAUD);	<p>Configura la comunicación asíncrona, con protocolo 8,1,1:</p> <ul style="list-style-type: none"> <li>• BAUD = Tasa de transferencia; la tasa estándar es 9600 Bauds</li> </ul>
Serial.print(dato);	<p>Transfiere por el puerto serie asíncrono información a otro dispositivo; dato puede ser:</p> <ul style="list-style-type: none"> <li>• Carácter</li> <li>• Cadena de caracteres</li> <li>• Variable</li> </ul> <p>Se pueden usar los formatos estándares de despliegue; en caso de enviar información al equipo de cómputo en uso, el IDE de Arduino cuenta con un monitor.</p>
Serial.println(dato);	<p>Similar a la función anterior, pero incluye salto de línea</p>
Serial.available();	<p>Indica si ha llegado un dato</p>
Serial.read();	<p>Lectura del dato que ha llegado</p>



Comunicación Serie Síncrona I2C	Descripción
<code>#include &lt;Wire.h&gt;</code>	Librería para uso y control de I2C; debe ser incluida para usar las funciones estándar de comunicación I2C
<code>Wire.begin();</code>	Inicia protocolo I2C
<code>Wire.beginTransmission(Address);</code>	Inicia la comunicación I2C <ul style="list-style-type: none"> <li>• Address: Dirección del esclavo</li> </ul>
<code>Wire.endTransmission();</code>	Finaliza la comunicación I2C
<code>Wire.requestFrom(address,nbytes);</code>	Solicita información de <b>n</b> datos (bytes) al esclavo
<code>Wire.available();</code>	Indica si existen datos pendientes para leer
<code>Wire.write();</code>	Escribe (envía) un dato al esclavo
<code>Wire.read();</code>	Lectura del dato enviado por el esclavo
<code>Wire.onReceive(handler);</code>	Registra una función callback al recibir un dato
<code>Wire.onRequest(handler);</code>	Registra una función callback al solicitar un dato

Existen una gran variedad de dispositivos I2C, en los cuales los fabricantes han programado bibliotecas para usar de manera transparente, por lo que facilita el uso de estos módulos.



Algunas de las bibliotecas de componentes.

<b>Display de Cristal Líquido LCD</b>	
#include <LiquidCrystal.h>	Biblioteca para uso LCD; usa protocolo de 4 bits de datos
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);	Asignación de conexión de LCD a Arduino
lcd.begin(Columnas, Renglones);	Define el tamaño del LCD <ul style="list-style-type: none"> <li>• Columnas</li> <li>• Renglones</li> </ul>
lcd.print(dato);	Imprime el dato en LCD; donde dato: <ul style="list-style-type: none"> <li>• variable</li> <li>• cadena de caracteres</li> </ul>
lcd.setCursor(, Coluna,Renglón);	Indica la siguiente posición a inicial el despliegue <ul style="list-style-type: none"> <li>• Columna: 0, 1,2, .... n</li> <li>• Renglón: 0, 1, 2, .... N</li> </ul> Por ejemplo: <ul style="list-style-type: none"> <li>• lcd.setCursor(0,0); //primer columna, primer columna</li> </ul>

<b>Servomotor</b>	
#include <servo.h>	Librería para uso del servomotor
Servo myservo;	Creación del objeto para controlar el servo
myservo.attach(pin);	Asigna el pin de Arduino en el que se conectará el servo
myservo.write(val);	Posiciona el servo <ul style="list-style-type: none"> <li>• val: 0-180</li> </ul>

Motor a pasos	
#include <Stepper.h>	Biblioteca motor a pasos
Stepper(ppr, pin1, pin2, pin3, pin4);	Crea una nueva instancia de la clase stepper <ul style="list-style-type: none"> <li>• ppr: pasos definidos por revolución del motor</li> <li>• pin1:pin4: terminales asignadas al motor de pasos a Arduino</li> </ul>
setSpeed(rpm);	Define la velocidad en revoluciones por minuto
step(pasos);	Cantidad de pasos a ejecutar <ul style="list-style-type: none"> <li>• Si <i>pasos</i> es positivo, el motor gira en un sentido.</li> <li>• Si <i>paso</i> es negativo, el motor gira en sentido contrario</li> </ul>

Display de cristal Líquido LCD I2C	
#include <Wire.h>	Biblioteca para uso del protocolo I2C
#include <LiquidCrystal_I2C.h>	Biblioteca para uso del LCD en formato I2C
lcd.init();	Iniciliza al LCD I2C
lcd.setCursor(columna,renglón);	<ul style="list-style-type: none"> <li>• Indica la siguiente posición a desplegar</li> </ul>
lcd.print(dato);	Imprime el dato en el LCD I2C <ul style="list-style-type: none"> <li>• Dato puede ser una variable o una cadena de caracteres, sigue el formato de despliegue convencional de printf</li> </ul>

Funciones de tiempo	
delay(int_mili);	Genera un retardo en milisegundos <ul style="list-style-type: none"> <li>• <i>int_mili</i>: es un entero que indica la cantidad de milisegundos a pausar</li> </ul>
delay_Microsecons(int_micro);	Genera retardo <ul style="list-style-type: none"> <li>• <i>int_micro</i>: es un número entero que indica la cantidad de microsegundos a pausar</li> </ul>
Micros();	Retorna la cantidad de <i>microsegundos</i> en la ejecución del programa
Millis()	Retorna la cantidad de <i>milisegundos</i> en la ejecución del programa

## **Programación de Arduino**

Los programas de Arduino se definen como sketch, los cuales deben de contener al menos dos funciones:

- a. **setup() {}** // En esta función se definen las configuraciones de los recursos usados dentro del sketch, o aquellas sentencias que serán ejecutadas por única ocasión dentro del sketch.
- b. **loop(){}**  // Contiene las sentencias del programa principal.

## **Sketch de Arduino**

```
#include<biblioteca> // Biblioteca(s) a incluir

int i;                // definición de variable(s) globales

void setup()

{
    /* Configuraciones */
    sentencias;
}

void loop()

{
    /* Programa principal */
    sentencias;
}
```

# IDE Arduino

El IDE presenta comandos equivalentes a otros IDE's vistos previamente; figura 11.3.

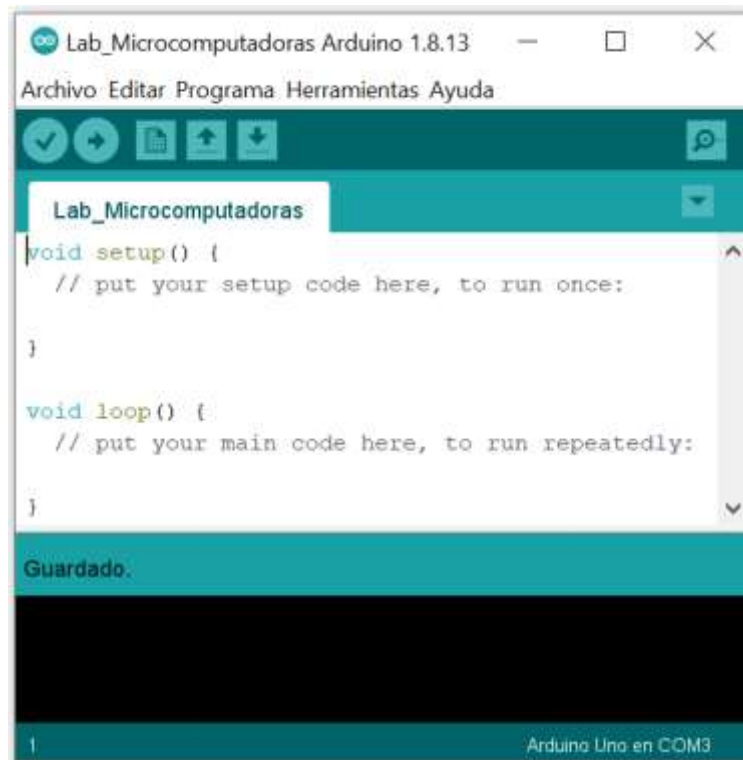


Figura 11.3 IDE Arduino

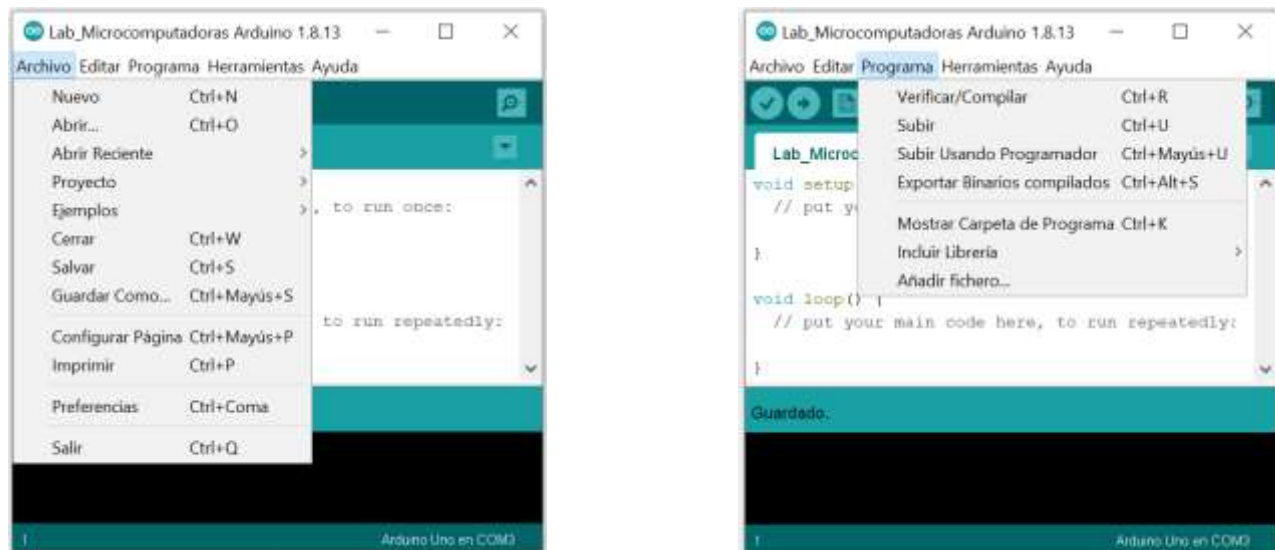
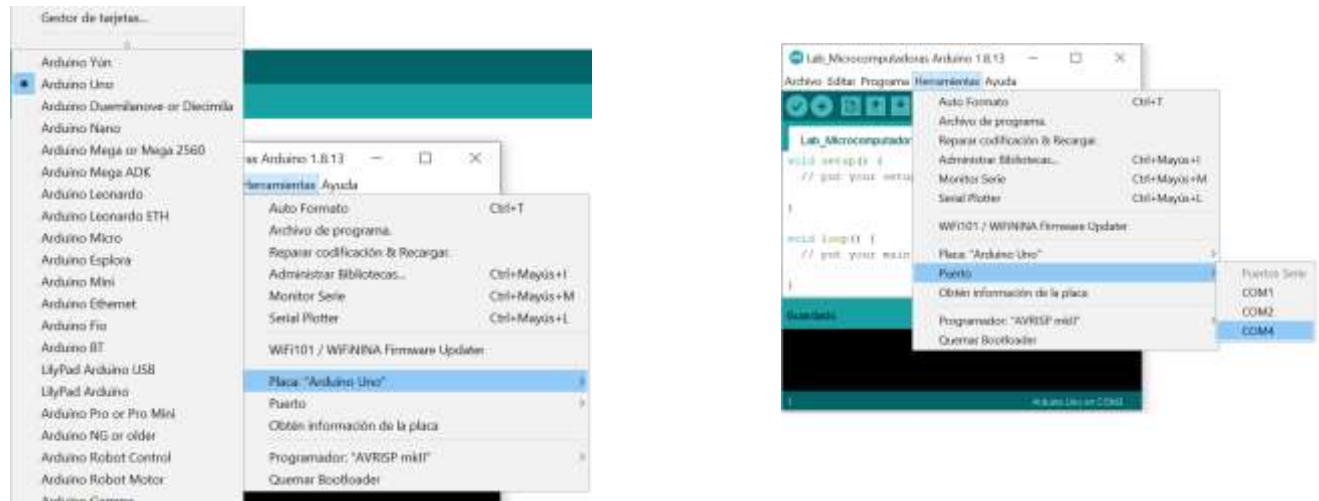


Figura 11.4 Comandos para manejo de archivos y programa





a. Selección de la plataforma

b. Configuración del puerto COM

Figura 11.5 Comandos herramientas

La figura 11.6 muestra los iconos de ejecución rápida.

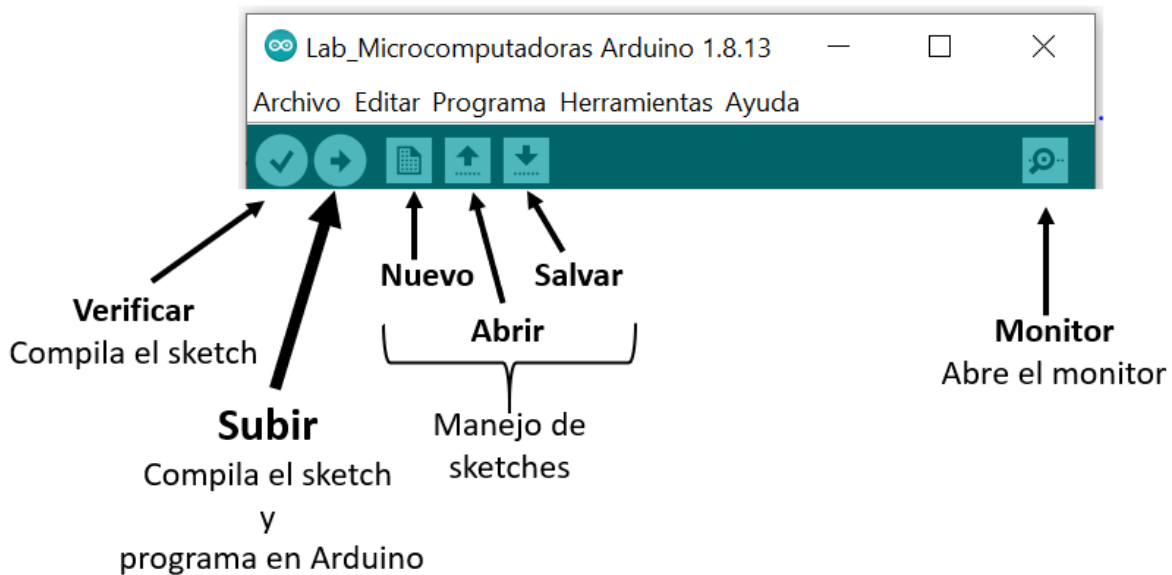


Figura 11.6 Comandos por medio de iconos

## Simulación de un sketch en Arduino

Para simular programas (sketches) de Arduino en Proteus, se deben realizar los siguientes procedimientos:

1. Descargar la librería de Arduino
2. Descomprimirla, copiar y pegar, de acuerdo a la versión instalada, podrá ubicarlo en alguna de las siguientes rutas:
  1. C:\ProgramFiles(x86)\LabcenterElectronics\Proteus8Professional\LIBRARY
  2. C:\ProgramFiles(x86)\LabcenterElectronics\Proteus8Professional\DATA\LIBRARY
  3. En caso de no ubicar el directorio DATA, entonces desde el directorio raíz habilitar los archivos ocultos para seguir la ruta siguiente:  
C:\ProgramData\Labcenter\Proteus8Profesional\LIBRARY
3. En el IDE de Arduino, escribir el sketch; el archivo que se requiere en Proteus es el HEX, por lo que es necesario configurar en Archivo -> Preferencias -> habilitar opción COMPILACIÓN (figura 11.7).
4. Compilar el sketch y en la zona de información, regularmente en la parte inferior, ubicar la ruta del programa HEX copiar la ruta de este sin incluir las comillas, tal como se muestra en la figura 11.8.
5. Crear el proyecto, una vez terminado seleccionar la placa Arduino para cargar el archivo; comenzar la simulación.

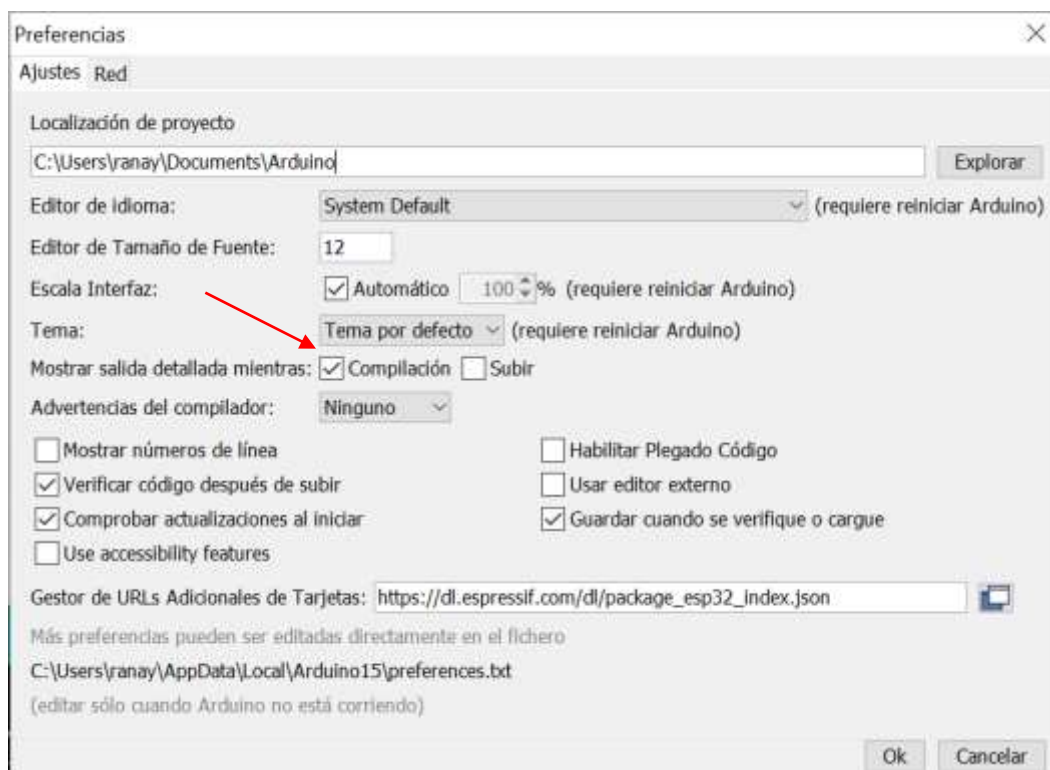


Figura 11.7 Configuración que mostrará la ruta del archivo HEX

```

ata\\Local\\Temp\\arduino_build_342306/Blink.ino.elf" "C:\\Users\\ranay\\AppData\\Local\\Temp\\arduino_build_342306/Blink.ino.elf"
on-lma .eeprom=0 "C:\\Users\\ranay\\AppData\\Local\\Temp\\arduino_build_342306/Blink.ino.elf" "C:\\Users\\ranay\\AppData\\Local\\Temp\\arduino_build_342306/Blink.ino.hex"

```

Figura 11.8 Ruta de ubicación del archivo HEX

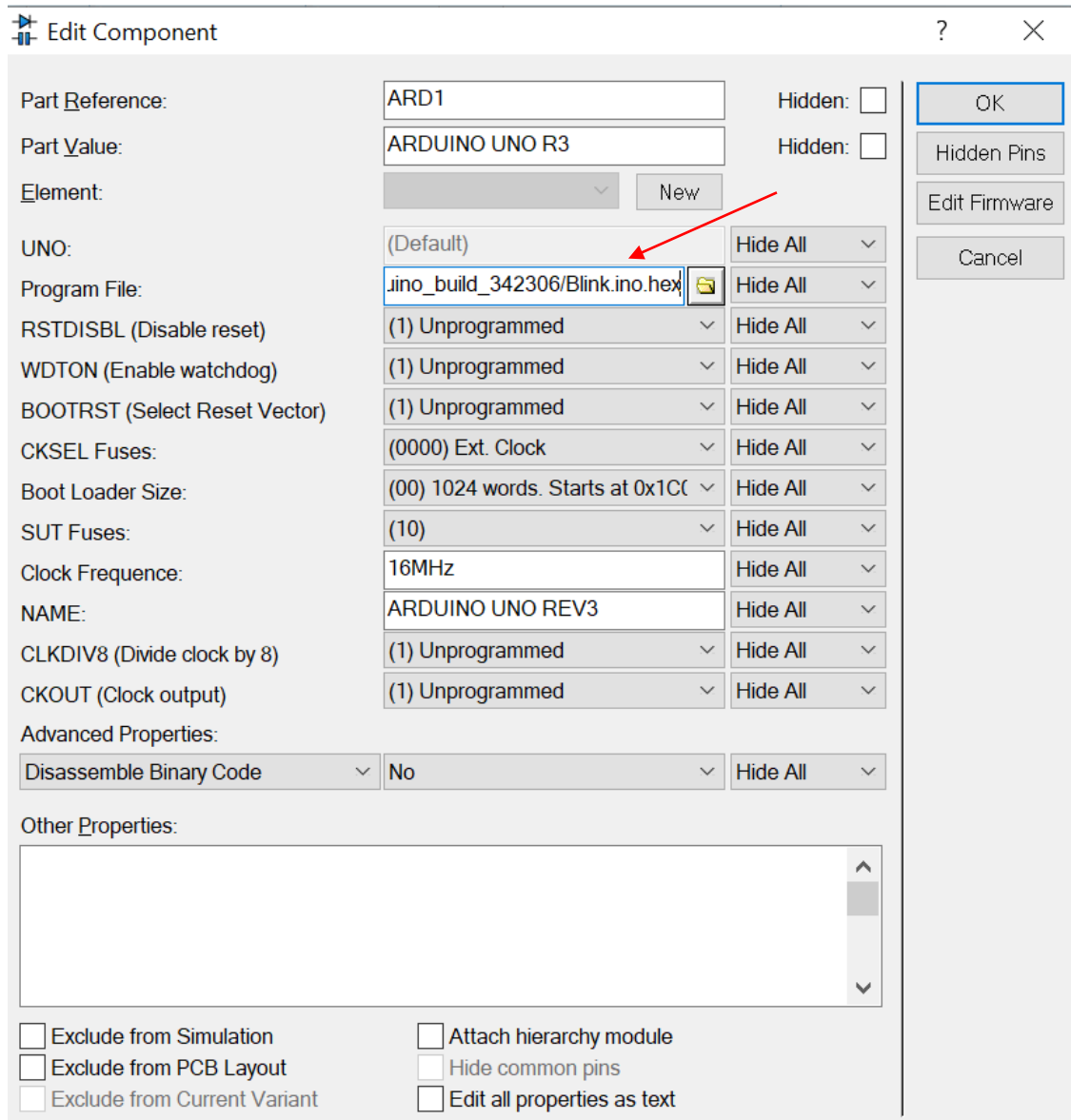


Figura 11.9 Selección archivo HEX en Proteus

## Desarrollo

1.- Ubicar y ejecutar los programas de ejemplo que contiene Arduino, conectar los componentes necesarios para comprobar el funcionamiento. Explicar en su reporte el funcionamiento de estos

- a. Blink
- b. Button
- c. AnalogInput
- d. AnalogInOutSerial
- e. Fading
- f. Servo Knob
- g. Servo Sweep
- h. Stepper\_one\_revolution
- i. HelloWorld

2.- Realizar un programa, en el que genere al menos seis diferentes efectos con ocho salidas digitales de Arduino, conectar leds y resistencias; la asignación de terminales queda a elección del alumno así como las animaciones.

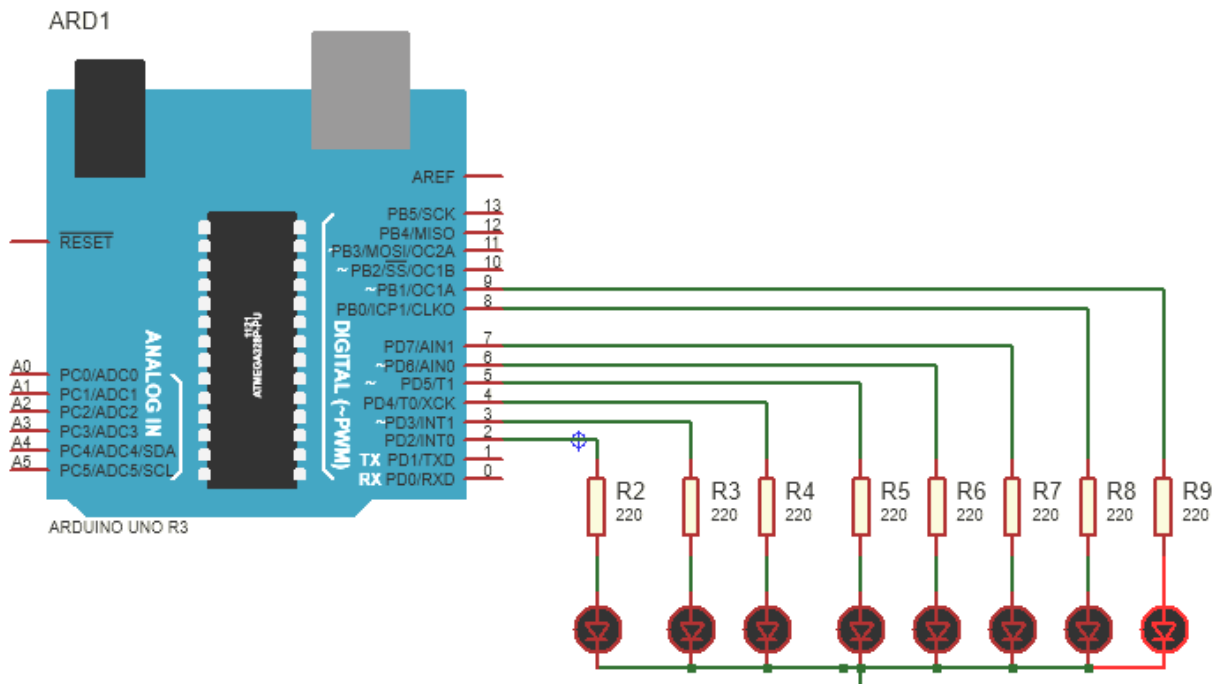


Figura 11.7 Esquemático actividad 2

3.- Escribir el siguiente sketch, comentar, alambrear y ejecutar.

```

/*                      I2C en Arduino UNO                      */
/*                      A4 (SDA)                                */
/*                      A5 (SCL)                                */
/*  Usa PCF8574 con configuración de dirección 0x27, display 16x2 */

#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);

void setup() {
    lcd.init();
    lcd.clear();
    lcd.setCursor(3,0);
    lcd.print(" U N A M ");
    lcd.setCursor(0,1);
    lcd.print("* Ingenieria *");
}

void loop()
{
}

```

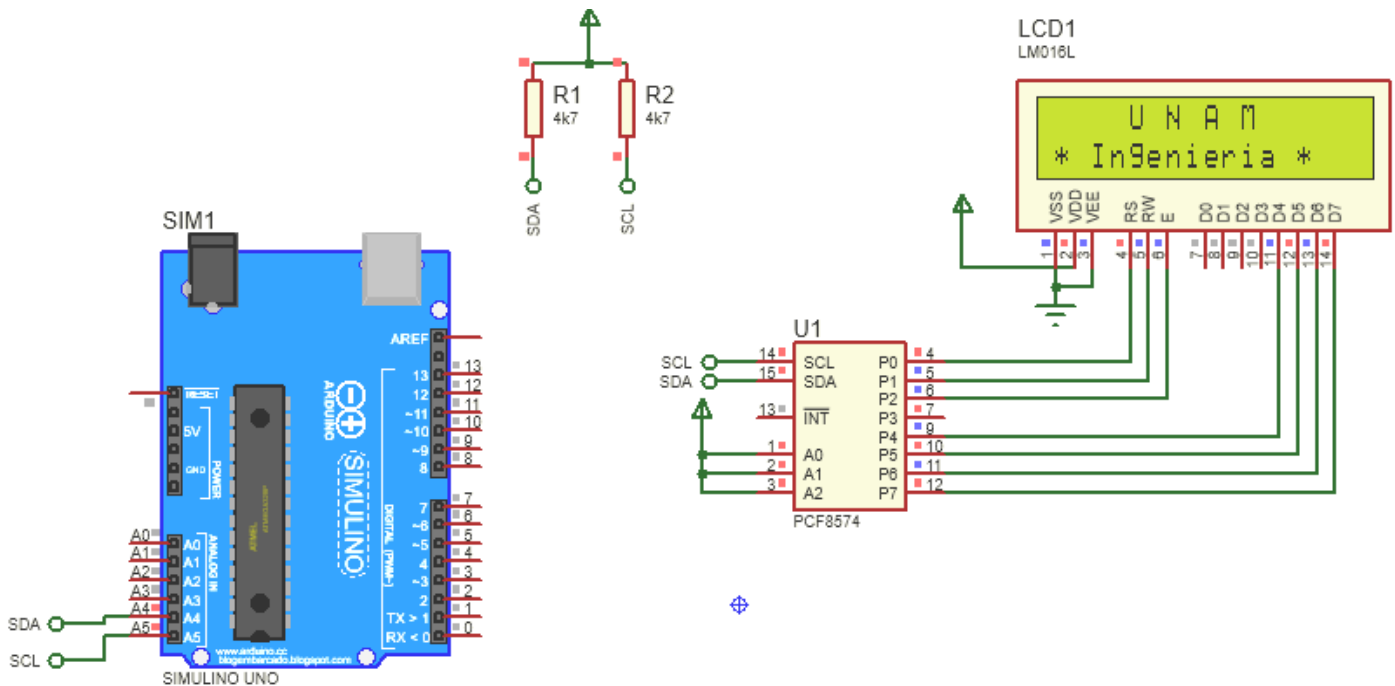


Figura 11.8 Circuito actividad 3

4.- Realizar un sketch, que muestre el voltaje de un canal analógico de Arduino en la primer línea un LCD con interfaz I2C, esta señal es usada para controlar el PWM que controla un motor de corriente directa; el ciclo de trabajo será mostrado en la segunda fila del LCD; de igual manera el maestro desplegará el resultado de la conversión en varios formatos: hexadecimal, decimal y voltaje.

Consideraciones:

- Realizar las conexiones necesarias
- La función que controla el PWM puede oscilar entre 0 a 255
- El resultado de la conversión es de 10 bits

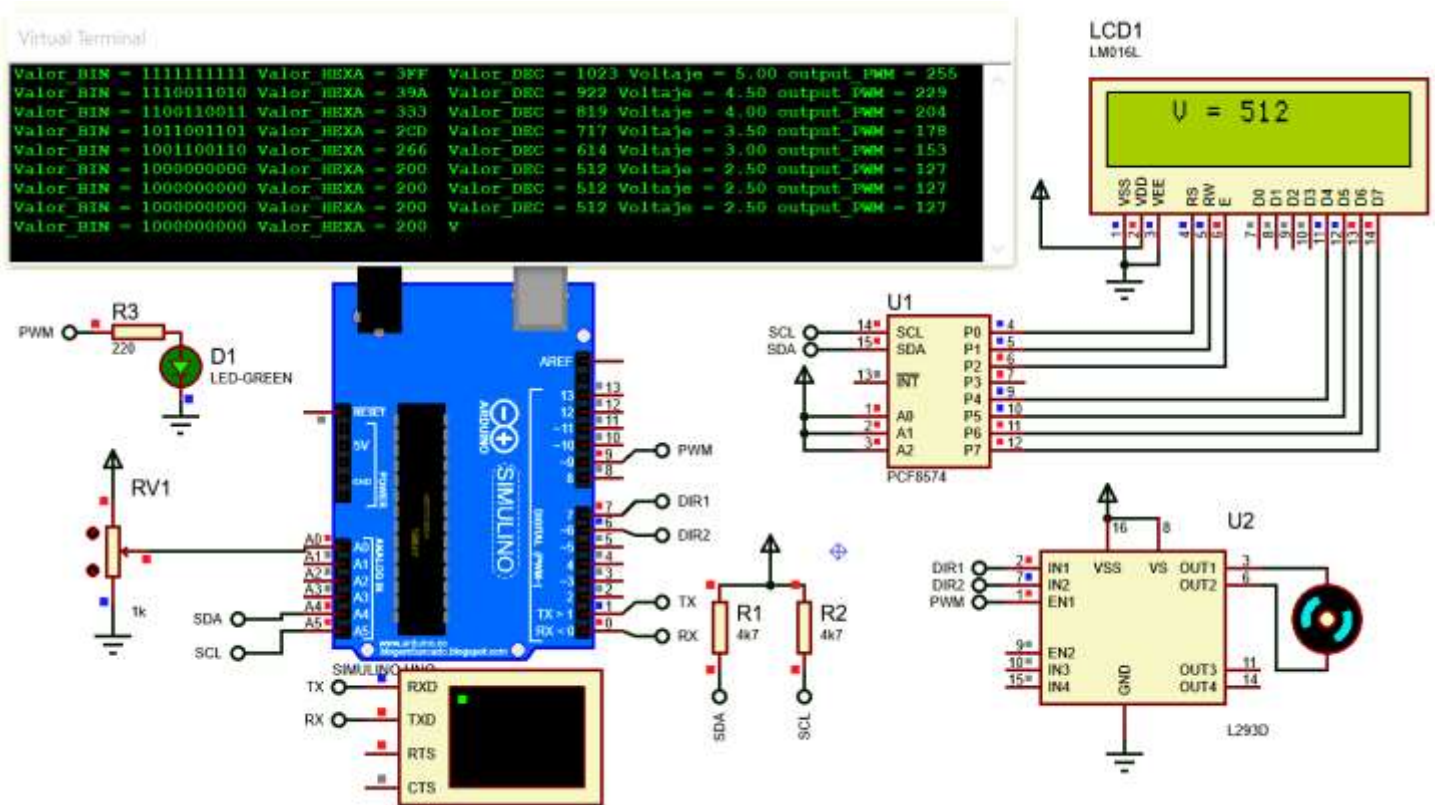


Figura 11.9 Circuito actividad 4

## Laboratorio de Microcomputadoras

## Práctica No. 12

## Plataforma Raspberry Pi

**Objetivo.** Aprenderá las características de la plataforma Raspberry, se introducirá en la programación, revisará los recursos que dispone, tanto de software como de hardware, realizará programas que permitan la interconexión con una gran variedad de dispositivos, periféricos y con otras plataformas de microcomputadoras.

## Introducción

Raspberry es una microcomputadora de bajo costo y tamaño reducido, debe ser instalado el sistema operativo (Rasbian o cualquier otro) en una memoria micro SD, conectarle teclado, mouse, un monitor o pantalla vía HDMI y la fuente de alimentación.

Similar a las plataformas estudiadas previamente; Raspberry tiene diferentes modelos y versiones, cada una de ellas con características específicas. De manera constante salen al mercado nuevas versiones, describiremos generalidades destacando las versiones Raspberry 3B+ y Raspberry 4.

Modelo	SOC	Frecuencia MHZ	RAM	Puertos USB	Ethernet	Wireless Bluetooth
Raspberry Pi						
Raspberry Pi A+	BCM2835	700	512 MB	1	NO	NO
Raspberry Pi B+	BCM2835	700	512 MB	4	SI	NO
Raspberry Pi 2 B	BCM2836	900	1 GB	4	SI	NO
Raspberry Pi 3B	BCM2837	1200	1 GB	4	SI	SI
Raspberry Pi 3B+	BCM2837B0	1500	1 GB	4	SI	SI
Raspberry Pi Zero	BCM2835	1000	512	1	NO	NO
Raspberry Pi Zero W	BCM2835	1000	512	1	NO	SI
Raspberry Pi 4B	BCM2711	1500	1,2,4 u 8	4	SI	SI
Raspberry Pi Pico						

Características adicionales:

## a. Raspberry Pi 3B+

- CPU + GPU: broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4 GHz
- RAM: 1GB LPDDR2 SDRAM
- Wi-Fi + Bluetooth: 2.4 Ghz y 5 GHz IEEE 802.11.b/g/n/ac, Bluetooth 4.4, BLE
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO de 40 pines
- HDMI
- 4 puertos USB 2.0
- Puerto CSI para conectar cámara
- Puerto DSI para conectar una pantalla táctil
- Salida de audio y video compuesto
- Micro SD
- Power-over-Ethernet (PoE)

## **b. Raspberry 4B**

- Broadcom BCM2711, Cortex núcleo cuádruple-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- SDRAM LPDDR4-2400 de 1 GB, 2 GB, 4 GB y 8 GB (según el modelo)
- 2,4 GHz y 5,0 GHz IEEE 802.11ac inalámbrico, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 puertos USB 3.0; 2 puertos USB 2.0.
- Cabezal GPIO estándar de 40 pines de Raspberry Pi (totalmente compatible con las placas anteriores)
- 2 × puertos micro-HDMI (soportados hasta 4kp60)
- Puerto de pantalla MIPI DSI de 2 vías
- Puerto de cámara MIPI CSI de 2 carriles
- Puerto de audio estéreo de 4 polos y de vídeo compuesto
- H.265 (decodificación 4kp60), H264 (decodificación 1080p60, decodificación 1080p30)
- Gráficos OpenGL ES 3.0
- Ranura para tarjetas Micro-SD para cargar el sistema operativo y el almacenamiento de datos
- 5V DC a través de conector USB-C (mínimo 3A\*)
- 5V DC vía cabezal GPIO (mínimo 3A\*)
- Alimentación a través de Ethernet (PoE) habilitada (requiere PoE HAT separado)
- Temperatura de funcionamiento: 0 – 50 grados C ambiente

El sistema operativo debe ser instalado en la memoria micros SD, descargar de manera libre de la pagina oficial de Raspberry Pi: <https://www.raspberrypi.org/>; en el mismo sitio podrá encontrar mayor detalle para la instalación e información adicional.

Procedimiento de instalación del sistema operativo:

1. Formatear la memoria SD
2. Descargar el SO Rasbian o el deseado
3. Descargar Balena Etcher u el software preferido para montar el SO a la memoria; consultar la guía de montaje
4. Una vez instalado, realizar las conexiones de los periféricos en la Raspberry.



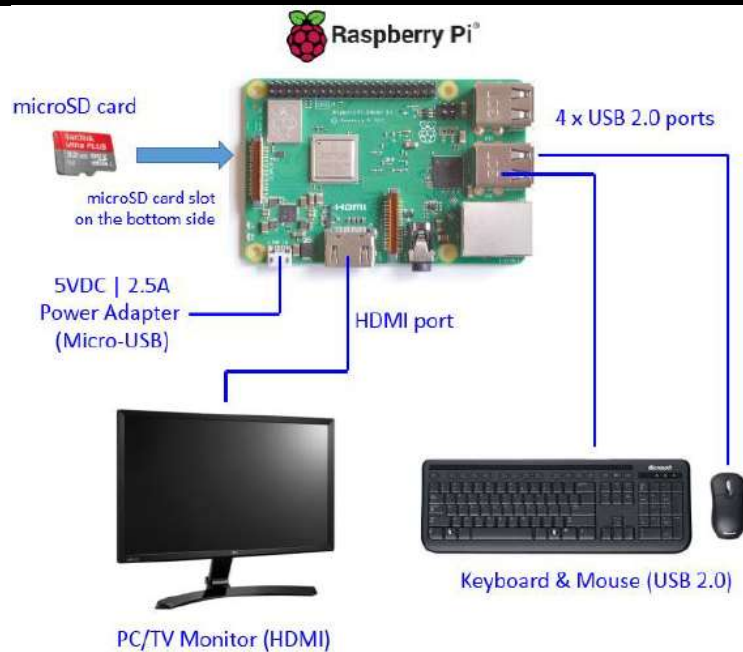


Figura 12.1 Conexión de Raspberry Pi

## Programación

La Raspberry es un ordenador, el cual se puede usar como tal, ya sea para enviar correos, administrar archivos o ver videos en internet.

Sin embargo, las capacidades de la Raspberry van más allá. Es decir, al usar Raspbian como sistema operativo, permite la libertad de usar todas las funcionalidades de un sistema Linux. Además de los diferentes periféricos integrados para interactuar.

Algunos ejemplos se mencionan son:

- Puertos USB
- Puerto Ethernet
- Puerto para Cámara
- Puerto para Display

La programación se realizará mediante Python.

## GPIO (General Purpose Input Output)

Una de las características más poderosas de la Raspberry se encuentra en la fila de los GPIO (General-Purpose Input/Output). Estos pines se pueden designar como entradas o salidas de propósito general y tener la posibilidad de añadir componentes externos de Hardware a la tarjeta. Dichos pines se pueden controlar mediante programación y para el caso de la Raspberry se usa Python como lenguaje de programación.

Para la Raspberry 3B+ se cuentan con 40 GPIO's. Sin embargo, no todos se pueden designar como entradas o salidas mediante software, ya que cumplen con funciones específicas.

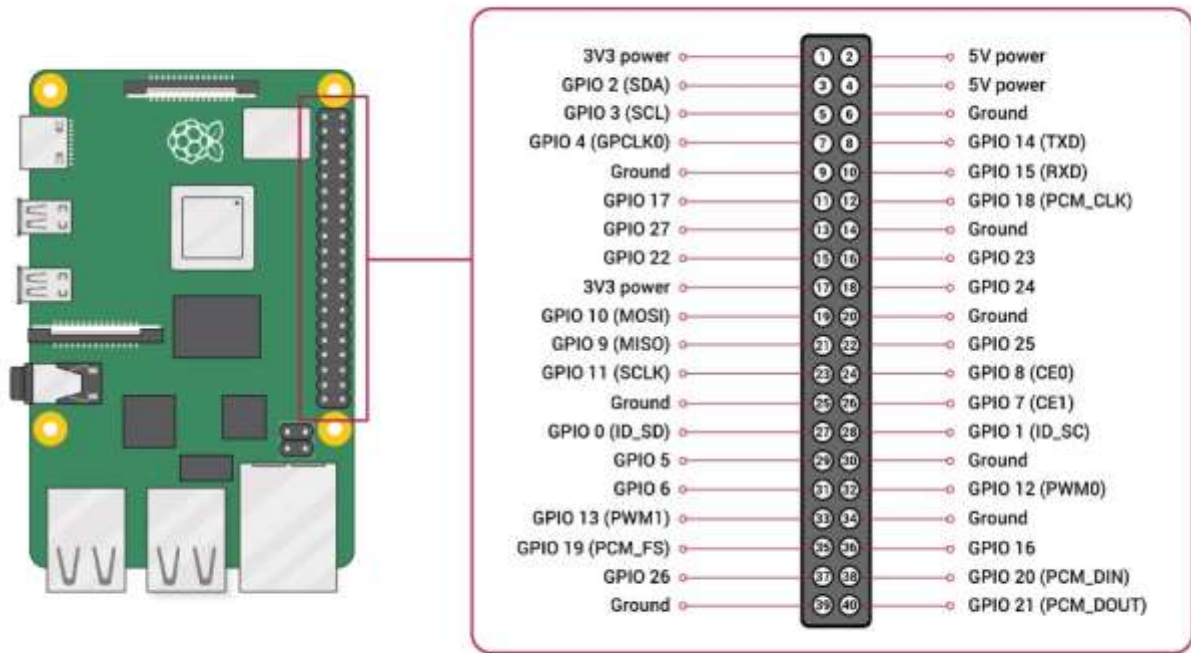


Figura 12.4 GPIO Raspberry Pi

**Características de GPIO**

**Voltajes**

Dentro de los GPIO existen dos pines que entregan 5 volts (Pines 2 y 4) de salida y dos pines de 3.3 volts (Pines 1 y 17) de salida.

También cuenta con 8 pines para tierra (Pines 6, 9, 14, 20, 25, 30, 34, 39).

**Outputs**

Un GPIO designado como salida puede ser puesto en un nivel lógico alto (3.3 V) o un nivel lógico bajo (0 V).

**Inputs**

Un GPIO designado como entrada puede ser leído en un nivel alto (3.3 V) o un nivel lógico bajo (0 V).

---

### GPIO's especiales

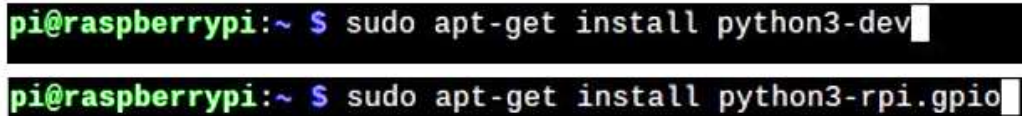
---

- PWM (Pulse-Width-Modulation)
  - ✦ Software de PWM disponible en todos los pines.
  - ✦ Hardware de PWM disponibles en GPIO12, GPIO13, GPIO18, GPIO19.
- SPI (Serial-Protocol-Interface)
  - ✦ SPI0: MOSI (GPIO10); MISO (GPIO9) - SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7).
  - ✦ SPI1: MOSI (GPIO20); MISO (GPIO19) - SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16).
- I2C
  - ✦ Data: (GPIO2); Clock (GPIO3).
  - ✦ EEPROM Data: (GPIO0); EEPROM Clock (GPIO1).
- Serial (UART)
  - ✦ TX (GPIO14); RX (GPIO15).

### Uso de GPIO

A continuación, se presentan ejemplos para manipular los GPIO como entradas o salidas. Se usará Python como lenguaje de programación.

Antes de comenzar es importante mencionar que se debe de instalar la biblioteca correspondiente ("RPi.GPIO") para el uso de los GPIO. Para esto se usa el siguiente comando desde la terminal.



```
pi@raspberrypi:~ $ sudo apt-get install python3-dev
pi@raspberrypi:~ $ sudo apt-get install python3-rpi.gpio
```

Figura 12.5 Instalación de biblioteca GPIO

## Desarrollo

1.- Abrir el editor de Python, escribir el siguiente código, alambrear el circuito y ejecutar.

```
#importar librerías
```

```
import RPi.GPIO as GPIO
import time
```

```
GPIO.setwarnings(False)           # Desactiva warnings
GPIO.setmode(GPIO.BCM)           # Define la asignación de pines (num_pin, I/O)
GPIO.setup(18,GPIO.OUT)           # configura el pin GPIO 18 como salida
```

```
#código principal
```

```
while(True):
    GPIO.output(18,True)           # Pone en alto el GPIO 18
    time.sleep(0.5)                # Genera un retardo de medio segundo
    GPIO.output(18,False)          # Pone en bajo el GPIO 18
    time.sleep(0.5)                # Retardo de medio segundo
```

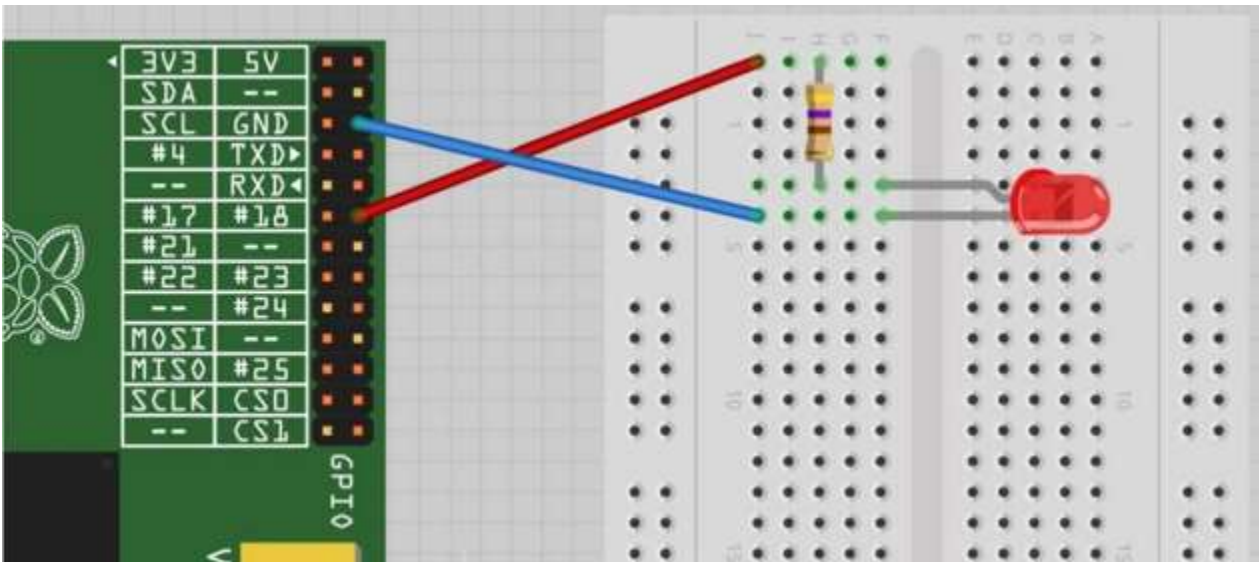


Figura 12.1 Circuito actividad 1

2.- Escribir, comentar y ejecutar el siguiente programa en la Raspberry Pi

```
#importar librerías

import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

switch_pin=18
led_pin=23
led_state=False
old_input_state=True

GPIO.setup(switch_pin, GPIO.IN,pullup_down=GPIO.PUD_UP)
GPIO.setup(led_pin,GPIO.OUT)

#código principal

while(True):
    new_input_state = GPIO.input(switch_pin)
    if new_input_state == False and old_input_state ==True:
        led_state = not led_state
        old_input_state = new_input_state

    GPIO.output(led_pin,led_state)
    time.sleep(0.5)
```

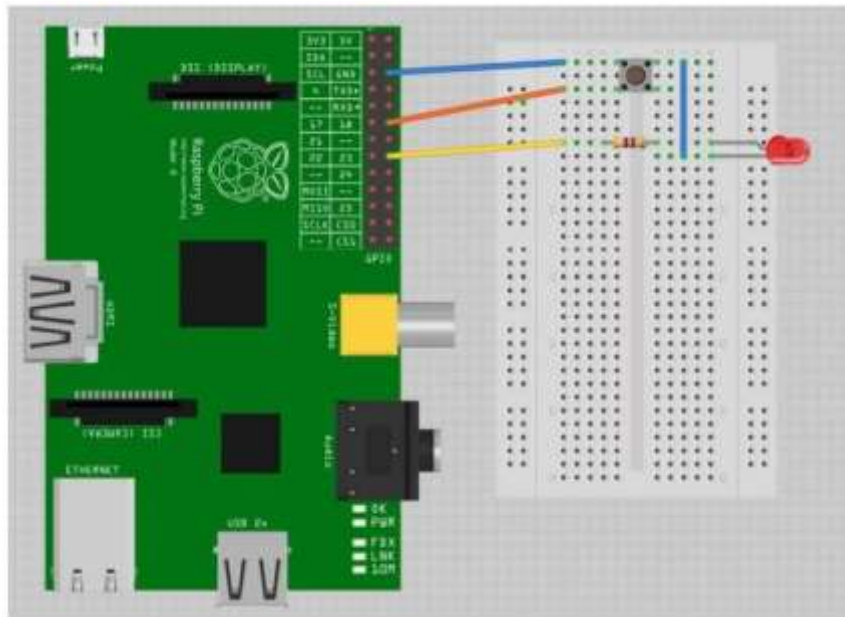


Figura 12.2 Circuito actividad 2

3.- Realizar los programas necesarios para establecer la comunicación serie asíncrona entre Raspberry Pi y Arduino, de acuerdo a:

- La plataforma Raspberry transmite un comando vía USART a Arduino
- De acuerdo al comando recibido, genera diferentes acciones

4.- Realizar los programas necesarios para establecer la comunicación serie síncrona I2C entre Raspberry Pi y Arduino, de acuerdo a:

- La plataforma Raspberry transmite un comando vía I2C a Arduino
- De acuerdo al comando recibido, genera diferentes acciones
- Arduino le responderá a la Raspberry