MICROCONTROLADORES: PIC Y ARDUINO

PUERTOS DE E/S, TEMPORIZADORES, INTERRUPCIONES, CONVERSOR AD, COMUNICACIÓN SERIE

Ing. Jorge Antonio Polanía Puentes

Ingeniero Electrónico. Universidad Distrital - Colombia

Magister en Ingeniería Electrónica. UNAM- México

CONTENIDO

INTRODUCCIÓN

Del Código al Circuito: Programación y Aplicación de Microcontroladores

Los microcontroladores son el cerebro de la electrónica moderna. Desde los dispositivos médicos y los sistemas de automatización industrial hasta los drones, la domótica y los electrodomésticos inteligentes, la capacidad de diseñar, programar y depurar sistemas embebidos es una competencia fundamental para ingenieros, técnicos y creadores de tecnología.

Sin embargo, con frecuencia el aprendizaje de estos dispositivos se fragmenta entre teoría abstracta, entornos de simulación aislados y montaje físico sin guía. Este libro, "Microcontroladores: PIC y Arduino", nace con el propósito de cerrar esa brecha. Basado en más de 30 años de experiencia docente en ingeniería electrónica, el texto integra programación en C, simulación en Proteus y prácticas de laboratorio verificadas para dominar las dos plataformas más utilizadas en la industria y la academia: el PIC 16F877 y el Arduino Uno.

Propósito del Libro

A lo largo de este texto, se busca no solo transmitir conocimiento sobre arquitectura de microcontroladores, sino también desarrollar la capacidad de escribir código eficiente, simular circuitos antes de montarlos e implementar sistemas reales en protoboard. Cada capítulo ha sido estructurado para guiar al lector desde el encendido de un LED hasta el control de motores, adquisición de datos analógicos y comunicación serie, manteniendo un equilibrio entre el rigor técnico y la aplicabilidad inmediata.

¿Qué Encontrará en Este Libro?

El contenido está organizado en ocho capítulos clave que cubren el espectro esencial del desarrollo con microcontroladores:

- Bloque PIC 16F877 + CCS C (Capítulos 1-3): Arquitectura interna, distribución de pines y conjunto de instrucciones. Manejo de puertos de E/S, resistencias pull-up y directivas del compilador CCS. Interrupciones externas, temporizadores (Timer0, Timer1, Timer2) y generación de señales. Simulación completa en Proteus con ejemplos comentados paso a paso.
- Bloque Arduino Uno + IDE (Capítulos 4-6): Estructura de la placa, memoria, pines digitales/analógicos y comunicación. Estructuras de control, salidas PWM, manejo de EEPROM y librería TimerOne. Prácticas de laboratorio: secuenciadores, alarmas, control de brillo y comunicación serie.
- Interfaces y Aplicaciones Avanzadas (Capítulos 7-8): Conversor Analógico-Digital (ADC): voltímetros digitales, termómetros con LM35 y NTC, barómetros. Comunicación serie UART, manejo de strings, control de motores DC con driver L293D y lectura de teclados matriciales.

Enfoque Pedagógico

Cada capítulo incluye:

- Explicaciones técnicas claras sobre registros, modos de operación y configuración de periféricos.
- Código fuente en C (CCS para PIC, Arduino IDE para Uno) con comentarios explicativos.
- Diagramas de hardware listos para simulación en Proteus o montaje en protoboard.
- Ejemplos progresivos que van de lo básico a aplicaciones de control y adquisición de datos.
- Prácticas de laboratorio con listas de materiales, conexiones y objetivos de verificación.

Se ha puesto especial énfasis en la metodología de desarrollo profesional: diseñar → simular → compilar → montar → depurar. Esta cadena de trabajo es la que diferencia a un aficionado de un ingeniero capaz de entregar proyectos funcionales y robustos.

Para Quién es Este Libro

Este texto está dirigido principalmente a:

- Estudiantes de ingeniería electrónica, mecatrónica, automática y de sistemas que cursan asignaturas de microcontroladores o sistemas embebidos.
- Técnicos y tecnólogos en automatización industrial que requieren actualizar sus habilidades en programación de hardware.
- Docentes que buscan material de apoyo con código, simulación y guías de laboratorio integradas.
- Aficionados a la electrónica y la robótica con bases en lógica digital y programación básica que desean dar el salto al control de hardware real.

Requisitos Previos

Para aprovechar al máximo este texto, se recomienda tener conocimientos básicos de:

- Electrónica básica y ley de Ohm.
- Lógica digital y manejo de compuertas.
- Fundamentos de programación en lenguaje C (variables, estructuras de control, funciones).
- Uso básico de multímetro y protoboard.

Conexión con la Serie Técnica

Este libro complementa directamente mis publicaciones anteriores en Amazon KDP:

- "Electrónica Digital: Diseño y Aplicaciones" establece la base lógica y secuencial.
- "Señales y Sistemas Continuos/Discretos" y "Control con MATLAB" introducen el modelado matemático.
- "Microcontroladores: PIC y Arduino" cierra el ciclo con la implementación física de algoritmos de control, adquisición de señales y comunicación, puente ideal hacia "Control de Motores con MATLAB" y "Dispositivos de Control Electrónico".

Nota del Autor

Con base en mi experiencia como Profesor Titular, Decano y Rector en la Universidad Surcolombiana, y la publicación de cursos virtuales y libros técnicos, he recopilado en este volumen los conceptos y prácticas esenciales que todo profesional debe dominar en el área de sistemas embebidos. Mi deseo es que este libro no solo le ayude a aprobar un examen, sino a desarrollar proyectos reales con seguridad, eficiencia y criterio de ingeniería.

Invito al lector a abordar este material con mente abierta y espíritu crítico. No se limite a copiar el código; modifique los tiempos, cambie los sensores, pruebe diferentes frecuencias de reloj y, sobre todo, no dude en montar los circuitos y medir con el osciloscopio. La programación de microcontroladores cobra vida cuando se compila, se simula y se observa su respuesta en el hardware real.

Agradezco de antemano cualquier comentario, sugerencia o corrección que los lectores deseen compartir, con el fin de enriquecer futuras ediciones de este texto.

Ing. Jorge Antonio Polanía Puentes

Neiva, Colombia

2025

CAPÍTULO 1. PUERTOS DE ENTRADA/SALIDA - TEORÍA

1.1 MICROCONTROLADOR PIC 16F877

1. INTRODUCCIÓN

Un microcontrolador es un circuito integrado programable que integra en un solo chip: Las unidades de memoria para el almacenamiento de datos, La unidad aritmética – lógica (ALU) para el cálculo de operaciones, Las unidades de entrada y salida (E/S) para comunicación con otros periféricos, Temporizadores, El Controlador de interrupciones y Otras unidades especiales.

La memoria generalmente está constituida por memoria RAM compuesta por registros que almacena datos temporales, y la memoria EEPROM para el almacenamiento del programa que se debe ejecutar. Cuenta con un registro que se llama Contador de programa que es el encargado de direccionar la instrucción a ejecutar.

La unidad aritmética lógica ALU es la encargada de realizar las operaciones aritméticas suma, resta y multiplicación y las operaciones lógicas como And, Or, Or- exclusivo.

Las unidades de entrada/salida se refieren a los puertos que tiene el micro para recibir o enviar datos en forma serie o en forma paralela. Cuenta además con módulos especiales para convertir señales analógicas a digitales o de digitales a analógicas.

Generalmente tienen arquitectura Harvard que es aquella en donde existen dos buses independientes para mejorar la velocidad de transferencia de información interna: el bus de datos y el bus de direcciones. El bus de datos puede ser de 8, 16, 32 bits y el de dirección depende de la cantidad de memoria del micro.

Los microcontroladores para temporizar sus operaciones de programación tienen internamente un reloj implementado que con solo añadir un cristal y un par de capacitores se genera la frecuencia requerida.

Para inicializar el micro después de conectar la alimentación, existe una señal de Reset que generalmente es activo bajo para limpiar registros internos y colocar bits de control.

Para funcionar el microcontrolador dispone de un conjunto de instrucciones que son traducidas a lenguaje de máquina (1’s y 0’s) por un programa que se llama Ensamblador. Igualmente existen Compiladores que se encargan de traducir un lenguaje de alto nivel como el lenguaje C a lenguaje o código de máquina. En ambos casos es el código ejecutable que se debe grabar en la memoria del micro (EEPROM) para que se ejecute el programa y desarrolle la aplicación que se quiere.

Los parámetros más importantes en un microcontrolador son:

- Bus de datos: 8, 16, 32 bits
- Capacidad de memoria: Tamaño de la memoria RAM y de la memoria EEPROM en kilobytes KB
- Velocidad: Numero de instrucciones a ejecutar por segundo. Depende de la frecuencia del oscilador del micro.
- Puertos: Puertos de entrada salida de forma paralela y serial para comunicación externa.
- Módulos: Para conversión A/D, D/A, PWM, USB, CAN, I2C, SPI, UART, USART, etc

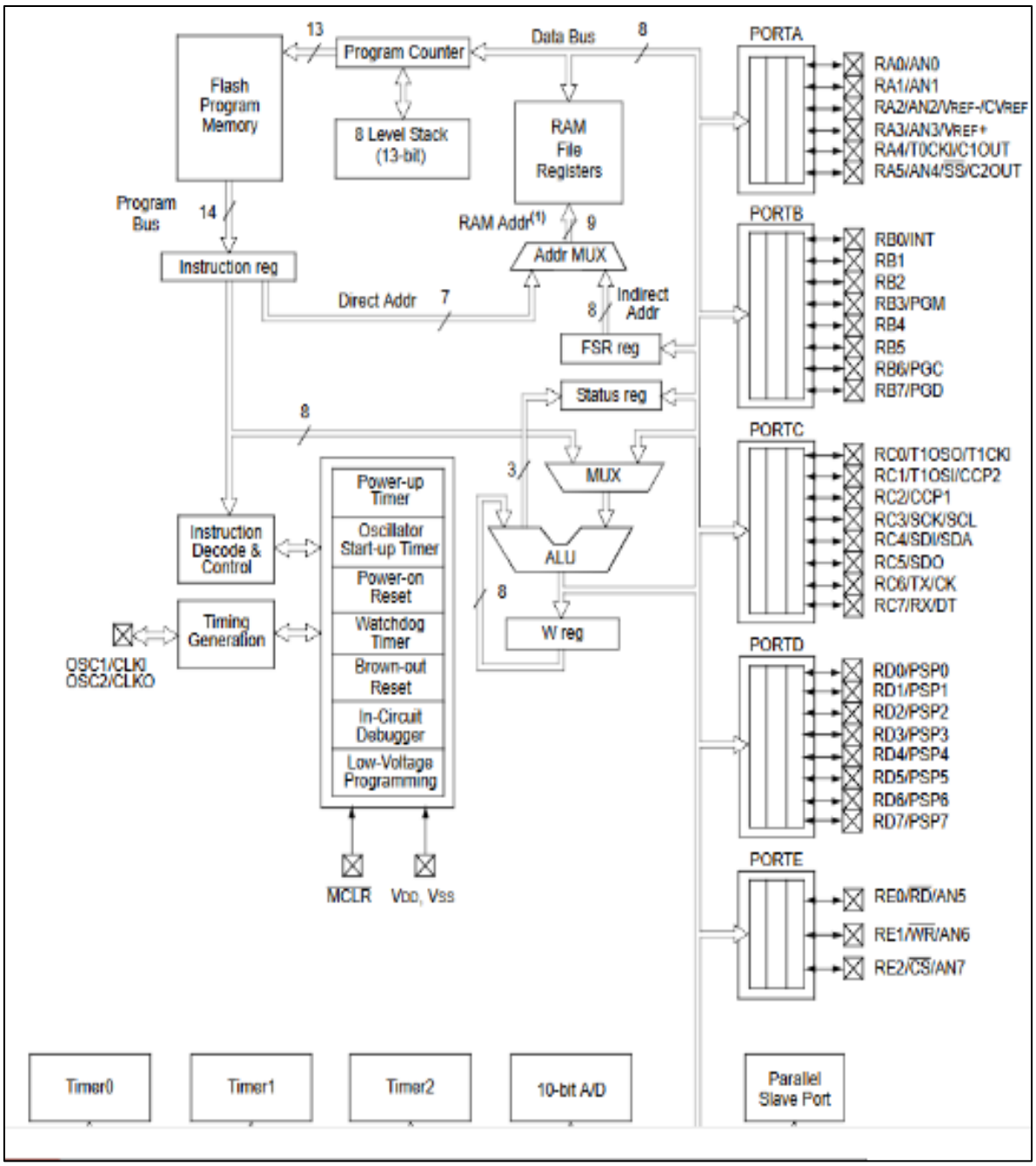
Microchip ofrece soluciones para microcontroladores de gama completa de 8bits,16 bits y 32 bits, con una poderosa arquitectura, tecnologías flexibles de la memoria, herramientas de desarrollo fácil de usar, documentación técnica completa y apoyo al diseño. Para este curso trabajaremos con el PIC 16F877.

2. CARACTERÍSTICAS

- CPU tipo RISC (conjunto de instrucciones reducidas)
- Modelos 873/6: 28 pines con tres puertos PA, PB, PC con 22 líneas de E/S y conversor A/D de 5 canales
- Modelos 874/7: 40 pines con cinco puertos PA, PB, PC, PD, PE con 33 líneas de E/S y conversor A/D de 8 canales
- Conversor A/D de 10 bits
- 35 instrucciones de 14 bits
- Ejecución de una instrucción en un ciclo, excepto las de bifurcación que la hacen en dos
- Frecuencia de 20 Mhz
- Memoria de programa flash hasta 8K x 14bits y memoria de datos RAM hasta 368 bytes, EEPROM hasta 256 bytes
- Hasta 14 fuentes de interrupción internas y externas
- Programación serie in-circuit en dos pines

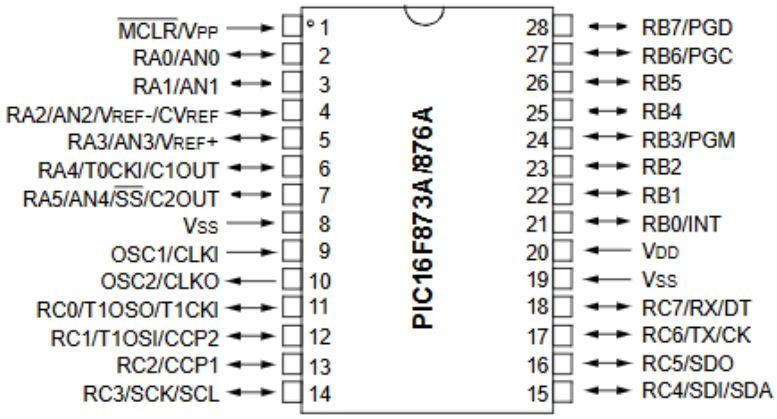
- Bajo consumo 2mA para 5V
- Tres timers: timer0, timer1, timer2
- Dos módulos de captura-comparación-pwm (CCP1 ,CCP2)
- Puerto serie síncrono (SSP) con SPI y I2C
- USART
- Puerto paralelo esclavo (PSP) para los de 40 pines

3. DIAGRAMA EN BLOQUES



4. DISTRIBUCIÓN DE PINES

28-Pin PDIP, SOIC, SSOP



5. CONJUNTO DE INSTRUCCIONES

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3

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		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
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	$\overline{TO}$,$\overline{PD}$	
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	

Como guía se ha utilizado el libro: Compilador C CCS y Simulador PROTEUS para Microcontroladores PIC de Eduardo García Breijo editado por Marcombo - Alfaomega.

CAPÍTULO 2. PUERTOS DE E/S DEL MICRO PIC 16F877

El 16F874/877 tiene cinco puertos de entrada/salida: Puerto A de 6 bits (RA5:RA0), Puerto B de 8 bits (RB7:RB0), el Puerto C de 8 bits (RC7:RC0), el Puerto D de 8 bits (RD7:RD0) y el Puerto E de 3 bits (RE2:RE0). En el puerto A, el pin RA4tiene salida en colector abierto lo que obliga a utilizar una resistencia pull up si se utiliza como salida. Este pin tiene entrada con trigger smitt ideal para utilizarlo como contador de eventos externos. El puerto B tiene resistencias pull up internas que tienen que habilitarse.

Los registros de dirección del puerto se hacen a través del correspondiente registro TRIS: TRISA, TRISB, TRISC, TRISD, TRISE, que están en la memoria RAM. Si el bit de TRIS=1 corresponde a una entrada y si el bit TRIS=0 es de salida. Ejemplo: TRISB=10010010, quiere decir que los pines son: RB0(salida), RB1(entrada), RB2(salida), RB3(salida), RB4(entrada), RB5(salida), RB6(salida), RB7(entrada). Como se observa en su arquitectura estos pines de los puertos son multifuncionales como convertidor AD, USART, I2C, etc, que mas adelante veremos su aplicación.

La gestión del bus de datos se realiza a través de los registros PORTA, PORTB, PORTC, PORTD, PORTE que hacen parte de la RAM.

Para manejar los puertos en lenguaje C, se utilizan las directivas del compilador:

- #use fast_io
- #use fixed_io
- #use standard_io

En los ejemplos se va a utilizar la directiva #standard_io, que tiene las siguientes funciones para manejar los puertos:

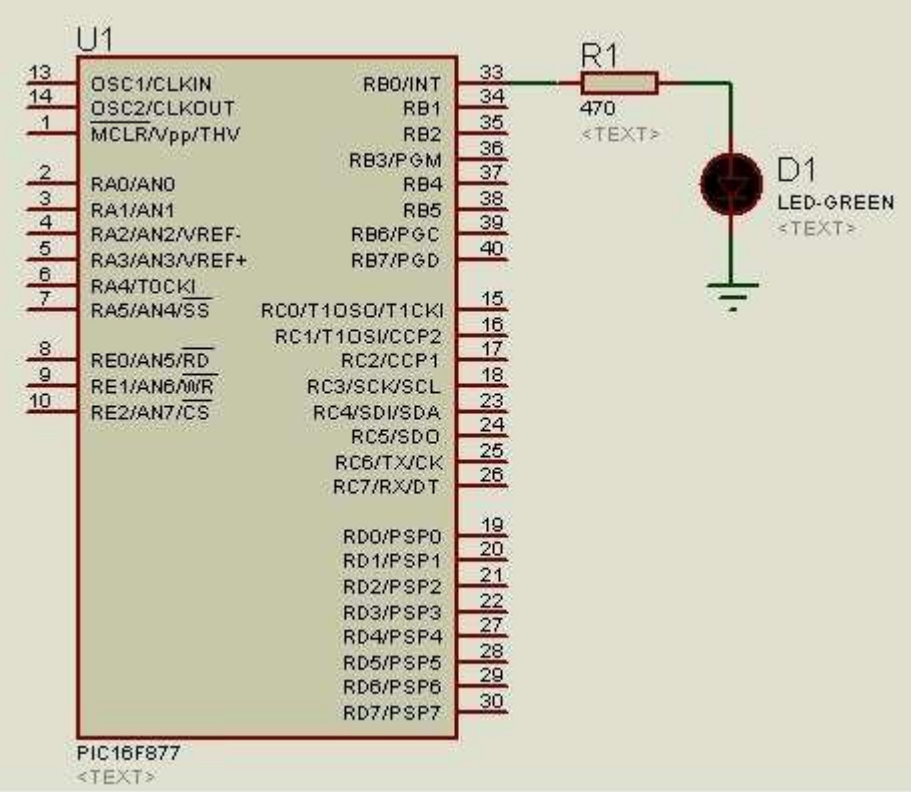
- output_X(valor) // para sacar un valor a puerto (0-255)
- input_X(valor)   // para leer un puerto
- port_B_pullups(valor) // habilta (valor=true) o deshabilita (valor=false) las resistencias pull up del puerto B
- output_low(pin_*) //pone pin =0 a la salida
- output_high(pin_*) //pone pin =1 a la salida
- output_toggle(pin_*)   //complementa el valor del pin
- input(pin_*)   // lee el pin de entrada
- bit_clear(var,bit); // pone bit de la variable =0
- bit_set(var,bit); // pone bit de la variable =1

A medida que vayamos haciendo programas vamos profundizando sobre el compilador CCS.

2.1 EJEMPLO: PRENDER UN LED

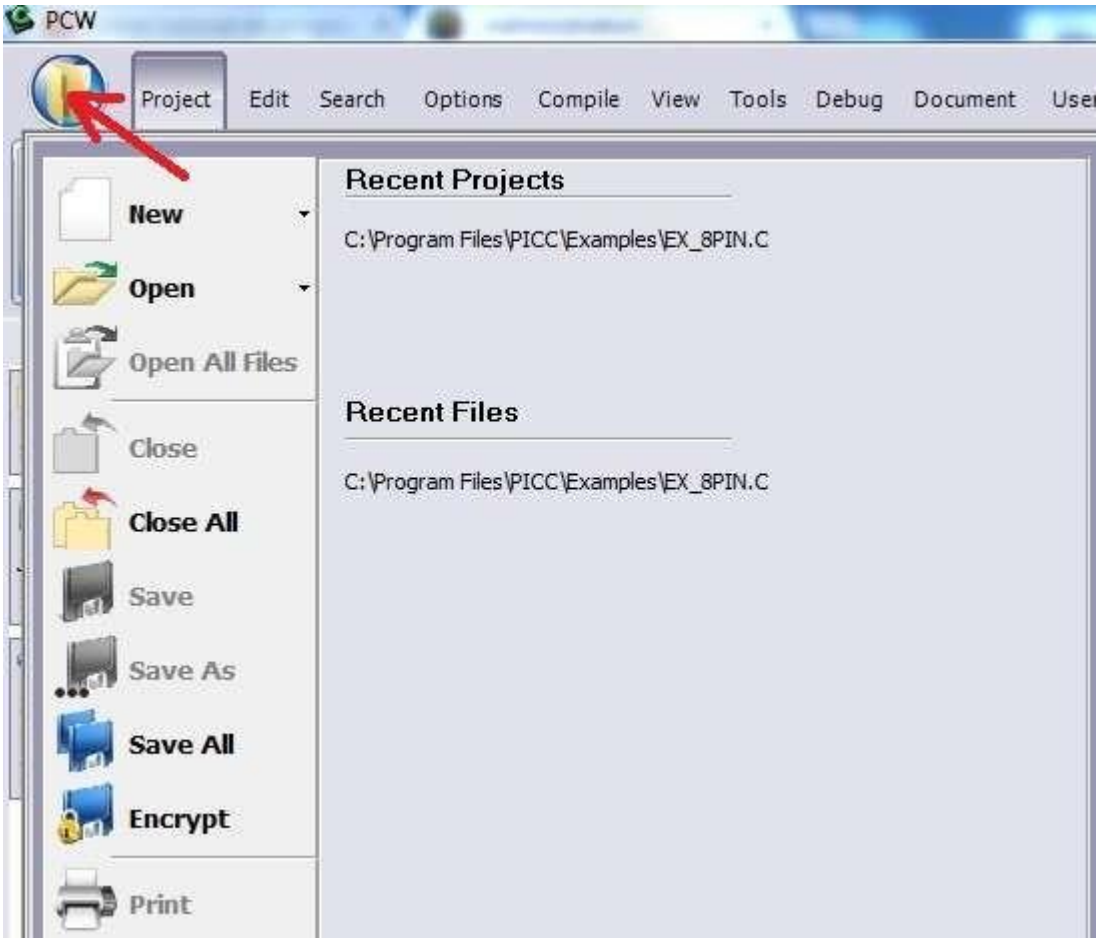
Este es el primer ejercicio que se va a resolver en el microcontrolador 16F877 que consiste en prender un LED que se ha colocado en en el pin RB0 del puerto 0 que se programará como un pin de salida tal como se presenta en este circuito que se ha realizado en el simulador Proteus.

1. Implementar hardware



Como ya se sabe un microcontrolador para que funcione además del hardware se debe utilizar el software mediante el cual el micro lo ejecuta. La programación que se implementará en este curso es el Lenguaje C cuyo compilador (traductor a lenguaje de máquina del micro) escogido es el CCS (Custom Computer Services). Yo he descargado la versión 4 de internet pero ya existe la versión 5. El siguiente es el ícono del compilador CCS

2. Ejecutar el compilador, aparece la siguiente ventana, dar clic donde está la flecha para la edición de un nuevo programa con New.



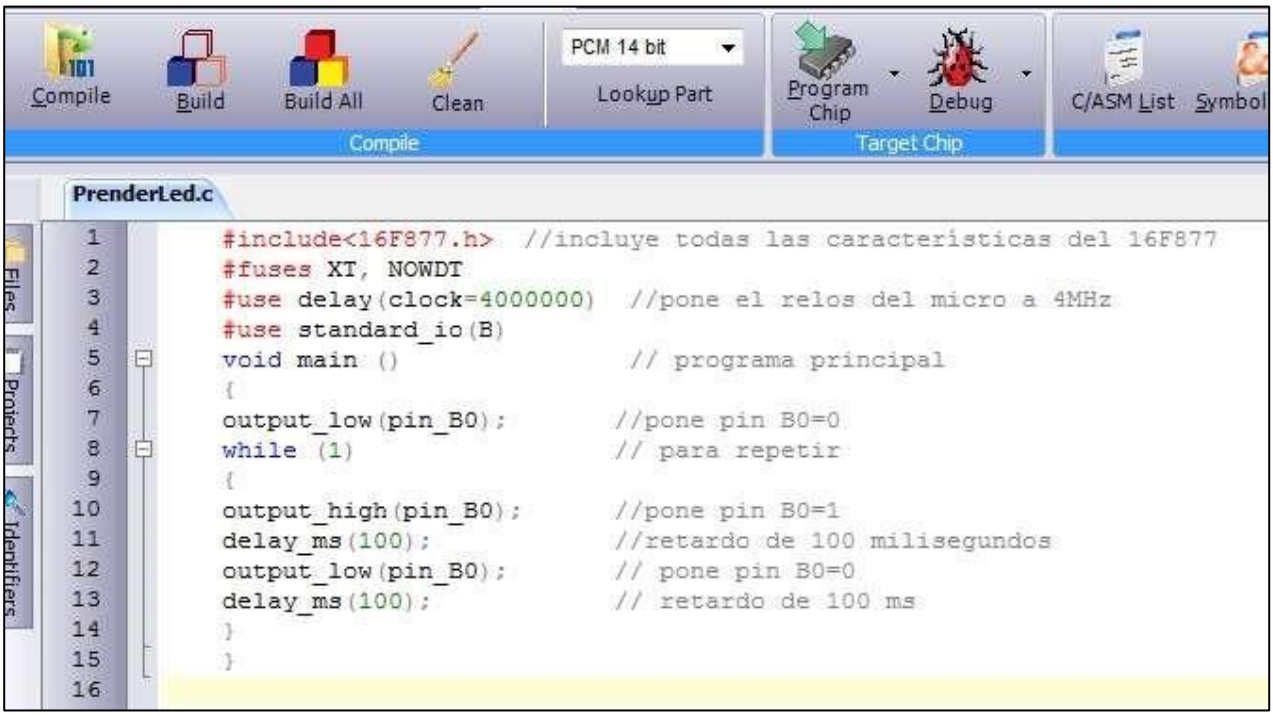
3. Editar el programa:

En general un programa consta de las siguientes partes:

Comentario: Se escribe // a comienzo de la línea o para bloque de comentario /*....*/

Directivas del procesador: Por ejemplo #include, #define, #use delay, #use standard_io, #fuses

Definición de datos: int8, int (8bits), int16, long (16bits), float(32bits), char Definición de las Funciones: delay_ms, input, output



Comentarios.

La directiva #fuses no afecta la compilación, pero si la programación (quemado). Tiene las siguientes opciones:

LP, XT, HS, RC: Para el reloj, XT para 4 MHz, HS para 20 MHz

WDT, NOWDT: Perro guardián

PROTECT, NOPROTECT

PUT, NOPUT (Power Up Timer)

BROWNOUT, NOBROWNOUT

Para poner el reloj puede ser también: #use delay(clock=4MHZ)

delay_cycles(cuenta): retardo en ciclos del pic de 1 a 255

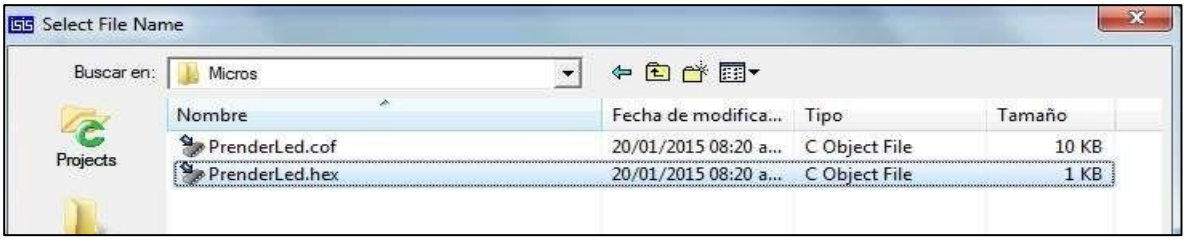
delay_ms(tiempo): retardo en milisegundos

delay_us(tiempo): retardo en microsegundos

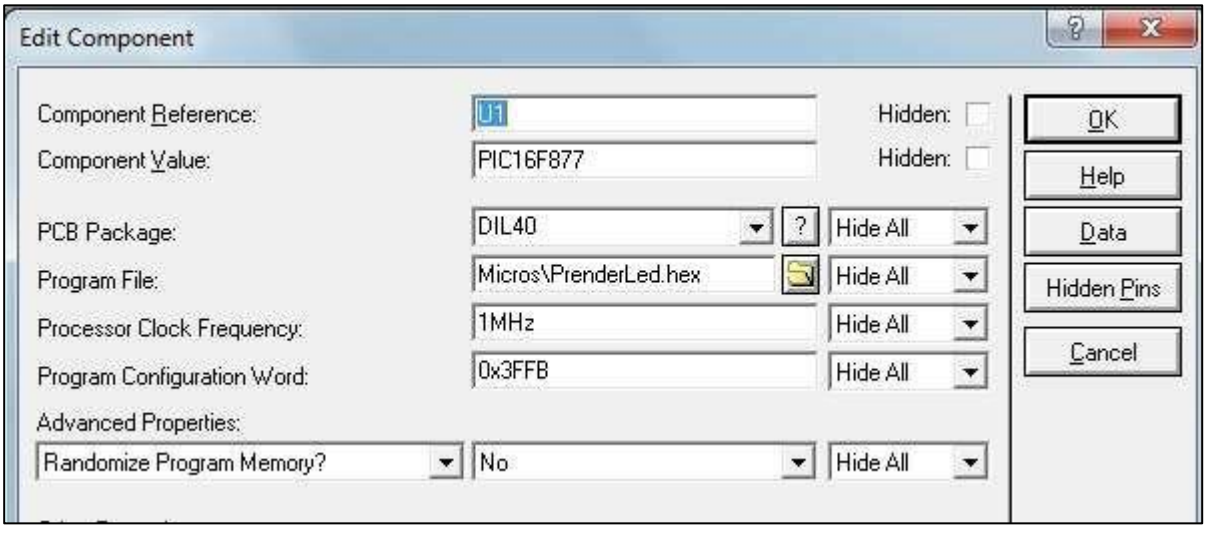
Ejemplo: #fuses HS, NOWDT, NOPROTECT, NOLVP para reloj de 20 MHz NOLVP es para no quemar con bajo voltaje

4. Guardar el programa con un nombre, para este caso PrenderLed. Compilar dando clic en Compile y se generan dos archivo el .cof y el .hex.

El archivo PrenderLed.hex es el que se carga en el micro del circuito simulado con Proteus.



Se da doble clic al micro, aparece esta ventana y se carga el programa PrenderLed.hex

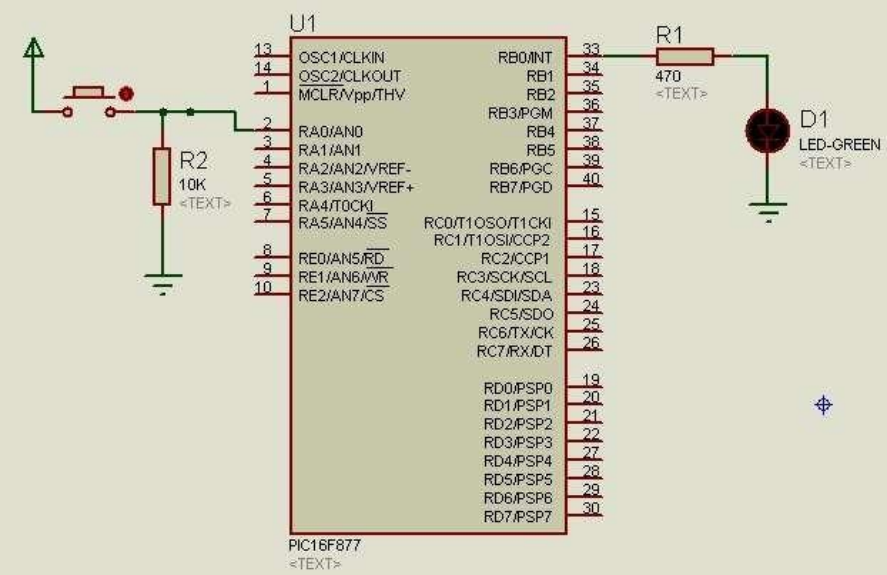


El circuito ya está para su simulación en Proteus. El Led se prenderá y apagará los retardos programados, en este caso 100 milisegundos.

2.2 EJEMPLO: LEER UN PULSADOR

Leer un pulsador colocado en el pin A0 y proyectarlo en el pin B0. Se utiliza la instrucción: if.

HARDWARE CON PROTEUS



SOFTWARE CON CCS

LeerPulsador.c

```
1   #include<16F877.h>  //incluye todas las caracteristicas del 16F877
2   #fuses XT, NOWDT
3   #use delay(clock=4000000)  //pone el reloj del micro a 4MHz
4   #use standard_io(B)
5   #use standard_io(A)
6   void main ()                // programa principal
7   {
8    output_low(pin_B0);        // B0=0
9    while(true)                // para repetir
10   {
11       if(input(pin_A0)==0)   // si A0=0
12          output_low(pin_B0); // B0=0
13       else                   // de lo contrario, si A0=1
14          output_high(pin_B0); // B0=1
15   }
16  }
17
18
```

While: es una instrucción que se usa cuando se requiere una iteración o loop.

La sintaxis es: while (expresión)

La expresión es evaluada y la instrucción es ejecutada hasta que llegue a ser falsa y en este caso la ejecución del programa continua después de este loop.

if - else: Es utilizada para tomar decisiones. La sintaxis es:

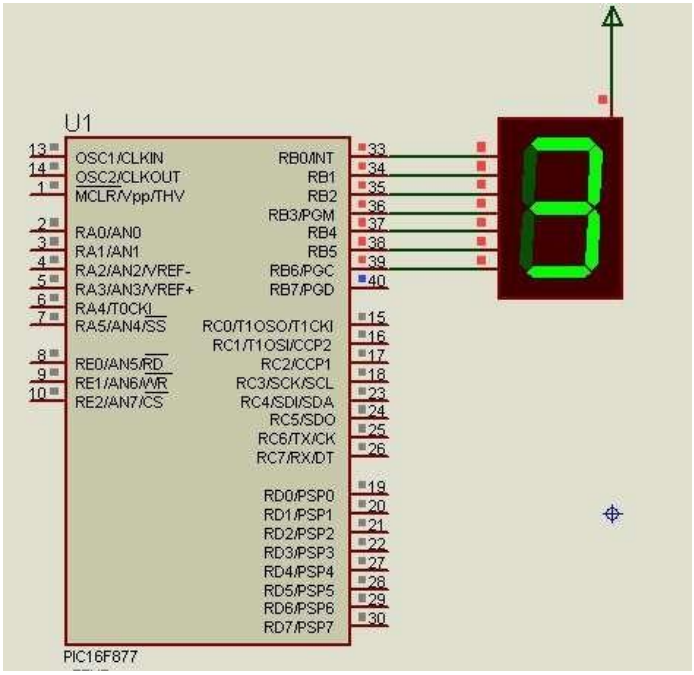
if (expr) instrucción1; [else instrucción2;]

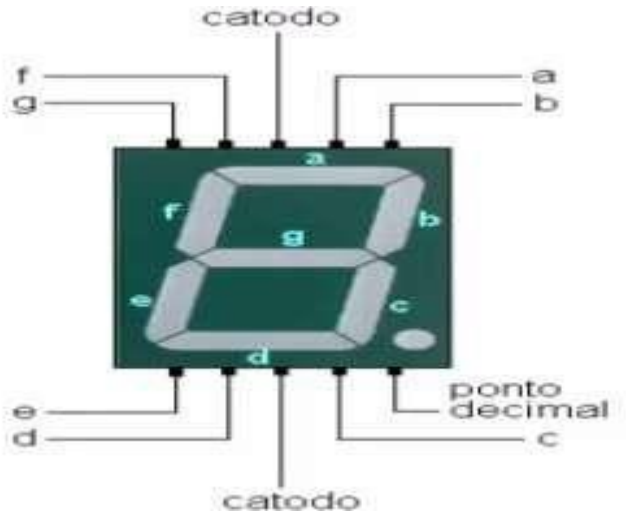
La expresión es evaluada y si es verdad se ejecuta la instrucción1 y si es falsa se ejecuta la instrucción2.

2.3 EJEMPLO: MANEJO DE DISPLAY DE 7 SEG

El display de 7 segmentos es de ánodo común que se conecta al puerto B de RB0 a RB6. Este es un sencillo programa contar en forma ascendente de 0 a 3 y regresar. B0 se conecta al segmento a, B1 al b, B2 al c, B3 al d, B4 al e, B5 al f y B6 al g. Para que despliegue el 3 se deben iluminar a, b, c, d, g, o sea, cargar B=B6B5B4B3B2B1B0=0110000.

HARDWARE CON PROTEUS





SOFTWARE CON CCS

Display.c

```
#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)
int16 tiempo=500;
void main ()                // programa principal
{
while(true)                // para repetir
 {
    output_B(0b1000000);
    delay_ms(tiempo);
    output_B(0b1111001);
    delay_ms(tiempo);
    output_B(0b0100100);
    delay_ms(tiempo);
    output_B(0b0110000);
    delay_ms(tiempo);
 }
}
```

En este programa se usa una variable entera de 16 bits (int16) llamada tiempo para poner el tiempo de retardo. De esta forma se puede cambiar rápidamente el retardo con delay_ms(tiempo).

2.4 EJEMPLO: DISPLAY LCD

Para visualizar los datos de salida de un microcontrolador se utilizan comúnmente los display tipo LCD que sacan la información por líneas y cada línea tiene un determinado número de caracteres. Por ejemplo, display de dos líneas y cada línea de 16 caracteres

con un bus de datos de 8 bits o de 4 bits. El compilador C tiene un archivo o driver para trabajar con LCD el lcd.c que debe llamarse como un #include. Este driver trabaja con el puerto D o el puerto B. Por defecto utiliza el D, pero si se quiere el B hay que poner:#define use_portB_lcd true

Este archivo dispone de las siguientes funciones:

lcd_init () // la primera función a ser llamada, borra el LCD y pone formato de 4 bits.

lcd_gotoxy(x,y) // indica la posición en x,y

lcd_getc(x,y) // lee el caracter de la posición x,y lcdputc(char) //escribe la variable en la posición correspondiente. Si se agrega:

\f (limpia el lcd),

\n (lleva cursor a la posición 1,2),

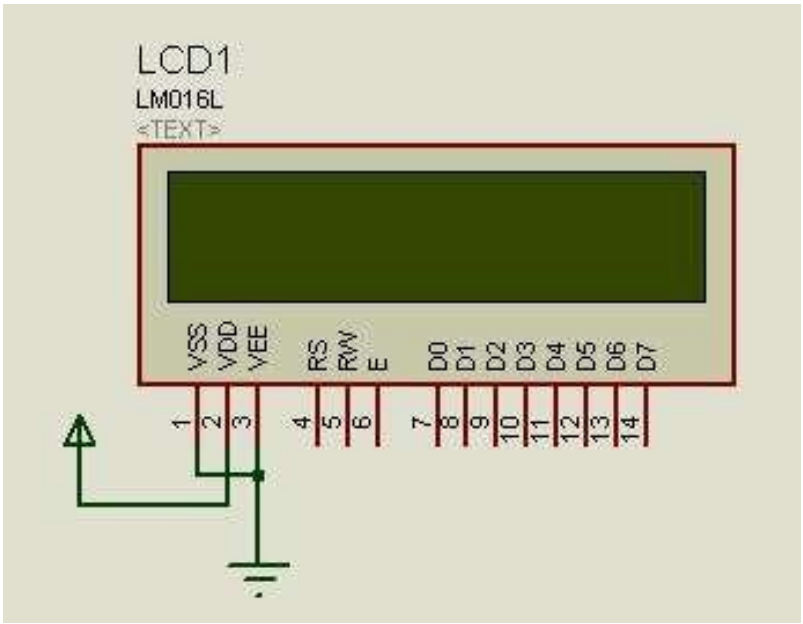
\b (retrocede cursor)

printf(string) //escribe cadena de caracteres entre comillas "...."

printf(cstring, values....) //escribe lista de variables separadas por comas

printf(fname. cstring, values) // fname es una función

Aquí tenemos un display de 2 líneas de 16 caracteres.

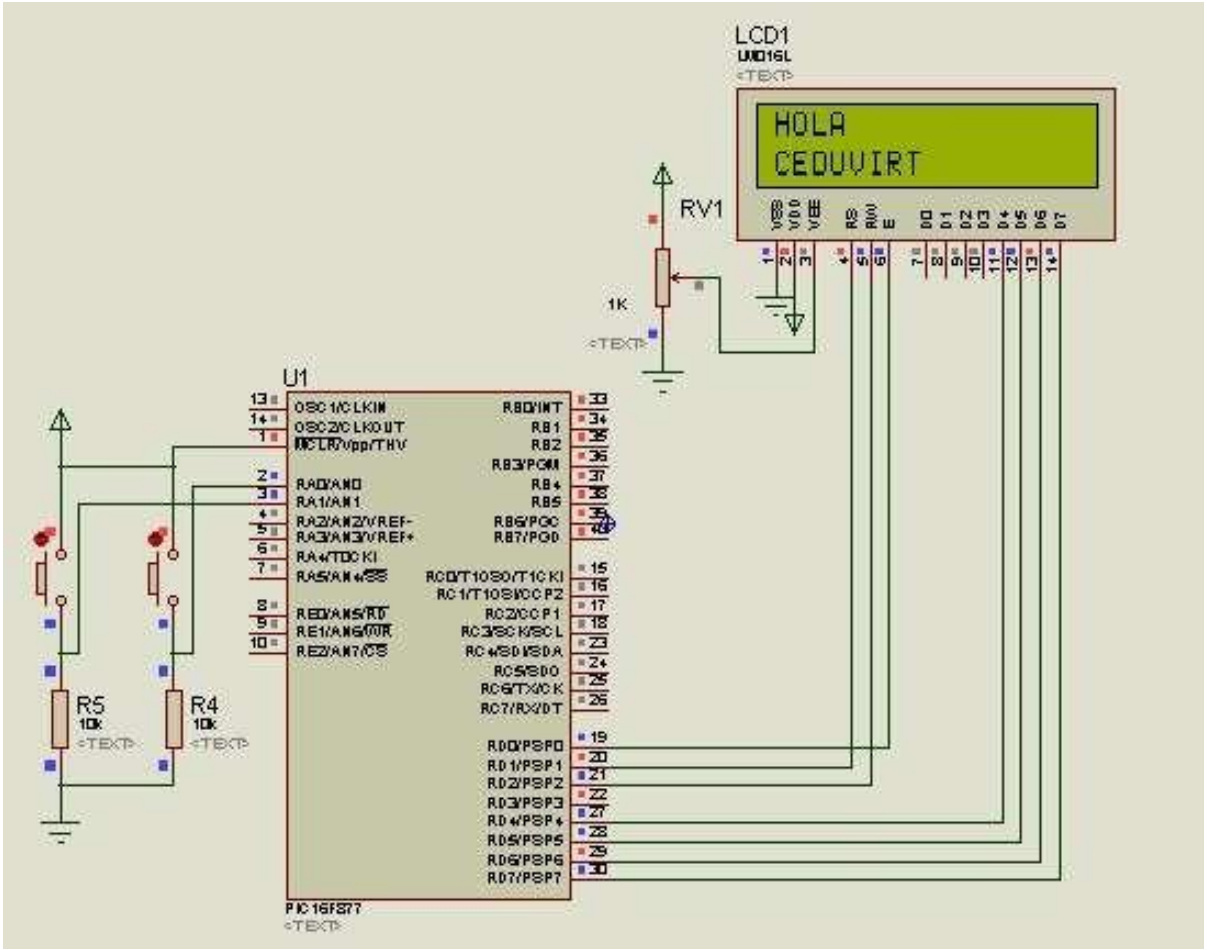


VSS (tierra), VDD (alimentación), VEE (contraste), RS (selecciona registro), R/W (leer o escribir), E (Habilitador), D0...D7 (datos) D0 es el menos significativo (LSB).

D0 ->E, D1 ->RS, D2 ->RW, D4 ->D4, D5 ->D5, D6 ->D6, D7 ->D7

Este es un ejemplo para desplegar HOLA carácter por carácter en la primera línea pulsando el pin A0 y la palabra CEDUVIRT en la segunda línea pulsando el pin A1.

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
#use standard_io(A)
#use standard_io(D)
```

```
void main ()      // programa principal
{
  port_b_pullups(true);
  lcd_init ();
  while (true)
  {
   if(input(pin_A0)==1)
   {
     lcd_gotoxy(1,1);
     lcd_putc('H');
     lcd_putc('O');
     lcd_putc('L');
     lcd_putc('A');
     delay_ms(300);
   }
   if(input(pin_A1)==1)
   {
     lcd_gotoxy(1,2);
     printf(lcd_putc,"CEDUVIRT");
     delay_ms(300);
   }
   }
  }
```

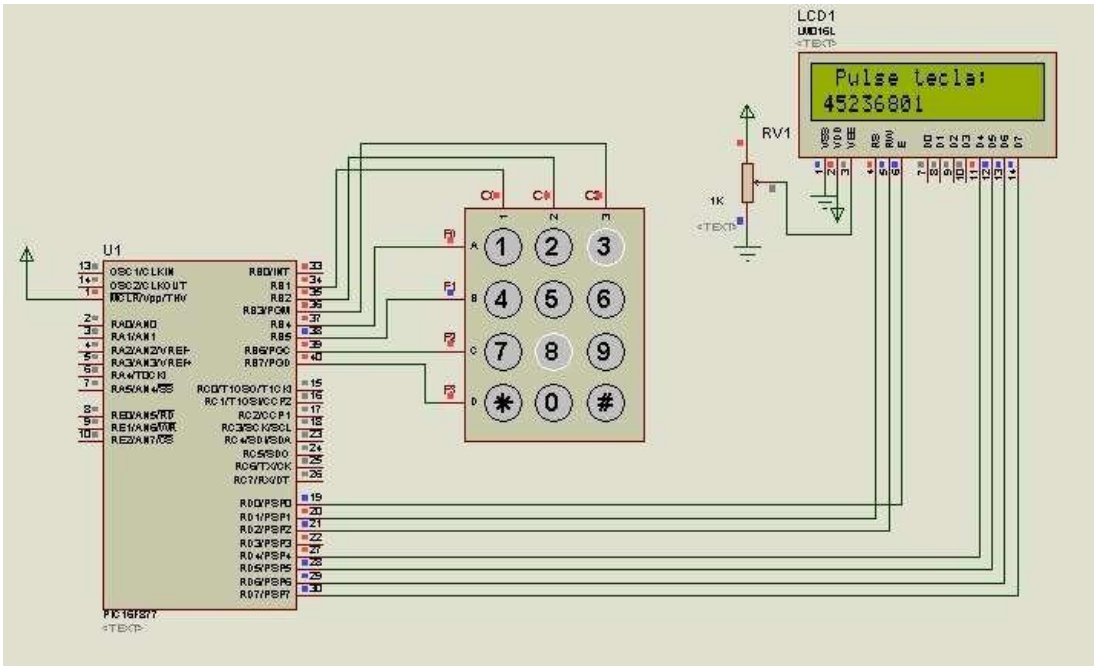
2.5 EJEMPLO: TECLADOS MATRICIALES

Los teclados matriciales son muy habituales en los sistemas con microcontroladores para entrar datos. Está conformado por una serie de pulsadores (teclas) en forma de columnas y filas. En este ejemplo se va a leer un teclado de 4x3 (4 filas y 3 columnas) y desplegarlo en un display LCD de dos filas de 16 caracteres cada una.

Para trabajar teclados el compilador debe tener en su Driver el programa o rutina para manejo del teclado. Para este ejercicio se ha considerado el **kgb_portb.c**, que tiene la función kgb_getc() para tomar el valor de la tecla pulsada. Como se va a colocar en el puerto B se debe habilitar con la dirtectiva: use_portb_kbd true.

Para el ejemplo siguiente se va escribir una frase en primera línea y desplegar las teclas pulsadas (hasta 16) en la segunda línea del display.

HARDWARE CON PROTEUS



SOFTWARE CON CCS

TecladoDisplay.c

```
#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
#include<kbd_portb.c>
#use fast_io(B)
#use fast_io(D)
#define use_portb_kbd true  //habilitar puerto B

 void main(void)
```

```
char c;   //variable donde se almacena tecla pulsada
int i;    // entero de 8 bits
lcd_init();                 //inicializa lcd
lcd_putc(" Pulse tecla:" ); //despliegue en primera fila
lcd_gotoxy(1,2);            //donde se va a mostrar tecla, col=1 fila=2
delay_ms(500);              //retardo de 500msg
```

```
for(i=0;i<16;i++){ //bucle para escribir hasta 16 caracteres
do{                //espera hasta ....
c=kbd_getc();      // lleva tecla a la variable c
}
while(c==0); //...pulsar una tecla
printf(lcd_putc,"%c",c);  // despliega tecla como caracter
   }
```

kbd_portb.c

```
char const KEYS[4][3] = {{'1','2','3'} ,
                         {'4','5','6'} ,
                         {'7','8','9'} ,
                         {'*','0','#'} };

#byte kbd_port_b = 6  // dirección del puerto B en RAM

char kbd_getc( )
{
  char tecla=0;
  int f,c,t,i,j;
  port_b_pullups(true); // para utilizar puerto B
  set_tris_b(0b00001111); // RB3-RB0 entradas, RB7-RB4 salidas
// i vector para filas=4, j vector para columnas =3
  for(f=0x10, i=0; i<4; f<<=1, i++)  //poner 1 en RB4 y correr a la izq
  {
    for(c=0x02, j=0; j<3; c<<=1, j++)  // poner 1 en RB1 y correr a la izq
    {
      kbd_port_b = ~f;
      delay_cycles(1);
      t = kbd_port_b & 0x0F;
      t = ~(t | 0xF0 );

      if(t == c)
      {
        delay_ms(20);
        tecla=KEYS[i ][j ];
        while(t==c)
        {
          restart_wdt( );
          t = kbd_port_b & 0x0F;
          t = ~(t | 0xF0) ;
        }
        break;
      }
    }
    if(tecla)
      break;
  }
  port_b_pullups(false);
  return tecla;
}
```

Comentarios

En este programa aparecieron ciertos operadores y agrego otros más que necesitan ser explicados:

++ Incremento

-- Decremento

!= Distinto

<<= Corrimiento a la izquierda

>>= Corrimiento a la derecha

~ Complemento a 1's

&& And lógico

|| Or lógico

! Negación lógica

& And de bits

| Or de bits

Encontramos estas nuevas instrucciones:

for: Es también usada para iteraciones (loop). La sintaxis es: for(expr1; expr2; expres3)

La expr1 es la inicialización, la expr2 chequea la terminación y la expr3 es el incremento. Cualquiera de ellas puede ser omitida. Para un bucle (loop) sin fin se utiliza: for (; ;)

do-while: La terminación se evalúa al final del bucle. Sintaxis: do {sentencias} while(expr). Para un bucle sin fin:

while(1) {sentencias} do {sentencias} while(1) return: Devulce datos a las funciones break: Permite salir de un bucle goto: Provoca un salto incondicional

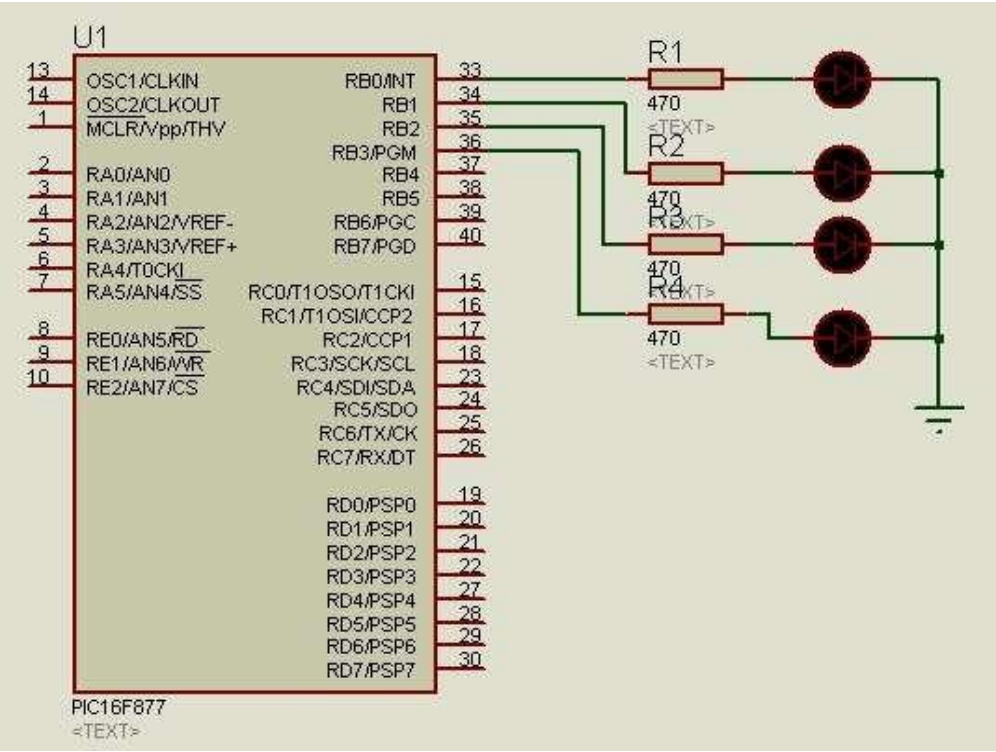
CAPÍTULO 3. PUERTOS DE E/S PIC - SIMULACIÓN

Las simulaciones se harán programando en lenguaje C y utilizando el Compilador CCS. El hardware se simulará con el software Proteus de Labcenter Electronics.

3.1 SIMULACIÓN 1. JUEGO DE LUCES

Hacer dos juegos de luces, uno activando un led seguido de otro cuatro veces y luego activándolos de dos en dos también cuatro veces. Se utiliza la instrucción for.

HARDWARE CON PROTEUS

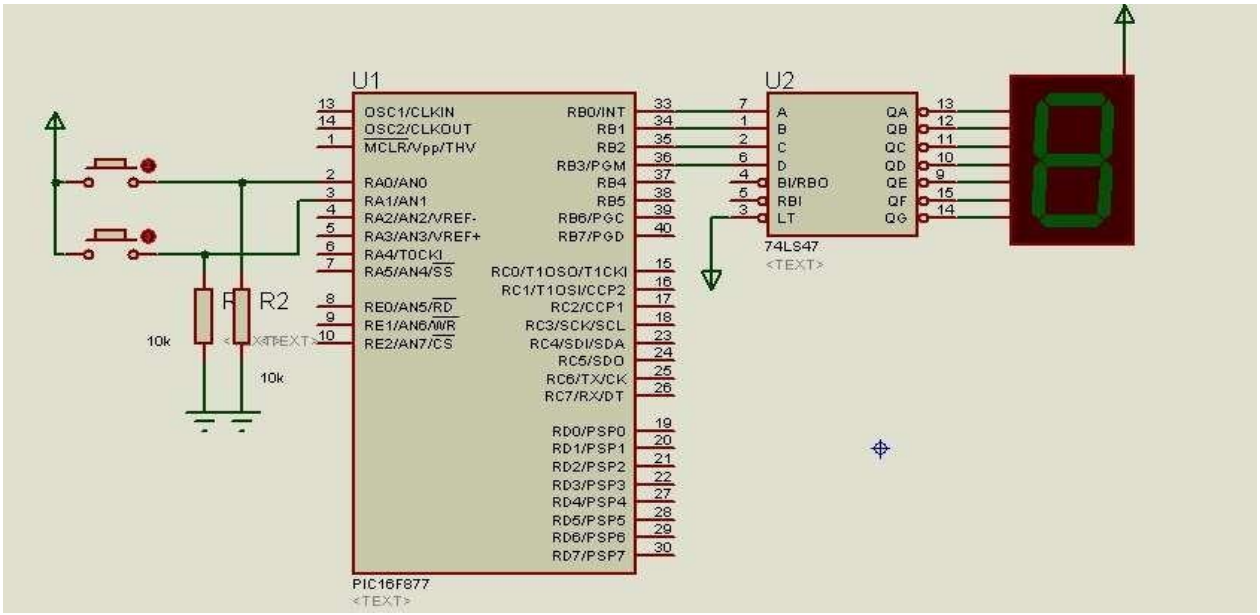


SOFTWARE CON CCS

```
#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)
int i=0;
int16 tiempo=500;
void main ()                // programa principal
{
while(true)                // para repetir
 {
   //primer juego
    for(i=0;i<=3;i++)
    {
    output_B(0b0000);
    delay_ms(tiempo);
    output_B(0b0001);
    delay_ms(tiempo);
    output_B(0b0010);
    delay_ms(tiempo);
    output_B(0b0100);
    delay_ms(tiempo);
    output_B(0b1000);
    delay_ms(tiempo);
    }

    // segundo juego
    for(i=0;i<=3;i++)
    {
    output_B(0b0000);
    delay_ms(tiempo);
    output_B(0b0011);
    delay_ms(tiempo);
    output_B(0b0110);
    delay_ms(tiempo);
    output_B(0b1100);
    delay_ms(tiempo);
    }
 }
}
```

3.2 SIMULACIÓN: CONTADOR UP-DOWN

HARDWARE CON PROTEUS



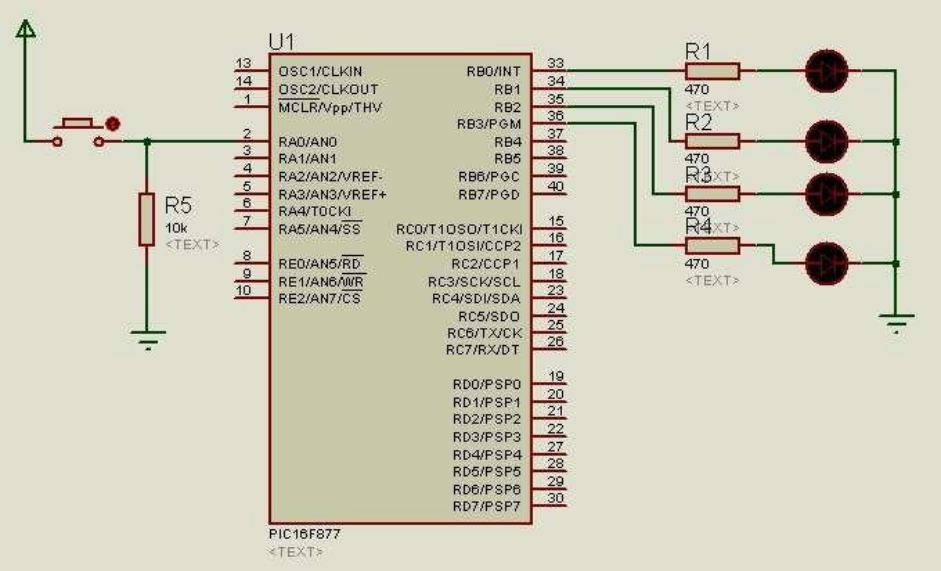
SOFTWARE CON CCS

Contador.c

```
#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)

void main ()      // programa principal
{
int contador=0;
output_B(0xFF);
  while(true)                // para repetir
  {
     if(input(pin_A0)==1)
     {
     contador++;
     output_B(contador);
     delay_ms(500);
     }
     if(input(pin_A1)==1)
     {
     contador--;
     output_B(contador);
     delay_ms(500);
     }
  }
}
```

3.3 SIMULACIÓN: CORRIMIENTOS

HARDWARE CON PROTEUS



SOFTWARE CON CCS

Corrimiento.c

```
/* CURSO DE MICROCONTROLADORES
CEDUVIRT-CENTRO DE EDUCACIÓN VIRTUAL
CORRIMIENTOS A LA DERECHA Y A LA IZQUIERDA
*/

#include<16F877.h>  //incluye todas las características del 16F877
  #fuses XT, NOWDT
  #use delay(clock=4000000)  //pone el reloj del micro a 4MHz
  #use standard_io(B)

  void main ()     // programa principal
  {
 short sentido=true;
 int variable=0b0001;
 long tiempo=300;

    while(true)                // para repetir la rutina
   {
    if(sentido==true)
      {
      output_B(variable);
      if(variable==0b0000)
         variable=0b0001;
      else
         variable=variable<<1;
      delay_ms(tiempo);
      }
```

```
28      // si A0=1 corrimiento a la derecha  sentido=false
29        else
30         {
31          output_B(variable);
32          if(variable==0b0000)
33             variable=0b1000;
34          else
35             variable=variable>>1;
36             delay_ms(tiempo);
37          }
38
39        if(input(pin_A0)==1)
40          {
41            if(sentido==true)
42               sentido=false;
43            else
44               sentido=true;
45            delay_ms(tiempo);
46          }
47        }
48      }
49
```

CAPÍTULO 4. ARDUINO UNO E/S - TEORÍA

4.1 ESTRUCTURA DE LA PLACA ARDUINO

El Arduino Uno es una placa electrónica basada en el Microcontrolador ATmega328. Cuenta con 14 pines digitales de entrada / salida (de los cuales 6 pueden utilizarse para salidas PWM), 6 entradas analógicas, un resonador cerámico 16MHz, una conexión USB, un conector de alimentación, una cabecera ICSP, y un botón de reinicio. Contiene todo lo necesario para apoyar el microcontrolador; basta con conectarlo a un computador con un cable USB o la fuente de poder con un adaptador de CA o la batería a CC.

1. CARACTERÍSTICAS

Microcontrolador	**ATmega328**
Tensión de Funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines digitales I / O	14 (de las cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC por Pin I / O	40 mA
Corriente DC de 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad De Reloj	16 MHz

El Arduino Uno puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente.

Potencia (no USB) externa puede venir con un adaptador de CA a CC o la batería. El adaptador se puede conectar al conectar un enchufe de 2.1mm centro positivo en el

conector de alimentación de la placa. Los cables desde una batería se pueden insertar en los cabezales de pin GND y Vin del conector de alimentación.

El tablero puede funcionar con un suministro externo de 6 a 20 voltios. Si se suministra con menos de 7V, sin embargo, el pin de 5V puede suministrar menos de cinco voltios y la junta puede ser inestable. Si se utiliza más de 12 V, el regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son como sigue:

- VIN. El voltaje de entrada a la placa Arduino cuando se trata de utilizar una fuente de alimentación externa (en oposición a 5 voltios de la conexión USB u otra fuente de alimentación regulada). Se puede suministrar tensión a través de este pin.
- 5V. Este pin es de salida, 5V regulados por la board. La board puede ser alimentada ya sea desde la toma de alimentación de CC (7 - 12 V), el conector USB (5V), o por el pin VIN del tablero (7-12V). El suministro de tensión a través de los pines de 5V o 3.3V no pasa por el regulador, y puede dañar su board. No es aconsejable.
- 3V3. Un suministro de 3,3 voltios generada por el regulador de la board. Suministra una corriente máxima es de 50 mA.
- GND. Pines de tierra.
- Instrucción IOREF. Este pin de la placa o board Arduino proporciona la referencia de tensión con la que opera el microcontrolador.

2. MEMORIA

El ATmega328 tiene 32 KB de memoria FLASH, 2 KB de SRAM (memoria estática) y 1 KB de EEPROM (memoria borrable y escribible).

3. ENTRADA - SALIDA

Cada uno de los 14 pines digitales en el ARDUINO UNO se puede utilizar como una entrada o salida, utilizando las funciones pinMode (), digitalWrite (), y digitalRead (). Funcionan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectado por defecto) de 20 a 50 Kohm. Además, algunos pines tienen funciones especializadas:

Serial: RX (0) y TX (1) Se utiliza para recibir (RX) y transmitir datos en serie (TX).

Interrupciones externas: 2 y 3. Estos pines pueden configurarse para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor.

PWM: 3, 5, 6, 9, 10, y 11. proporcionan una salida PWM de 8 bits con la función analogWrite ().

SPI: SS (10), MOSI (11), MISO (12), SCK (13) Estos pines admite la comunicación SPI utilizando la librería SPI.

LED: 13. Hay un LED incorporado conectado al pin digital 13. Cuando el pasador es de alto valor, el LED está encendido, cuando el pasador es bajo, es apagado.

El Uno tiene 6 entradas analógicas, etiquetado A0 a A5, cada uno de los cuales proporcionan 10 bits de resolución (es decir, 1.024 valores diferentes). Por defecto se miden desde tierra a 5 voltios, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la función analogReference (). Además, algunos pines tienen funciones especializadas:

TWI: Pin A4 o A5 o SDA y SCL para comunicación I2C. Apoyo TWI utilizando la librería Wire.

Hay un par de patas de la placa:

AREF. Voltaje de referencia para las entradas analógicas. Se utiliza con analogReference ().

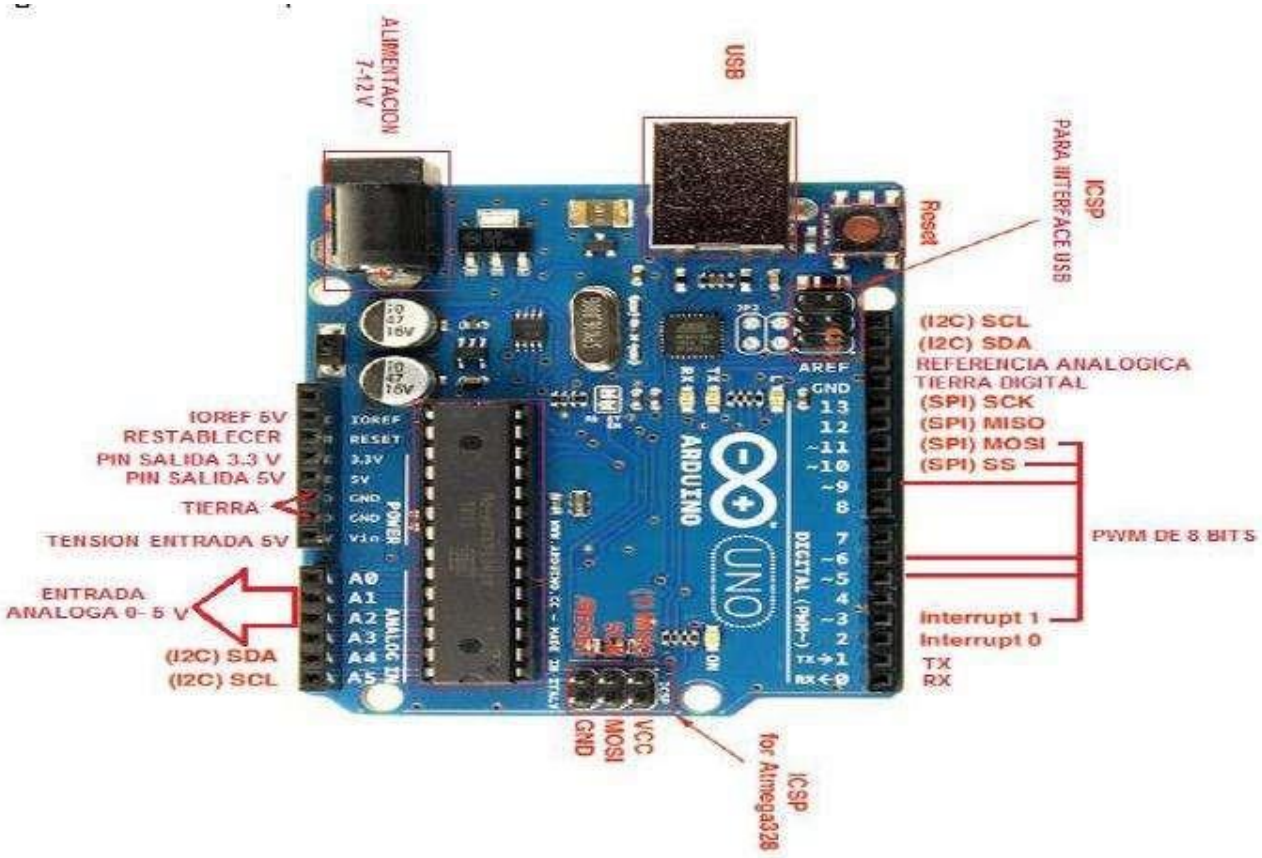
RESET. Se coloca esta línea en BAJO para reajustar el microcontrolador.

4. **COMUNICACIÓN**

El Arduino Uno tiene una serie de instalaciones para comunicarse con un computador, otro Arduino u otros microcontroladores. El ATmega328 ofrece UART TTL (5V) de comunicación en serie, que está disponible en los pines digitales 0 (RX) y 1 (TX).

Una biblioteca Software Serial permite la comunicación en serie en cualquiera de los pines digitales del Uno.

El ATmega328 también es compatible I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso del bus I2C. Para la comunicación SPI, utilice la librería SPI.



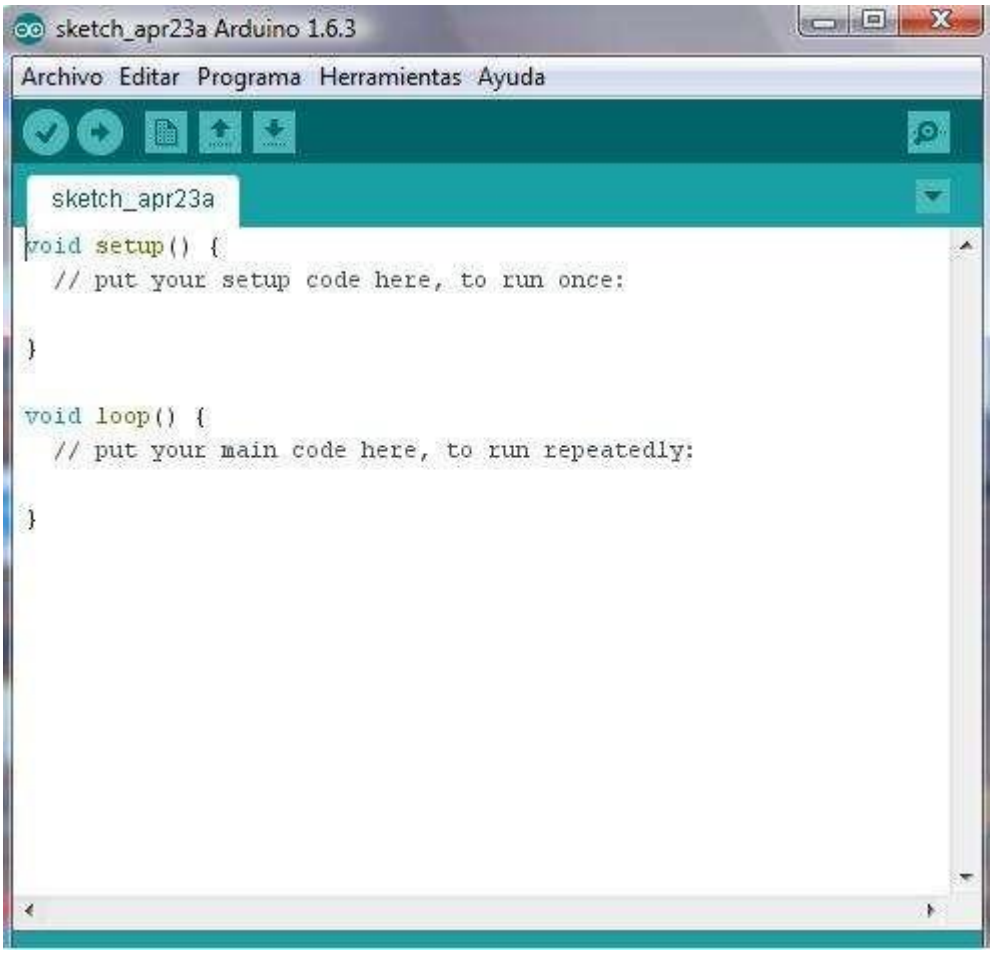
4.2 ARDUINO E/S - SIMULACIÓN

Para trabajar con el Arduino lo primero que se debe hacer es descargar el software para desarrollar aplicaciones. este software es gratuito y libre, de ahí su importancia. Se descarga del siguiente enlace:

http://www.arduino.cc/en/Main/Software

Aparece la ventana arriba en donde seleccionamos el sistema operativo. En este caso Windows.

1. Después de descargarlo, Arduino 1.6.3.exe, lo instalamos
2. Conectar el Arduino al computador por medio del cable USB. Hay que comprobar a qué puerto quedó conectado, se hace:
   Panel de control -->Administrador de dispositivos-->Puertos(COM, LPT)
   -->Arduino Uno(COM8)
3. Abrir el Arduino para configurarlo. Aparece esta ventana.



Los íconos de la parte superior de la ventana indica: verificar (para observar si hubo errores en la compilación), Subir (enviar el programa del computador a la tarjeta Arduino),

Nuevo (para editar un nuevo programa), Abrir (para abrir un programa ya realizado) y Salvar (para guardar el programa).

Ir a:

Herramientas-->Placa-->Arduino Uno-->Serial Port-->COM8

4. Escribir mi primer programa:

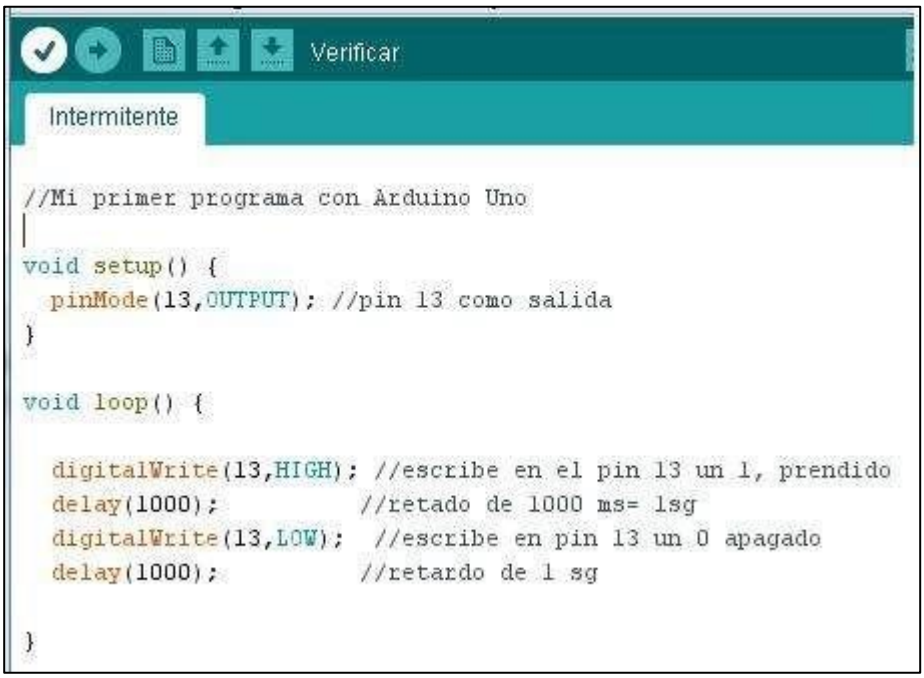
El programa tiene dos partes:

Void setup: Para escribir la programación por ejemplo de los puertos como de entrada o de salida,

Void loop: Es el principal donde escriben las instrucciones o funciones del Arduino para implementar el problema planteado.

EJEMPLO: ENCENDER Y APAGAR UN LED

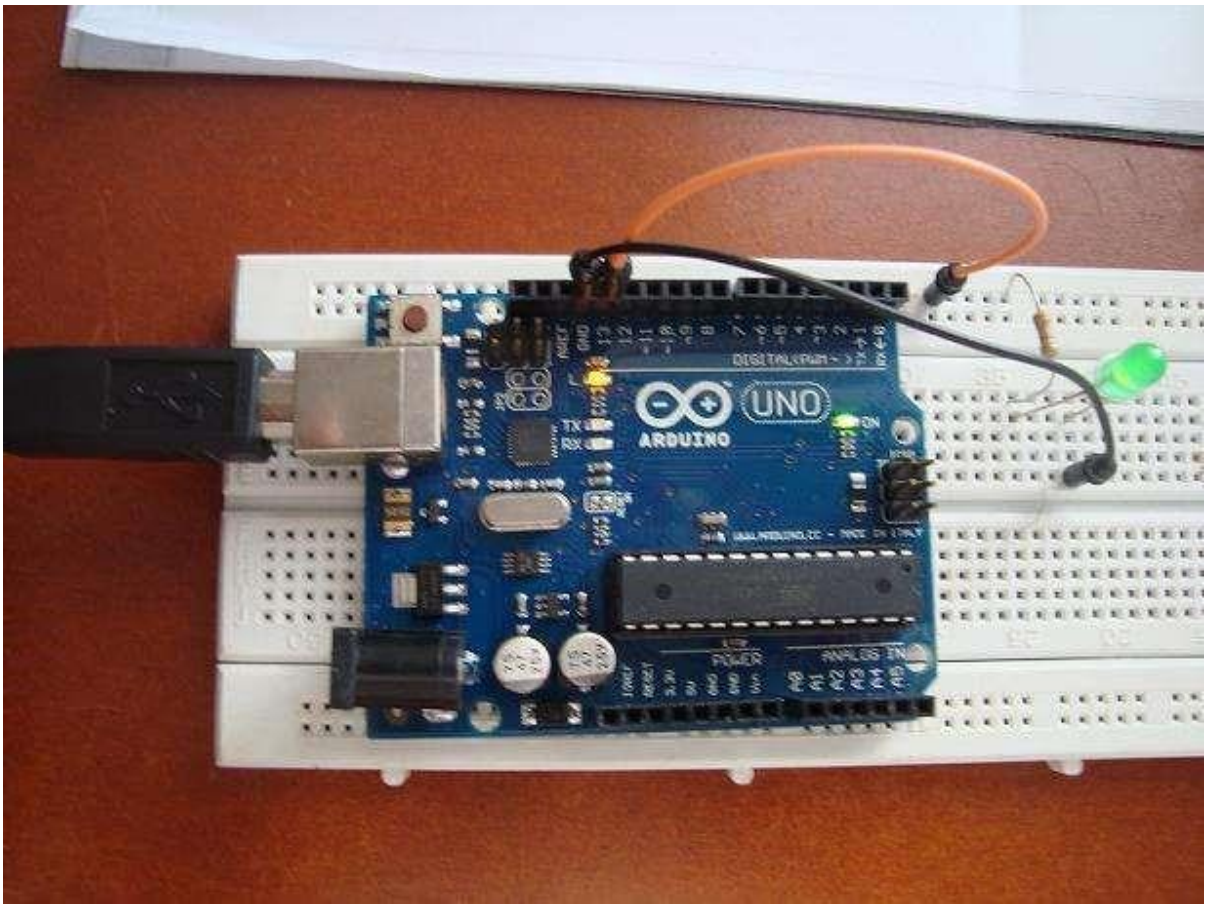
Encender y apagar un LED de tal forma que el parpadeo sea de 1 segundo. El led se coloca en el pin digital 13. Se edita el siguiente programa:



Se verifica haciendo clic en el ícono de verificar para comprobar que en la compilación no hubo errores. Se conecta el Arduino al computador por el cable usb y se hace el

hardware adicional en el Protoboard y su conexión con Arduino como se indica a continuación.

A continuación, se envía el programa al Arduino haciendo clic en el ícono de Subir de la plataforma de edición. Se comprueba que el LED parpadea con un intervalo de 1 segundo.



4.3 ARDUINO E/S - LABORATORIO

El objetivo de la siguiente práctica es adquirir habilidades para programar el Arduino Uno realizando algunas experiencias del manejo de puertos digitales y análogos de este sistema de desarrollo para el microcontrolador ATMEGA328 de ATMEL.

1. FUNCIONES

Funciones de E/S:

pinMode(pin,modo): Configura los pines como entrada o salida. Modo=INPUT (para usar un pin como entrada), OUTPUT (para usar un pin como salida).

digitalRead(pin): Para leer un pin digital. Pin=0,1,2,...,13.

digitalWrite(pin,valor): Escribe un valor LOW o HIGH para 0 o 1 en el pin indicado.

Función de tiempo:

delay(ms): Produce un retardo en milisegundos, ms va de 1 hasta 1000 (1 seg).

Tipos de datos:

int x: Para indicar que x es una variable entera (16 bits)

long x: Para indicar que x es una variable entera larga (32 bits)

array [ ]={valor0, valor1,....} //arreglos

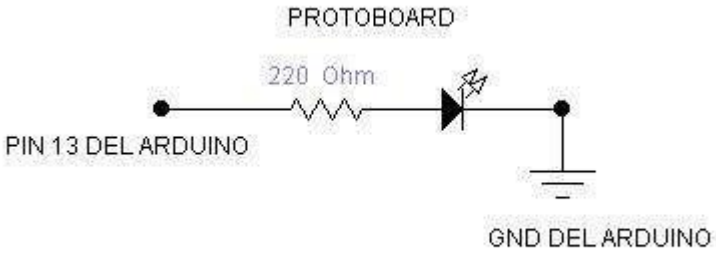
Ej: Array[3] = 10; //asigna a la cuarta posición del índice el valor 10. int Array[5]; //declara un array de enteros con 6 posiciones.

2. EQUIPO Y MATERIAL NECESARIO

- Un computador
- Placa Arduino Uno
- Cable de conexión para usb al arduino
- Protoboard
- Un pulsador
- 4 Leds
- 4 resistencias a 1/4W de 330Ω
- 1 potenciómetro lineal de 10KΩ
- Conectores

EJEMPLO: LED INTERMITENTE

Implementar un led intermitente por intervalos de 1 segundo colocado en el pin digital 13 y tierra GND, protegido con una resistencia de 220Ω.



a. Editar el programa como se indica a continuación.

Intermitente

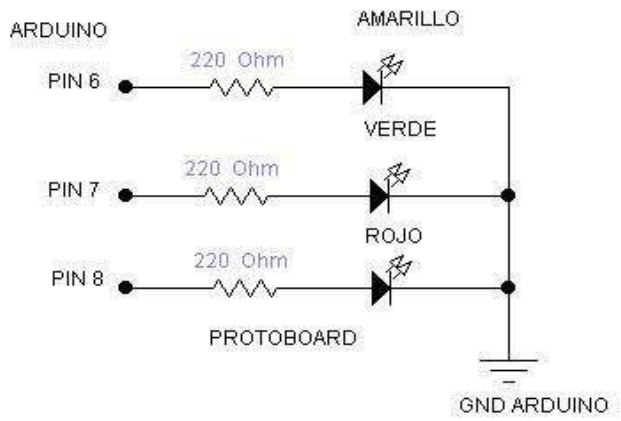
```
/*LED intermitente de un segundo colocado en el pin digital
13 programado como salida y protegido con una resistencia de
220 ohm */

int ledpin=13;  //variable entera ledpin

void setup() {
  pinMode(ledpin,OUTPUT); //pin 13 como salida
}

void loop() {

  digitalWrite(ledpin,HIGH); //escribe en el pin 13 un 1, prendido
  delay(1000);          //retado de 1000 ms= 1sg
  digitalWrite(ledpin,LOW);  //escribe en pin 13 un 0 apagado
  delay(1000);          //retardo de 1 sg

}
```

b. Salvar o guardar el programa como Intermitente.
c. Verificar la compilación del programa para buscar errores si los hay.
d. Realizar el hardware en el protoboard
e. Subir o pasar el programa compilado al Arduino.
f. Comprobar el funcionamiento.

EJEMPLO: SECUENCIADOR DE TRES LEDs

Hacer parpadear tres leds cada 200 ms en forma secuencial colocados en los pines digitales 6, 7 y 8.

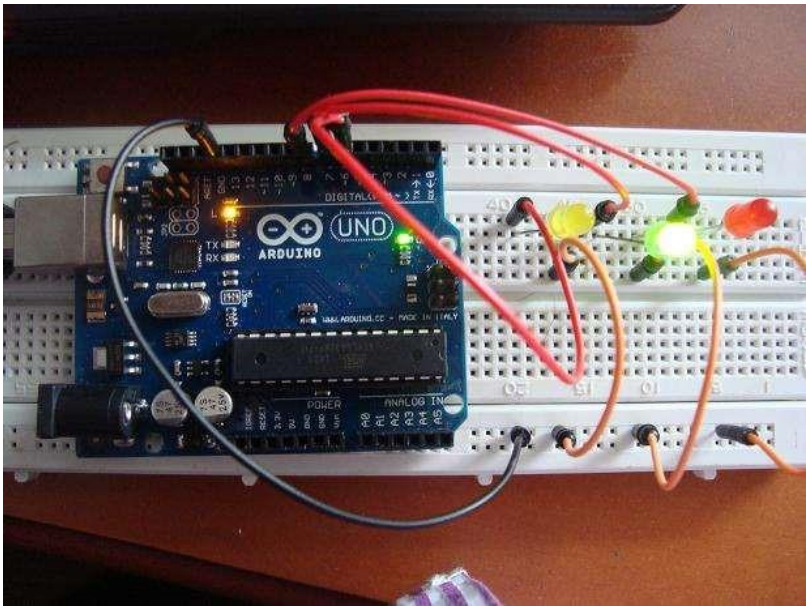
Hardware a implementar en protoboard:



Conecte el Arduino al computador y este al protoboard, edite el siguiente programa para solucionar el problema planteado.

```
void setup() {
  pinMode(6,OUTPUT);  //pin 6 como salida
  pinMode(7,OUTPUT);  //pin 7 como salida
  pinMode(8,OUTPUT);  //pin 8 como salida

}

void loop() {
  digitalWrite(6,HIGH);  //escribe 1 en el pin 6, lo enciende
  delay(500);            //retardo de 200 mseg
  digitalWrite(6,LOW);   //apaga el led en pin 6
  digitalWrite(7,HIGH);  //enciende el led en pin 7
  delay(500);            //reatdo de 200 mseg
  digitalWrite(7,LOW);   //apaga led en pin 7
  digitalWrite(8,HIGH);  //enciende led en pin 8
  delay(500);            //retardo de 200 mseg
  digitalWrite(8,LOW);   //apaga led en pin 8 y se repite secuencia
}
```

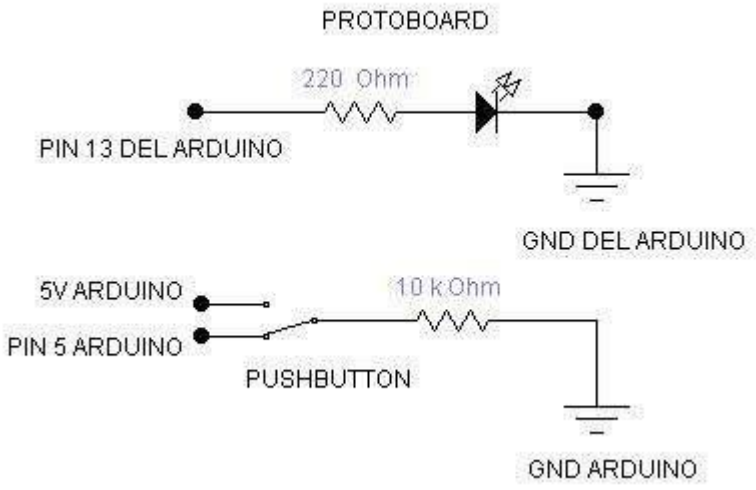
Verifique (Verificar) la compilación, envíe el programa al Arduino (Subir), Guarde el programa (Salvar) con el nombre de secuenciador. Observe el resultado en los leds del protoboard.



EJEMPLO: ALARMA SENCILLA

Leer un pulsador conectado en el pin 5, si está a 5V el LED debe estar apagado, si está a 0V el led parpadea cada 200 ms. El led está conectado al pin 13.

Hardware para implementar en el protoboard:



Editar el siguiente programa y correrlo en el Arduino.

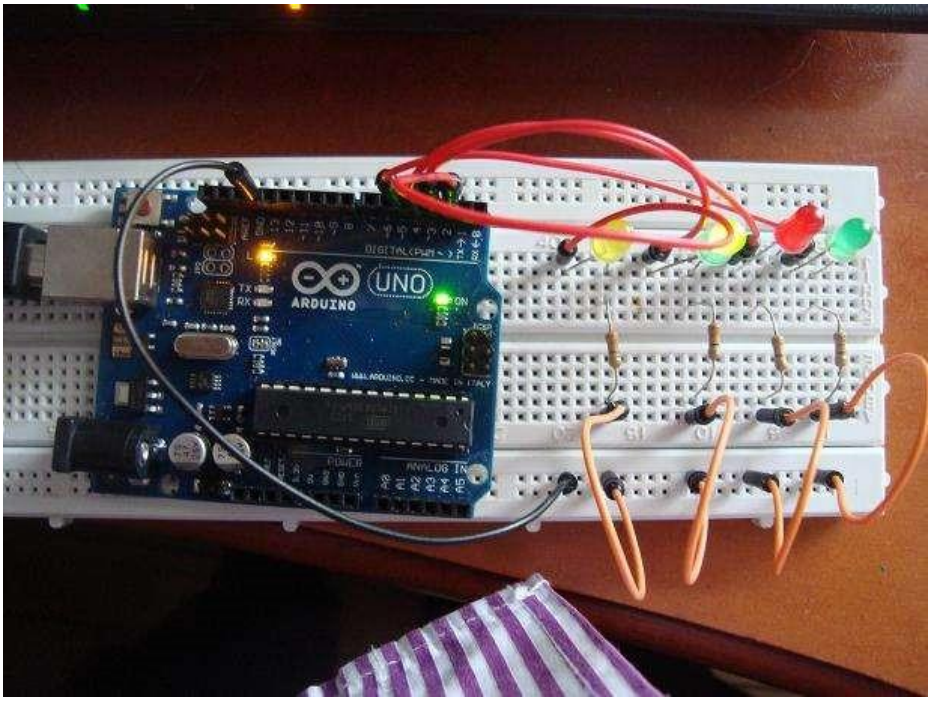
```
/*Alarma sencilla que lee un pulsador y acciona al led. Si está
a 5V el led está apagado y si está a 0V parpadea a 200 ms */

int val=0;     // val es una variable entera

void setup() {
 pinMode(5,INPUT);  //configura pulsador como entrada en pin 5
 pinMode(13,OUTPUT);  //configura pin 13 (salida) led
}



void loop() {
  val=digitalRead(5);  // lee el pin 5
  if(val==HIGH){         // bucle condicionado  val=1 (5V)
    digitalWrite(13,LOW);   //pone pin 13 led bajo , o sea, apaga
  }
  else{                 // de lo contrario
    digitalWrite(13,LOW);  // led parpadea, primero apagado
    delay(200);           // retardo de 200 ms
    digitalWrite(13,HIGH);    //lo pone en alto enciende
    delay(200);       //retardo de 200 ms
  }

}
```

EJEMPLO: COCHE FANTÁSTICO

El siguiente programa consiste en hacer parpadear 4 leds colocados en los pines 2, 3, 4,y 5 cada 100 ns en forma secuencial y luego repetir el parpadeo pero de regreso, o sea, 5,4,3,2. No olvide colocar resistencias de protección en serie con el led de 220 o 330 ohm.

```
/*PROGRAMA COCHE FANTÀSTICO*/
//definición de variables
int array[]={2,3,4,5};
int cont=0;
int timer=100;
//configuración de pines
void setup() {
for (cont=0; cont<4;cont++){
  pinMode(array[cont],OUTPUT);
  }
}

  //programa principal
  void loop() {
  //parpadeo a la derecha
  for(cont=0; cont<4;cont++){
    digitalWrite(array[cont],HIGH);
    delay(timer);
    digitalWrite(array[cont],LOW);
    delay(timer);
  }

 //parpadeo a la izquierda
 for(cont=3; cont>=0;cont--){
   digitalWrite(array[cont],HIGH);
   delay(timer);
   digitalWrite(array[cont],LOW);
   delay(timer);
   }
 }
```

HARDWARE PARA IMPLEMENTAR



4.4 ESTRUCTURAS DE CONTROL DEL ARDUINO

Operadores aritméticos

x++: crecimiento x=x+1; x--: decrecimiento x=x-1

x+=: x=x+y; x-=: x=x-y; x*=: x=x*y; x/=: x=x/y

Operadores de comparación

x==y: x es igual a y; x!=y: x es diferente de y

x<y: x es menor que y; x<=y: x es menor o igual a y

x>y: x es mayor a y; x>=y: x es mayor o igual a y

ESTRUCTURAS DE CONTROL

Estructura if - else:

//si se cumple la condición

if(condicion)

{bloque de funciones

}

//de lo contrario se ejecuta el siguiente bloque

else {bloque de funciones

}

Estructura for

for(inicialización; condición; expresión) {

bloque de funciones

}

4.5 SALIDAS PWM

Los pines (3, 5, 6, 9, 10, y 11). Proporcionan salidas PWM de 8 bit, La modulación por ancho de pulso (PWM) puede ser utilizada en el Arduino mediante las funciones analogWrite, digitalWrite y digitalRead. analogRead(pin):

Lee el valor desde un pin analógico especificado con una resolución de 10 bits. Esta función sólo trabaja en los pines analógicos (A0 a A5). Los valores enteros devueltos están en el rango de 0 a 1023. analogWrite(pin,valor):

Escribe un valor usando modulación por ancho de pulso (PWM en inglés) a un pin de salida marcado como PWM. En los Arduinos con el chip ATmega 328, esta función trabaja en los pines 3, 5, 6, 9, 10 y 11. El valor puede ser especificado como una variable o constante con un valor de 0 a 255. El valor de salida varía de 0 a 5 V según el valor de entrada (de 0 a 255) en función del tiempo de pulso.

EJEMPLO: AJUSTE DE BRILLO DE UN LED CON PMW

Este programa ajusta el brillo de un led colocado en el pin pwm 9. Primero se envía al pin un ajuste pwm de 0 a 255 en forma ascendente y luego en forma descendente de 255 a 0 utilizando la función for.

BrilloLed

```
//AJUSTE DE BRILLO DE UN LED USANDO PWM
//variables y constantes
#define ledpin 9 //led se pone en pin pwm 9
int i;
void setup() {
}


//programa principal
void loop() {
  for(i=0;i<=256;i++){
  analogWrite(ledpin,i);
  delay(20);
}
for(i=255;i>=0;i--){
  analogWrite(ledpin,i);
  delay(20);
}
}
```

CAPÍTULO 5. MÓDULOS DEL MICRO 16F877- TEORÍA

Las interrupciones en un microcontrolador son las encargadas de interrumpir la ejecución de un programa cuando ocurre un suceso interno o externo. En ese momento se ejecuta un salto a la rutina que se ha diseñado para que sea atendida por esa interrupción. Cuando termina de ejecutar la rutina vuelve al programa que estaba ejecutando. Los PIC como el 16F877 tienen 14 fuentes de interrupción que son controladas por el registro INTCON que tiene el micro.

Las interrupciones que se van a tratar en esta unidad será la Externa por el pin RBo, la del Timer0, timer1 y Timer2. Los timers son módulos integrados en el micro que pueden funcionar como temporizadores o contadores. Como es tradicional se presentará su Teoría, Simulaciones, Laboratorios y Evaluación.

5.1 TEMPORIZADORES E INTERRUPCIONES

1. EJEMPLO: INTERRUPCIÓN EXTERNA POR RB0

Para interrumpir el micro de forma externa se utiliza el pin RB0 para provocarla. Esta interrupción obra por cambio de nivel en el pin ya sea de alto a bajo o de bajo a alto. Usa la directiva:

#int_ext

La interrupción se habilita mediante:

enable_interrupts(int_ext) y la global

enable_interrupts(global)

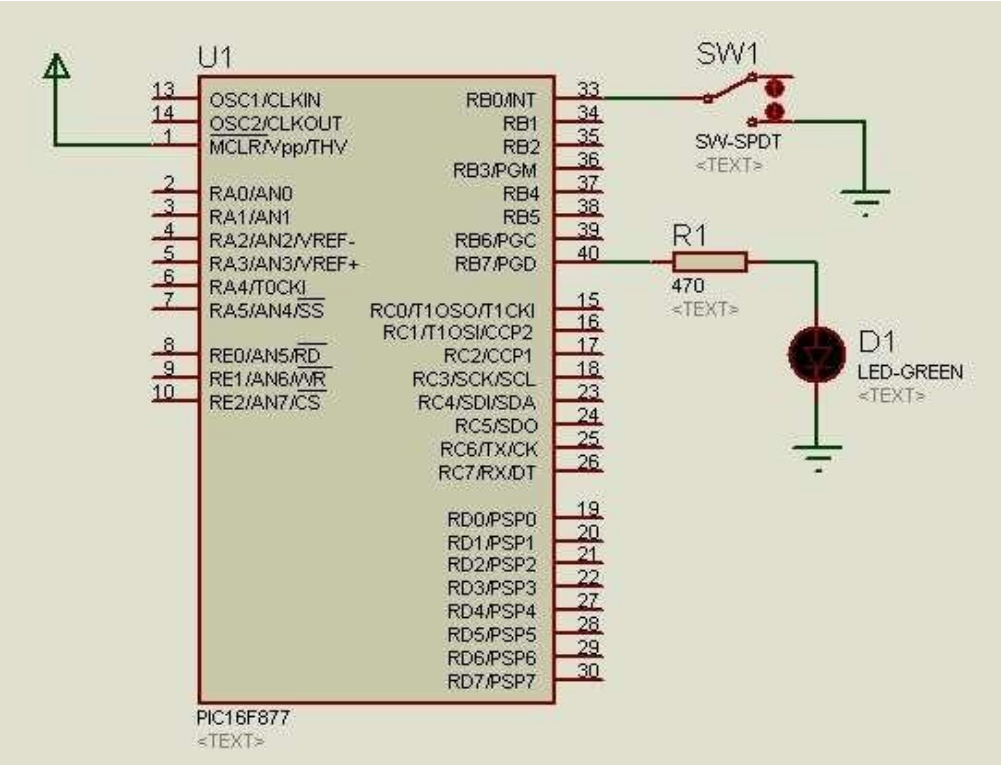
Las instrucciones son:

ext_int_edge(H_TO_L) para interrumpir por flanco de subida o

ext_int_edge(L_TO_H) para interrumpir por flanco de bajada.

A continuación, se presenta un ejemplo para encender un led durante 2 segundos colocado en el pin RB7 cuando hay una petición de interrupción por RB0 al cambiar el nivel de bajo a alto (por flanco de subida).

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES PIC
CEDUVIRT-CENTRO DE EDUCACIÓN VIRTUAL
EJEMPLO DE INTERRUPCIÓN EXTERNA POR PIN RBO
*/

#include<16F877.h>  //incluye todas las características del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use fast_io(B)
#int_ext          // directiva de int externa
// rutina que atiende cuando se le interrumpe
void atender_int (void)
{
output_high(pin_B7);  //pone pin alto enciende led
delay_ms(2000);       //retardo de 2 seg
output_low(pin_B7);   //apaga led
}
```

```
//programa principal para habilitar interrupción externa por pin B0
 void main()
{
port_b_pullups(true);
set_tris_B(0B00000001);    // pin B0 como entrada pin B7 como salida
output_low(pin_B7);        //inicia con led apagado
enable_interrupts(int_ext);  // habilita int externa
ext_int_edge(L_TO_H);        // habilita int por flanco de subida
enable_interrupts(global);   // habilta int global
while(true){                 //espera indefinidamente interrupción
 }
}
```

2. EJEMPLO: INTERRUPCIÓN POR OVERFLOW DEL TIMER0

El Timer0 es un registro del micro que hace parte de la memoria RAM que está en la posición 101H. Es de 8 bits (hasta 255) y funciona como:

- Contador de eventos externos a través del pin RA4, o
- Temporizador haciendo uso del reloj interno del micro a una frecuencia de fosc/4

Se puede preescalar haciendo un divisor de frecuencia programable por 2, 4, 8, 16, 32, 64, 128 o 256.

La directiva a utilizar es:

#int_timer0

Para configurar el Timer0 se utilizan las funciones:

setup_timer_0(modo)

Donde el modo puede ser:

RTCC_INTERNAL, RTCC_EXT_L_TO_H, RTCC_EXT_H_TO_L,

RTCC_DIV2, RTCC_DIV4, RTCC_DIV8,

RTCC_DIV16, RTCC_DIV32, RTCC_DIV64, RTCC_DIV128, RTCC_DIV256

Pueden agruparse distintos modos usando el operador OR (|)

Para cargar (escribir) o leer el timer0 se utilizan las funciones:

set_timer0(valor); // valor hasta 255

get_timer0 ()

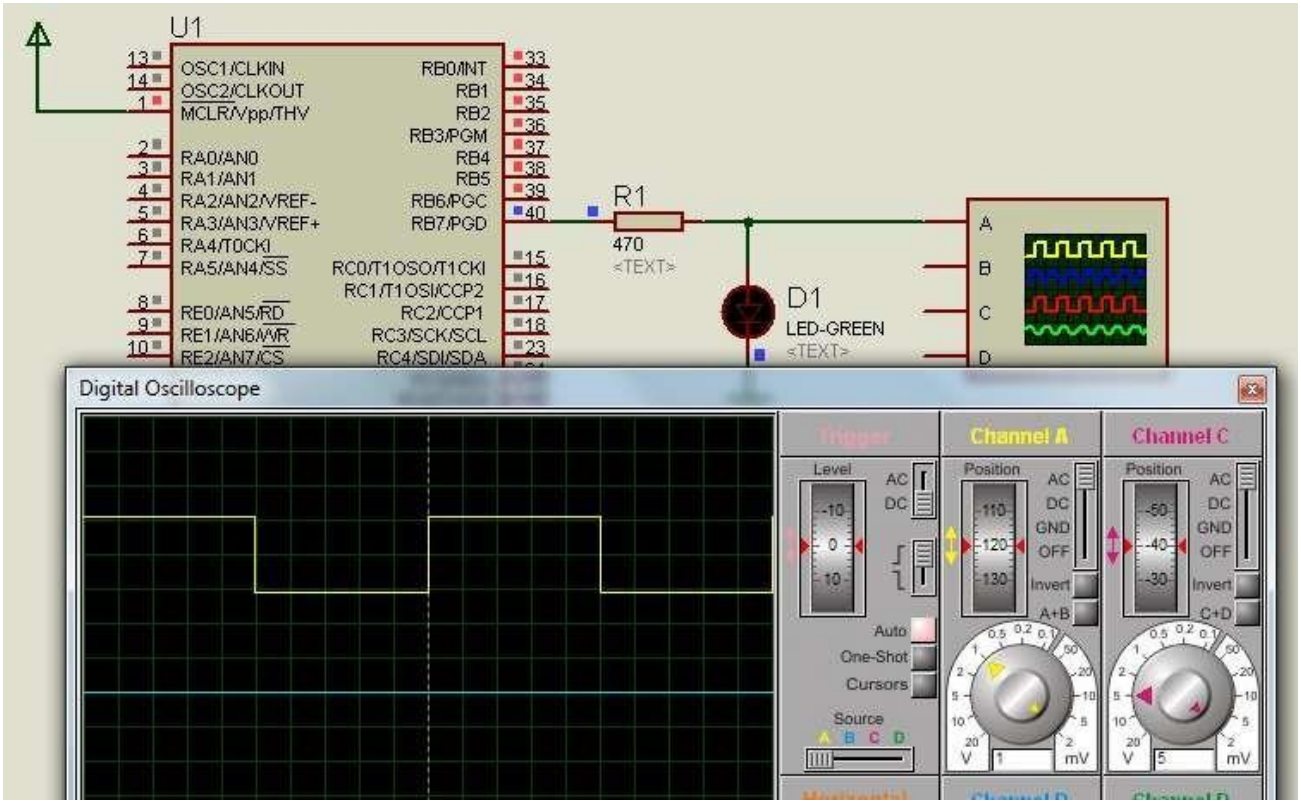
Dependiendo del tiempo de overflow (desborde) T del timer se calcula este valor asi: valor = 256 -T / (prescala*(4 / fosc))

Por ejemplo, si se quiere generar una señal cuadrada de una frecuencia de 1000 Hz, el periodo es de 1000 usg, o sea, un T= 500 usg, con prescala de 2, entonces,

valor = 256 - 500usg / (2*(4 / 4 MHz)) = 6

Es recomendable adicionar unos 20 ciclos más pues lo que necesita el programa principal para llegar a la interrupción. Se debe cargar el Timer0 con un valor igual a 26. El led colocado en el pin RB7 titilará cada 500 usg. Si se lleva paso a paso la simulación del programa se observa que el timer va de 0 a 26 se incrementa sucesivamente hasta llegar a 255, ocurre la interrupción, va a 0 se carga con 26 llega a 255, etc.....

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES PIC
CEDUVIRT-CENTRO DE EDUCACIÓN VIRTUAL
EJEMPLO DE INTERRUPCIÓN POR TIMER0
*/

#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)
#int_timer0      // directiva de int externa


// rutina que atiende cuando se le interrumpe
void atender_int (void)
{
output_toggle(pin_B7);  //pone pin alto enciende led
set_timer0(26);
}

//programa principal para habilitar interrupción externa por pin B0
 void main()
{
port_b_pullups(true);
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_2);
enable_interrupts(INT_TIMER0);
enable_interrupts(global);
while(true){                //espera indefinidamente interrupción
 }
}
```

3. EJEMPLO: INTERRUPCIÓN POR OVERFLOW DEL TIMER1

El Timer1, es otro temporizador/contador conformado por dos registros de 8 bits el TMR1H y el TMR1L, o sea, que se ve como un registro de 16 bits. El registro se incrementa hasta FFFFH ocurre el desbordamiento, vuelve a 0 y luego se carga al valor asignado para seguir incrementándose hasta FFFFH y así sucesivamente. Puede operar como temporizador, contador asíncrono o contador síncrono. El valor de la carga es igual a:

valor= 65536 -T / (prescala*(4 / fosc)), la prescala puede ser de 2, 4, 8
Directiva:
#int_timer1;

Las funciones utilizadas para el módulo TMR1 son las siguientes:
setup_timer_1(modo): Habilita o deshabilita el timer1

modo: TI_DISABLE, T1_INTERNAL, T1_EXTERNAL, T1_EXTERNAL_SYNC, T1_CLK_OUT,
T1_DIV_BY_2, T1_DIV_BY_4, T1_DIV_BY_8
valor= get_timer1 (); //toma el valor del timer1
set_timer1(valor) ; //carga el timer1

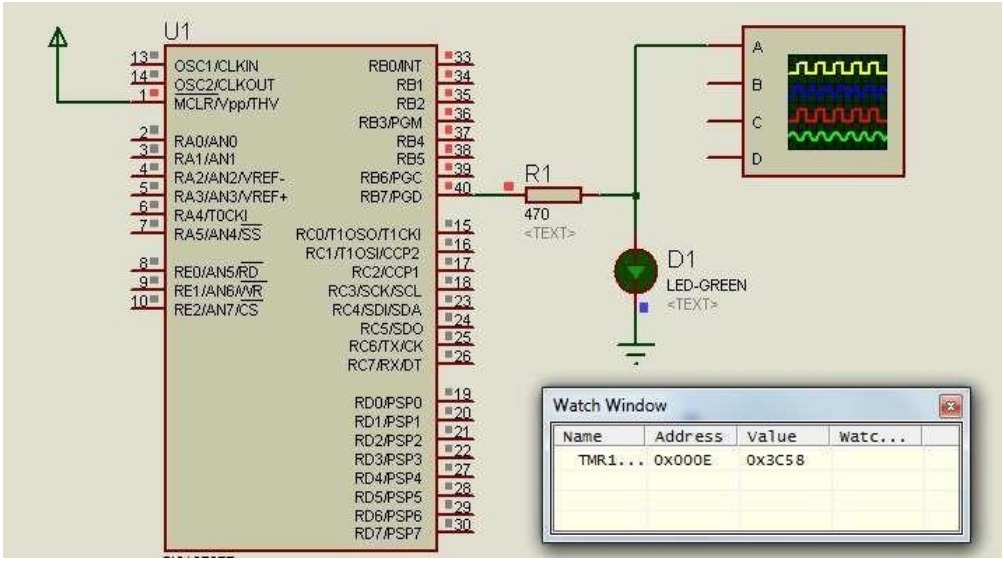
4. EJEMPLO: GENERAR UNA SEÑAL

Generar una señal de 1 segundo. Esto requiere una temporización de 0.5 sg y alternar para generar la señal. Con el Timer0 no se puede hacer porque es de sólo 8 bits, si fosc = 4Mhz, Ts = 0.25 us, como cada incremento del contador se hace cada 4 ciclos de reloj, entonces lo máximo que puede temporizar es de 4*0.5 us*256*prescala. La máxima escala es de 255, o sea, puede llegar hasta 256*255= 65280 us = 65.28 ms y se necesitan 500 ms. Por ello se usa el timer1 que tiene 16 bits, esto es puede llegar la temporización hasta 65536*preescala. La máxima prescala para el Timer1 es de 8, entonces, 65536*8= 524288 us = 514 msg.

Valor a cargar en el Timer1:

Prescala = 8,
valor= 65536 -T / (prescala*(4 / fosc))
valor = 65536 - 5000000us / (8*(4/4Mhz)) = 65536 - 62500
valor = 3036

HARDWARE CON PROTEUS



SOFTWARE CON CCS

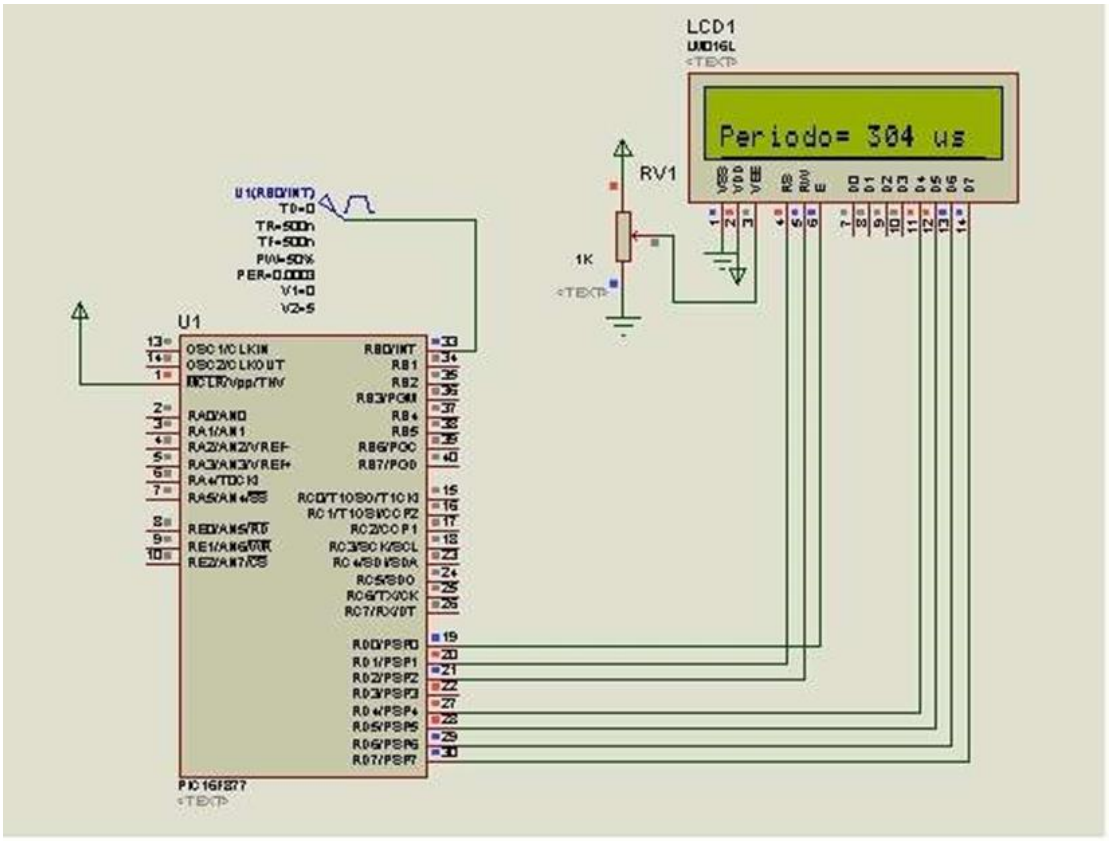
```
#include<16F877.h>  //incluye todas las características del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)

int1 cont=0;
#int_timer1      // directiva de interrupción por timer1


// rutina que atiende cuando se le interrumpe
void atender_int (void)
{
if (cont==1)
output_toggle(pin_B7);  //pone pin alto enciende led
set_timer1(3036);
cont++;      // hacer el retardo dos veces
}

//programa principal
 void main()
{
port_b_pullups(true);
setup_timer_1(T1_INTERNAL|T1_DIV_BY_8);
set_timer1(3036);
enable_interrupts(INT_TIMER1);
enable_interrupts(global);
while(true){                //espera indefinidamente interrupción
}
}
```

5.2 MÓDULOS DEL PIC - SIMULACIÓN

1. SIMULACIÓN 1. INTERRUPCIÓN POR RBO Y RB1

En el siguiente ejemplo se requiere leer el periodo de una señal cuadrada que se inyecta por la entrada RB0 con el fin de utilizar la interrupción externa por este pin. La señal es de un periodo de 300 us. La interrupción externa inicialmente lle el valor del registro timer1 por el flanco de subida y vuelve a leer este registro cuando interrumpe por el flanco de bajada. El display se coloca en puerto D.

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES PIC
CEDUVIRT-CENTRO DE EDUCACIÓN VIRTUAL
EJEMPLO DE INTERRUPCIÓN POR RB0 y TIMER1
MEDIR EL PERIODO DE UNA SEÑAL CUADRADA Y DESPLEGARLO EN UN LCD
*/

#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)
#include<lcd.c>  // teclado en puerto D
```

```
//definición de variables
int16 valorL;
float T;
int1 pulso=0;
int1 nivel=0;

#int_ext
void atender_int_ext()    // interrupción externa por RB0
{
if (nivel==0)
 {set_timer1(0);            //pone timer1 en 0
  ext_int_edge(H_TO_L); //configura flanco de bajada
  nivel=1;
 }

 else
 {valorL=get_timer1();    //valor de timer1 en flanco de bajada
 ext_int_edge(L_TO_H);   //configura flanco de subida
 nivel=0;
 if(pulso==0)              //fin de pulso
  {pulso=1;
  }
 }
}


//programa principal
void main()
{
lcd_init();                      //inicializa display
setup_timer_1(T1_INTERNAL |T1_DIV_BY_1); //configura timer1
ext_int_edge(L_TO_H);        //configura flanco de subida
nivel=0;
enable_interrupts(int_ext);    //habilita interrupción externa
enable_interrupts(global);     //habilitación general


do
{
 if(pulso==1)
 {T=valorL*2*4*0.25;           //valor del timer1 en usg fosc=4Mhz
  printf(lcd_putc,"\nPeriodo=%4.0g us",T);  //despliega ancho del pulso
 }
 pulso=0;
}

while(true);{                 //espera indefinidamente interrupción
  }
 }
```

```
 else
 {valorL=get_timer1();     //valor de timer1 en flanco de bajada
 ext_int_edge(L_TO_H);    //configura flanco de subida
 nivel=0;
 if(pulso==0)               //fin de pulso
  {pulso=1;
  }
 }
}

//programa principal
void main()
{
lcd_init();                      //inicializa display
setup_timer_1(T1_INTERNAL |T1_DIV_BY_1); //configura timer1
ext_int_edge(L_TO_H);         //configura flanco de subida
nivel=0;
enable_interrupts(int_ext);    //habilita interrupción externa
enable_interrupts(global);     //habilitación general


do
{
 if(pulso==1)
 {T=valorL*2*4*0.25;            //valor del timer1 en usg fosc=4Mhz
  printf(lcd_putc,"\nPeriodo=%4.0g us",T);  //despliega ancho del pulso
 }
 pulso=0;
}

while(true);{                 //espera indefinidamente interrupción
  }
 }
```

2. SIMULACIÓN: INTERRUPCIÓN POR OVERFLOW DEL TIMER2

El Timer2, es un temporizador de 8 bits que puede tener preescalar por 1,4, 16 o postescala de 1, 2,..hasta 16. El tiempo de desbordamiento se calcula así:

valor=T / (prescala*postescala*(4 / fosc)) - 1

Directiva:

#int_timer2

Las funciones utilizadas para el módulo TMR2 son las siguientes:

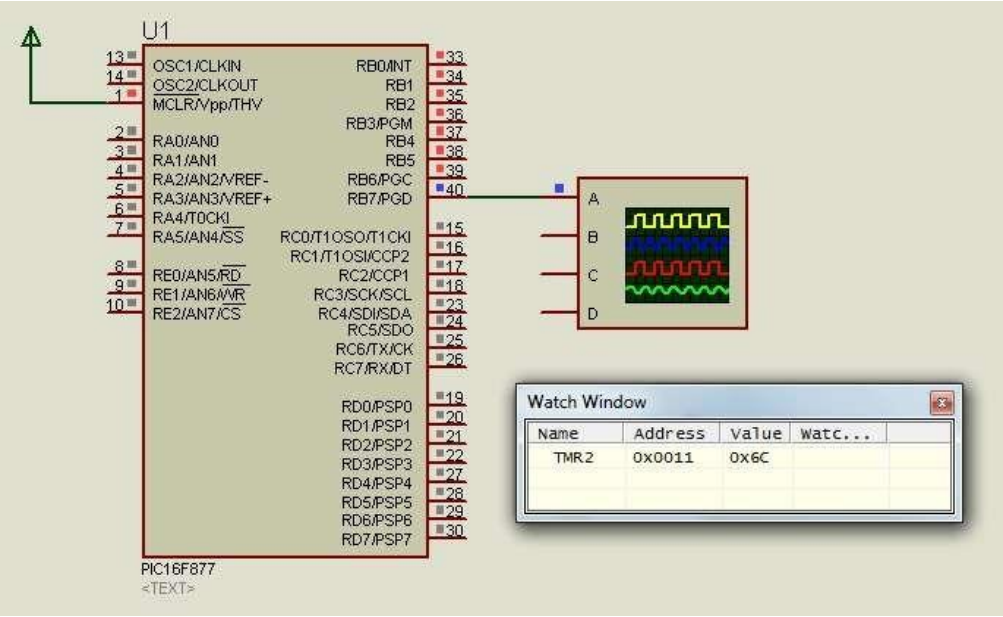
setup_timer_2(modo, periodo, postescala): Habilita o desahabilta el timer2
modo: T2_DISABLE, T2_DIV_BY1, T1_DIV_BY_4, T1_DIV_BY_16
periodo: Es un valor entre 0 y 255
postescala: Es un número de 1 a 16 que determina cuantos desbordamientos antes de una interrupción (1 vez, 2 veces, etc)
valor= get_timer2 ();
set_timer2(valor);

3. EJEMPLO: GENERAR SEÑAL CUADRADA

Generar una señal cuadrada de 1 Khz utilizando la interrupción del Timer2. T = 0.5 ms = 500 us.

Prescala = 4, posescala = 1, fosc = 4 Mhz
valor=T / (prescala*postescala*(4 / fosc)) - 1,
valor = carga de Timer2
valor = 500/(4*1*(4/4)) -1 = 124
Se debe programar el Timer2 así:
setup_timer_2(T2_DIV_BY_4,124,1);

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES PIC
CEDUVIRT-CENTRO DE EDUCACIÓN VIRTUAL
EJEMPLO DE INTERRUPCIÓN POR TIMER2
GENERAR UNA SEÑAL DE 1 Khz
*/

#include<16F877.h>  //incluye todas las características del 16F877
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#use standard_io(B)

#int_timer2      // directiva de interrupción por timer2

// rutina que atiende cuando se le interrumpe
void atender_int (void)
{
 output_toggle(pin_B7);  //pone pin alto enciende led
 set_timer2(10);
 }



//programa principal
 void main()
{
port_b_pullups(true);
setup_timer_2(T2_DIV_BY_4,124,1);   //prescala=4, periodo=124, postescala=1
enable_interrupts(INT_TIMER2);
enable_interrupts(global);
while(true){                 //espera indefinidamente interrupción
}
```

CAPÍTULO 6. MÓDULOS - ARDUINO

6.1 MEMORIA EEPROM

La board del microcontrolador de Arduino tiene una memoria EEPROM para guardar datos cuando la board es apagada. Su librería se usa para leer y escribir en ella. El Arduino Uno almacena 1024 bytes (1 KByte) en el Atmega328.

FUNCIONES

EEPROM.read(dirección);

Lee un byte de la eeprom. Las locaciones que nunca han sido escritas tienen un valor de 255 (todas 1). Donde la dirección es la locación a leer, empieza de 0. Retorna el valor almacenado en esa locación (byte).

EEPROM.write(dirección,valor);

Escribe un byte a la eeprom. La dirección es la locación donde se va a escribir, arrnacando de 0. El valor a escribir va de 0 a 255 (byte).

EEPROM.update(dirección,valor);

Escribe un byte a la eeprom solamente si difiere de uno que ya se ha guardado en la misma dirección.

EEPROM.get(dirección,valor);

Lee cualquier tipo de dato u objeto de la eeprom. El dato a leer, puede ser de tipo primitivo (ej float) o una estructura. EEPROM.put(dirección,valor);

Escribe cualquier tipo de dato u objeto de la eeprom.

EEPROM [dirección];

Permite usar el identificador EEPROM como un arreglo. Las celdas de la eeprom pueden ser leídas o escritas directamente usando este método.

EJEMPLO 1. BORRAR EEPROM

Este ejemplo borrará toda la memoria eeprom del microcontrolador del Arduino Uno que tiene un almacenamiento de 1Kbyte. Se requiere cargar la librería <EEPROM.h>.

BorrarEEPROM

```
#include <EEPROM.h>

void setup() {
  }

void loop() {
 for(int i=0;i<1023;i++){
    EEPROM.write(i,0);
    }
}
```

EJEMPLO 2. GUARDAR CLAVE EN EEPROM

Utilizando teclado matricial y display lcd se guarda una clave en la memoria eepron del arduino.

```
#include <Keypad.h>
#include <LiquidCrystal.h>
#include <EEPROM.h>


int colum=0;
int fila=0;
int dir=0;
byte lect;
int tecla;
int clave=0;

//configuración del lcd
LiquidCrystal lcd(8,9,10,11,12,13); //pin para rs,e,d4,d5,d6,d7
//configuración del teclado
const byte rows=4;
const byte cols=4;
char keys [rows][cols]={
```

```
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowspin[rows]={4,5,6,7};  //pines a las filas f1 f2 f3 f4
byte colspin[cols]={0,1,2,3};  //pines a las columnas c1 c2 c3 c4
Keypad teclado=Keypad(makeKeymap(keys),rowspin,colspin,rows,cols);

void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);  // lcd de 16 col y 2 filas
  lcd.setCursor(0,0);
  lcd.print("CLAVE DE ACCESO");
  lcd.setCursor(0,1);
  lcd.print("USANDO EEPROM");
  delay(1000);


  lcd.setCursor(0,0);
  lcd.print("                ");
  lcd.setCursor(0,1);
  lcd.print("                ");
  delay(1000);
  lect=EEPROM.read(128);
}

void loop() {
  //ingreso por primera vez y no hay clave guardada
  //-----------------------------
  if((lect==0)&(bande==0)){
    lcd.setCursor(0,0);
    lcd.print("NUEVO USUARIO");
    lcd.setCursor(0,1);
    lcd.print("INGRESE CLAVE");
    delay(1000);
```

```
  lcd.setCursor(0,0);
  lcd.print("CLAVE:          ");
  lcd.setCursor(0,1);
  lcd.print("4 DIGITOS       ");
  delay(1000);
  bande =5; //bandera de registro de usuario
  colum=7;
  fila=0;
  dir=0;
}

//-----------------------------------------
//control y escritura en la eeprom
char tecla=teclado.getKey();
if(tecla){   // si se pulsó tecla
  if(bande==5){  //bandera de nuevo usuario
     lcd.setCursor(colum,fila);
     lcd.print(tecla);
     colum++;
     EEPROM.write(dir,tecla);
     dir=dir+1;
     //-------------------------
     if((bande==5)&(colum>12)){
       lcd.setCursor(0,0);
       lcd.print("EXCEDIO CAMPO   ");
       delay(1000);
       lcd.setCursor(0,0);
       lcd.print("CLAVE:          ");
       lcd.setCursor(0,1);
       lcd.print("4 DIGITOS       ");
       colum=7;
       fila=0;
       dir=0;
     }
     //-----------------------------
```

```
//------------------------------
if((bande==5)&(tecla=='#')){
  if(colum<12){
   lcd.setCursor(0,0);
   lcd.print("CLAVE INCOMPLETA");
   delay(1000);
   lcd.setCursor(0,0);
   lcd.print("CLAVE:          ");
   lcd.setCursor(0,1);
   lcd.print("4 DIGITOS       ");
   colum=7;
   fila=0;
   dir=0;
  }
  if((tecla=='#')&(colum==12)){
   lcd.setCursor(0,0);
   lcd.print("USUARIO REGISTRADO");
   lcd.setCursor(0,1);
   lcd.print("                ");
   delay(1000);
   lcd.setCursor(0,0);
   lcd.print("ADMINISTRADOR     ");
   delay(1000);
   EEPROM.write(128,244);
   dir=0;
  }
 }
}
```

EJEMPLO 3. CONTROL CON CLAVE EN EEPROM

Este ejemplo permite un acceso para realizar un control con clave guardada en la eeprom.

```
void loop() {

  if(lect==244){
    lcd.setCursor(0,0);
    lcd.print("INTRODUZCA CLAVE   ");
    lcd.setCursor(0,1);
    lcd.print("CLAVE:             ");
    colum=7;
    fila=1;
    dir=0;
  }
  //-----------------------------
  char tecla=teclado.getKey();
  if(tecla){
    lcd.setCursor(colum,fila);
    lcd.print(tecla);
    colum++;
    lect=EEPROM.read(dir);
    dir=dir+1;
    if(tecla==lect)
      clave=clave+10;
    if(colum>12){
     lcd.setCursor(0,0);
     lcd.print("EXCEDIO CAMPO   ");
     lcd.setCursor(0,1);
     lcd.print("CLAVE:             ");

 delay(1000);
 colum=7;
 fila=1;
 lcd.setCursor(0,0);
 lcd.print("INGRESE CLAVE   ");
 clave=0;
 dir=0;
}
```

```
if(tecla=='#'){
  if(colum<12){
    lcd.setCursor(0,0);
    lcd.print("CLAVE INCOMPLETA");
    lcd.setCursor(0,1);
    lcd.print("                ");
    delay(1000);
    lcd.setCursor(0,0);
    lcd.print("REINGRESE CLAVE ");
    lcd.setCursor(0,1);
    lcd.print("CLAVE:          ");
    delay(1000);
    colum=7;
    fila=1;
    clave=0;
    dir=0;
   }
 //---------------------------------

      //---------------------------------------
      if((tecla=='#')&(colum==12)&(clave==50)){
        lcd.setCursor(0,0);
        lcd.print("CLAVE CORRECTA  ");
        lcd.setCursor(0,1);
        lcd.print("                ");
        delay(1000);
        lcd.setCursor(0,0);
        lcd.print("FELICITACIONES  ");
        delay(1000);
        EEPROM.write(128,244);
        dir=0;
    }
  }
 }
}
```

6.2 INTERRUPCIONES EXTERNAS

Las interrupciones en Arduino te permiten detectar eventos de forma asíncrona con independencia de las líneas de código que se estén ejecutando en ese momento en el microcontrolador.

Las interrupciones se pueden usar:

Para detectar cambios cuando un pulsador en uno de sus pines ha sido presionado.

Para determinar cuándo se ha terminado de gestionar la memoria EEPROM o Flash de tu Arduino.

A modo de despertador del controlador. Esta funcionalidad permite mantener el consumo al mínimo de energía dejando el Arduino en standby hasta que suceda algún evento. Con ello las baterías duran mucho más.

Como complemento ideal a los módulos digitales de sonido, temperatura… que disponen de un potenciómetro que regula cuándo se activa la salida digital. Por ejemplo, realizar un montaje simple en el que ocurra alguna acción cuando se supere un cierto umbral de sonido o una cierta distancia.

Utilizando las interrupciones de Arduino se puede tener el código ejecutando las instrucciones que sean y, sólo cuando esa interrupción se active, el programa se va al código asociado a esa interrupción, lo ejecuta y luego retorna a donde estaba.

Sólo hay unos pocos pines en los que se pueden realizar interrupciones y dependen del modelo de la board o placa. Para el caso del Arduino Uno solamente hay dos interrupciones externas: Interrupción 0 por el pin 2 y la interrupción 1 por el pin 3. Los modos de activación se determinan según las posibilidades siguientes:

LOW: La interrupción se activa cuando el voltaje del pin elegido es bajo, esto es, 0V.

CHANGE: La interrupción se activa cuando el pin cambia de valor, es decir, cuando pasa de LOW a HIGH o de HIGH a LOW.

RISING: Se activa únicamente cuando el valor del pin pasa de LOW a HIGH.

FALLING: Es el caso opuesto al modo RISING. Se activa la interrupción cuando el valor pasa de HIGH a LOW.

FUNCIONES

attachInterrupt(interrupt,ISR,modo)

Especifica un servicio de interrupción (ISR) cuando una llamada a interrupción externa ocurre.

detachInterrupt(interrupt)

Apaga la interrupción en acción.

EJEMPLO 4. ENCENDER UN LED

Se quiere conmutar un LED o cualquier otro dispositivo mediante el uso de interrupciones, de tal forma que cada vez que se presione un pulsador, éste cambie de encendido a apagado y viceversa.

Interrup1

```
/*cambiar de estado un led colocado en pin 8 cada vez que haya
una interrupción externa de tipo interrupt_0 */

int Led = 8;   //led en pin 8
volatile int estado = HIGH;  //volátil porque cambia de valor

//función o rutina que se ejecutará cada vez que se active
//la interrupción.
void cambio_estado(){
  estado = !estado;  //cambio de estado
}

void setup(){
  pinMode(Led, OUTPUT);
  // modo rising e interrupción por pin 2 (Interrupt_0)
  attachInterrupt(0, cambio_estado, RISING);
  delayMicroseconds(2000); // antirrebote
}
void loop(){
  digitalWrite(Led,estado);
  }
```

EJEMPLO 5. ALARMA

Interrupt_Ext_0

```
/*El led verde normalmente está parpadeando
hasta que por medio de un pulsador se llame una interrupción
función alarma, que encenderá el led rojo y apagará el verde.
*/
int red = 8;
int green = 5;
int boton = 2;
void setup()
{
pinMode(red, OUTPUT);
pinMode(green, OUTPUT);
pinMode(boton, INPUT);
//Creamos la interrupcion
attachInterrupt(0, alarma, HIGH);
}
void loop()
{
digitalWrite(green, HIGH);
delay(1000);
digitalWrite(green, LOW);
delay(200);
}


//Codigo de la interrupcion
void alarma()
{
digitalWrite(green, LOW);
while(digitalRead(2) == HIGH)
{
```

```
digitalWrite(red, HIGH);
delay(200);
digitalWrite(red, LOW);
delay(200);
}
}
```

6.3 TIMER'S (TEMPORIZADORES)

Los timer's se usan para medir y controlar tiempos. Un timer puede disparar una interrupción y controlar por ejemplo la alarma de un reloj. Los timers trabajan aumentado un contador variable (counter register) hasta su máximo valor donde llega el overflow (sobreflujo), se resetea y vuelve a cero y en este momento puede solicitar servicio de interrupción (interrupt service routine ISR) para correr un determinado código. Para aumentar el valor del contador en intervalos regulares se requiere una fuente de reloj. El ATmega328 del arduino uno tiene tres timers: Timer0, Timer1 y Timer2.

Timer0

Es de 8 bits, o sea, el counter register tiene un valor máximo de 255. Este timer lo usa el arduino en las funciones delay() y millis().

Timer1

Es de 16 bits llega hasta 65535. se utiliza en la librería <servo.h>

Timer2

Es de 8 bits. Se utiliza en la función tone().

PWM y Timer

El arduino tiene 3 timers y 6 pines de salida PWM. La relación es:

Pins 5 and 6: controlado por timer0

Pins 9 and 10: controlado por timer1

Pins 11 and 3: controlado por timer2

Configuración

Timer1.initialize(microseconds): Esta función debe ser llamada de primero. "microseconds" es el periodo de tiempo.

Timer1.setPeriod(microseconds): Pone un nuevo periodo después que la librería ha sido inicializada.

Run Control

Timer1.start(): Arranca el timer empezando un nuevo periodo.

Timer1.stop(): Para el Timer.

Timer1.restart(): Restaura el Timer, empieza un nuevo periodo.

Timer1.resume(): Resume corriendo un timer parado, no empieza nuevo periodo.

PWM Signal Output

Timer1.pwm(pin, duty): Configura uno de los pines del timer PWM. "duty" es de 0 to 1023, donde 0 pone el pin siempre en LOW y 1023 en HIGH.

Timer1.setPwmDuty(pin, duty): Pone nuevo PWM sin configurar el pin.

Timer1.disablePwm(pin): Para el uso del PWM en un pin. El pin revierte a ser controlado por digitalWrite().

Interrupt Function

Timer1.attachInterrupt(function): Corre una función cada vez que termina el periodo del timer. La función corre como un interrupt.

Timer1.detachInterrupt(): Desahbilita la interrupción.

EJEMPLO 6. PARPADEO DE LED USANDO TIMER1

Timer1

```
#include <TimerOne.h>

/* Este ejemplo usa la interrupción del timer1
para parpadear un led cada 0.15 seg y demostrar como se comparte
una variable entre la interrupción y el programa principal
*/

const int led = 8;  // pin del led
```

```
void setup()
{
  pinMode(led, OUTPUT);
  Timer1.initialize(150000);  // inicializa con 0.15 seg
  Timer1.attachInterrupt(parpadeo); // interrupción pot timer1
  Serial.begin(9600);
}

// La interrupción hace parpadear el led y
//guarda el número de veces que parpadea.
int estado = LOW;
volatile unsigned long cuenta = 0;

void parpadeo()
{
  if (estado == LOW) {
    estado = HIGH;
    cuenta = cuenta + 1;  // incrementa cuenta parpadeo
  } else {
    estado = LOW;
  }
  digitalWrite(led, estado);
}


// El programa principal imprimirá la cuenta del parpadeo
// en el monitor serial
void loop()
{
  unsigned long copiaCuenta;  // holds a copy of the blinkCount

  // Se deshabilita la interrupción mientras se hace la copia.
  noInterrupts();
  copiaCuenta = cuenta;
  interrupts();

  Serial.print("Cuenta = ");
  Serial.println(copiaCuenta);
  delay(100);
}
```

EJEMPLO 7. PWM y TIMER1

Poner la salida PWM en el pin 9 con un 50% de ciclo de trabajo (duty) y forzar una interrupción que conmute el pin digital 10 cada medio segundo (500000 ms).

Timer1PWM §

```
/*
 * Poner la salida PWM en el pin 9 con un 50% de ciclo de trabajo (duty)
 * y forzar una interrupción que conmute el pin digital 10 cada medio segundo (500000 ms).
 */

#include "TimerOne.h"

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);         // Inicializa Timer1 con 0.5 segundos
  Timer1.pwm(9, 512);                // setup para pwm en pin 9, con 50% ciclo de trabajo
  Timer1.attachInterrupt(rutina);    // Pone rutina() como interrupción de timer overflow
}

void rutina()
{
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // Programa principal
}
```

CAPÍTULO 7. CONVERSOR A/D – 16F877- TEORÍA

El convertidor A/D de aproximaciones sucesivas es el más utilizado cuando se requieren velocidades de conversión entre medias y altas del orden de algunos microsegundos a decimas de microsegundos. El proceso de conversión para este tipo de convertidores se basa en la realización de comparaciones sucesivas de manera descendente o ascendente, hasta que se encuentra la combinación que iguala la tensión entregada por el D/A y la de la entrada.

Un conversor de señal analógica a digital es un dispositivo electrónico capaz de convertir una señal analógica de voltaje en una señal digital con un valor binario. La señal analógica, que varía en el tiempo, se conecta a la entrada del conversor y se somete a un muestreo a una velocidad fija, obteniéndose así una señal digital a la salida del mismo. Estos conversores poseen dos señales de entrada llamadas Vref+ y Vref- y determinan el rango en el cual se convertirá una señal de entrada.

El dispositivo establece una relación entre su entrada (señal analógica) y su salida (digital) dependiendo de su resolución. Esta resolución se puede saber, siempre y cuando conozcamos el valor máximo que la entrada de información que utiliza y la cantidad máxima de la salida en dígitos binarios.

La técnica de conversión más empleada es el de aproximaciones sucesivas, apto para aplicaciones que no necesita conversión baja pero a cambio poseen una relación señal a ruido muy elevada, la mayor de todos. En esta unidad se estudiará el conversor utilizado por el PIC16877 y como es tradicional se presentará su Teoría, Simulaciones, Laboratorios y Evaluación.

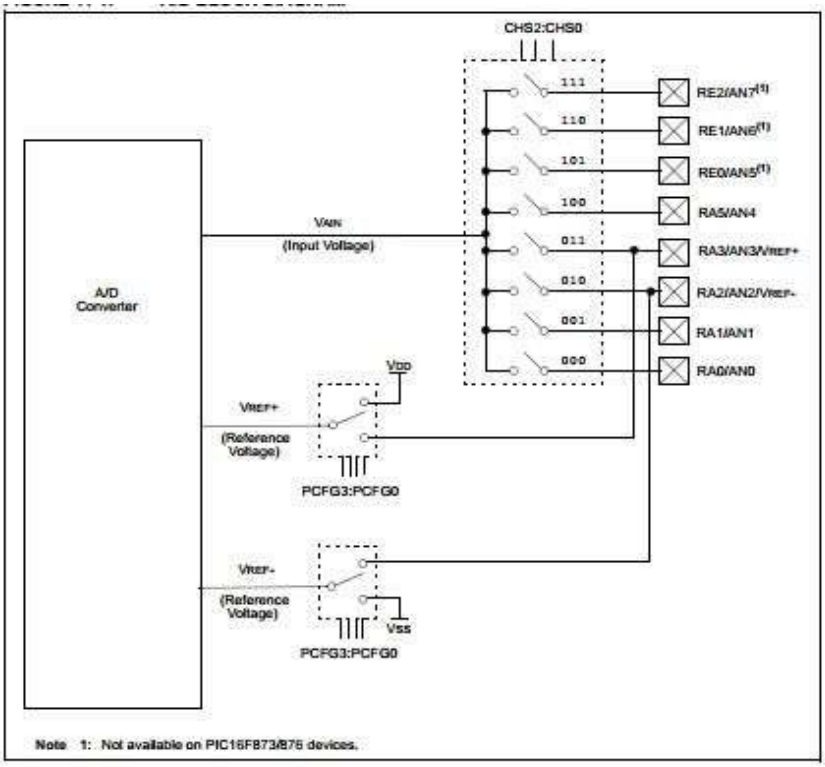
7.1 INTRODUCCIÓN

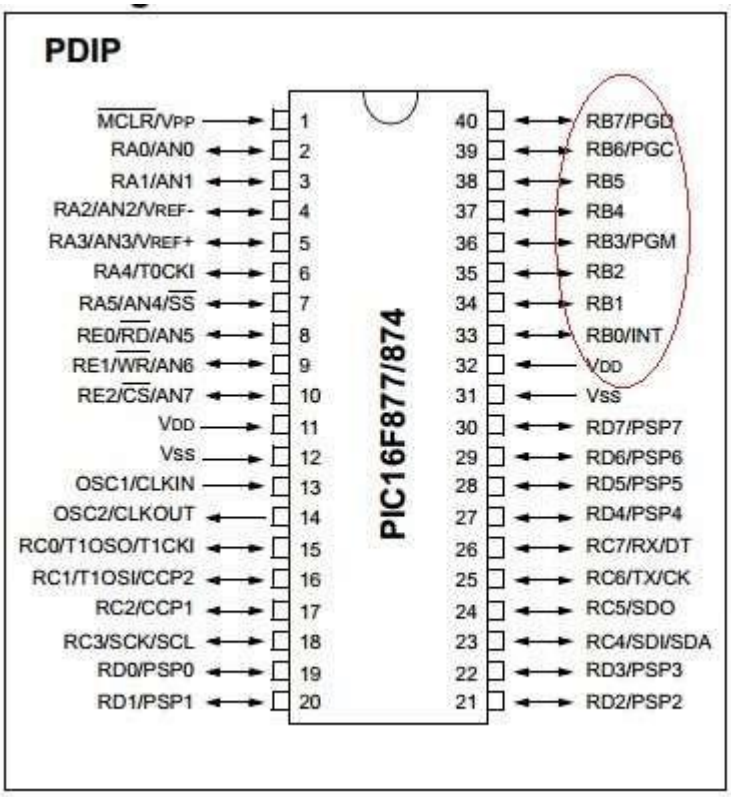
Los convertidores PIC16F87X tienen un convertidor análogo/digital de 10 bits de resolución con 5 canales de entrada en el modelo de 28 pines como el 16F876 y 8 canales en el modelo de 40 pines como el 16F877, que es el que vamos a utilizar. La resolución en la conversión depende del voltaje de referencia de esta forma:

Resolución = (Vref+ - Vref-) / 1024
Si por ejemplo Vref+=5V y Vref- = 0, la resolución es de 4.8mV/bit
Así, 0V = 00 00000000, 5V = 11 11111111
El voltaje diferencial mínimo es de 2V

La señal analógica que entra al canal es muestreada y luego convertida a digital en 10 bits usando la técnica de aproximaciones sucesivas. A continuación, se presenta el

diagrama en bloques del conversor. Canal 0 por pin RA0/AN0, canal1 por pin RA1/AN1, hasta, canal7 por pin RE2/AN7.





7.2 FUNCIONES EN LENGUAJE C

#device adc= 10; // 10 bits

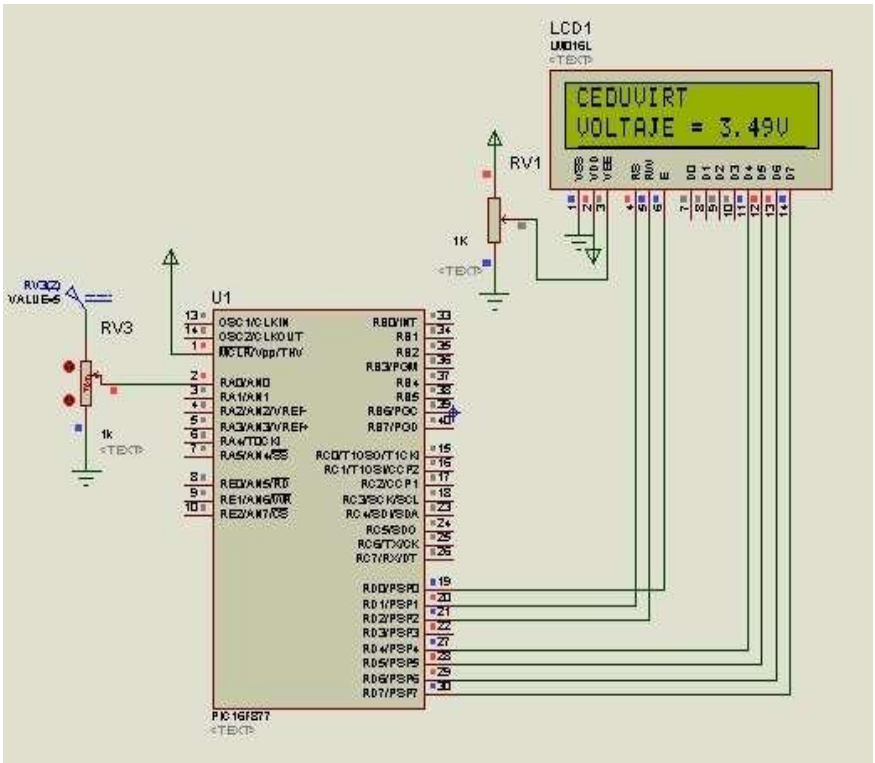
En el compilador CCS las funciones para manejar el conversor AD son las siguientes:

setup_adc(modo); //modo de configuración depende del PIC
modo: ADC_OFF, ADC_CLOCK_INTERNAL, ADC_CLOCK_DIV_2, ADC_CLOCK_DIV_8, ADC_CLOCK_DIV_32
setup_adc_ports(valor); // Seleccionar entradas análogas
valor: ALL_ANALOG, NO_ANALOGS, ANALOG_RA3_REF, RA0_RA1_RA3_ANALOG
setup_adc_channel(canal); // selecciona canal, 0 (AN0),....., 7(AN7)
valor = read_adc([modo]); //lee el resultado
modo: es opcional, puede ser, ADC_START_AND_READ (lee continuamente, es el default), ADC_START_ONLY (arranca la conversión y retorna), ADC_READ_ONLY (lee la última conversión)

EJEMPLO 1. VOLTÍMETRO DIGITAL

En este ejemplo se va a leer un voltaje análogo y desplegarlo en una pantalla LCD.

HARDWARE CON PROTEUS



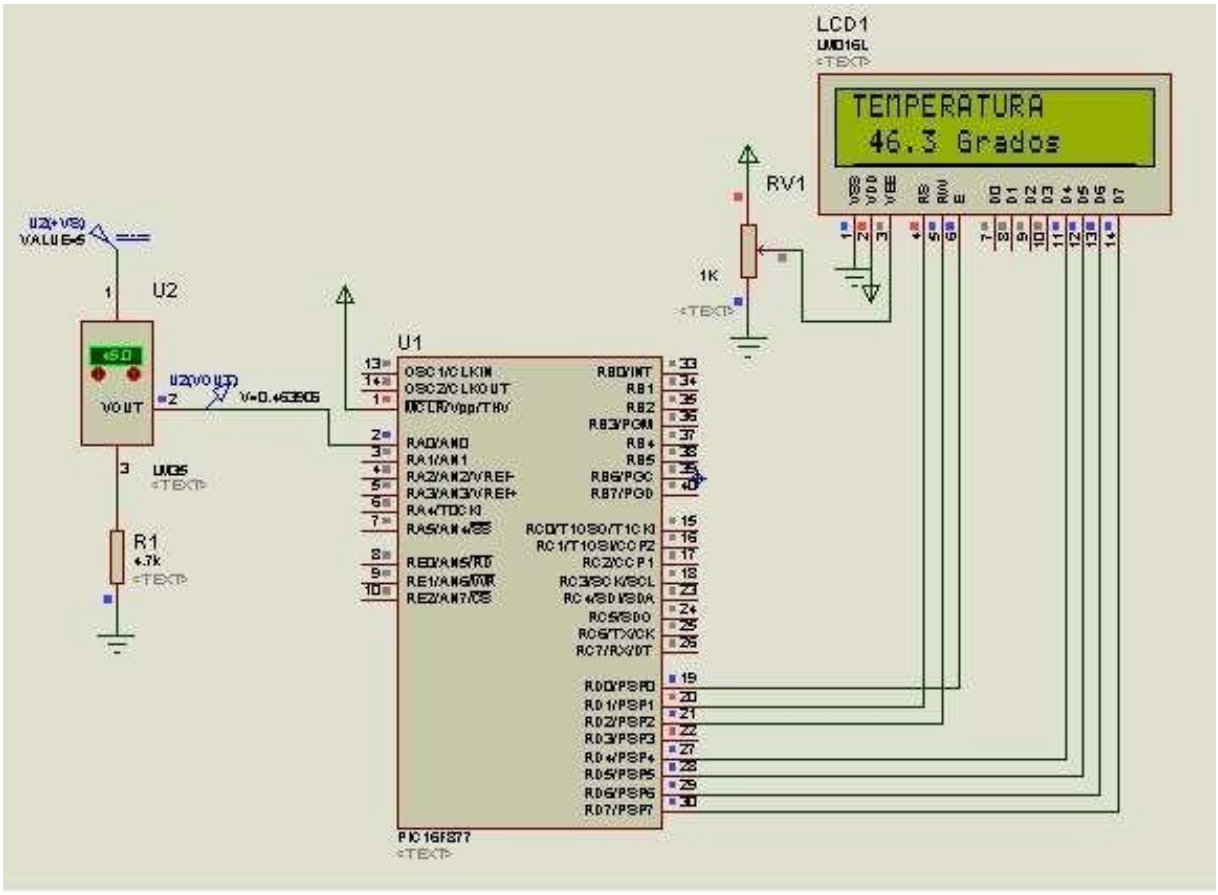
SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES
   LEER UN VOLTAJE ANALOGO Y DESPLEGARLO EN LCD LM016
*/

#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#device adc =10
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
//#use standard_io(A)
#use standard_io(D)

void main ()     // programa principal
{int16 q;
float p;

lcd_init ();
setup_adc_ports(AN0);  //selecciona canal 0
setup_adc(ADC_CLOCK_INTERNAL);  // reloj del conversor el interno

for (;;)  //bucle sin fin
{set_adc_channel(0);   //habilita canal 0
 delay_us(20);
 q=read_adc();     //lee canal 0
 p=5.0*q/1024.0;   //conversión a voltaje
 printf(lcd_putc,"\fCEDUVIRT"); //limpia lcd y despliega CEDUVIRT
 printf(lcd_putc,"\nVOLTAJE = %1.2fV",p);  //flotante con truncado
 delay_ms(100);
 }
}
```

EJEMPLO 2. TERMÓMETRO DIGITAL

El circuito se basa en 2 componentes principales el sensor de temperatura LM35 y el PIC 16F877. El LM35 es un sensor de temperatura con una precisión calibrada de 1ºC y un rango que abarca desde -55º a +150ºC. Lo que quiere decir que por cada 1ºC en la variación de la temperatura, el sensor en su salida obtendrá una variación de 10 mV. Por ejemplo si la temperatura es de -55ºC podemos obtener 550mV y si fuera de 150ºC la salida seria 1500mV. Este voltaje es el que se inserta al PIC 16F877A el cual a través de su conversor A/D mostrara los datos en el LCD.

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES
   LEER UN SENSOR DE TEMP Y DESPLEGARLO EN LCD LM016
*/

#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#device adc =10
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
#use standard_io(D)

void main ()      // programa principal
{int16 q;
float p,T;
```

```
lcd_init ();
setup_adc_ports(AN0);  //selecciona canal 0
setup_adc(ADC_CLOCK_INTERNAL);  // reloj del conversor el interno
set_adc_channel(0);   //habilita canal 0
delay_us(20);
printf(lcd_putc,"\fTEMPERATURA"); //limpia lcd y despliega CEDUVIRT

do {
 q=read_adc();      //lee canal 0
 p=5.0*q/1024.0;    //conversión a voltaje
 T=p*100;           //conversión a grados
 printf(lcd_putc,"\n %2.1f Grados",T);  //flotante truncado
}while (true);
}
```

7.3 CONVERSOR A/D 16F877- SIMULACIÓN

Seguimos realizando simulaciones en Proteus y CCS de ejemplos sobre el conversor A/D del 16F877 de Microchip.

SIMULACIÓN 1. TERMÓMETRO CON NTC

El termistor es un sensor resistivo de temperatura cuyo coeficiente de temperatura resistivo es negativo (NTC). También existen los PTC cuyo coeficiente es positivo. Cuando la temperatura aumenta la resistencia del semiconductor disminuye en los NTC y aumenta en los PTC. Se fabrican a partir de óxidos como el óxido férrico, de níquel o de cobalto. El principal inconveniente es la falta de linealidad.

La NTC se conecta mediante un divisor de tensión como se indica en la figura del hardware utilizado, en donde,

VT = V*RT/ (R1+RT), despejando RT,

RT = VT*R1 / (V-VT)

Para este ejemplo se ha utilizado una R1=10K y V = 5, VT es el voltaje de la NTC

La resistencia para un valor dado de temperatura es igual a:

$$RT = Ro * e^{\beta\left(\frac{1}{T}-\frac{1}{To}\right)}, \qquad despejando\ T$$

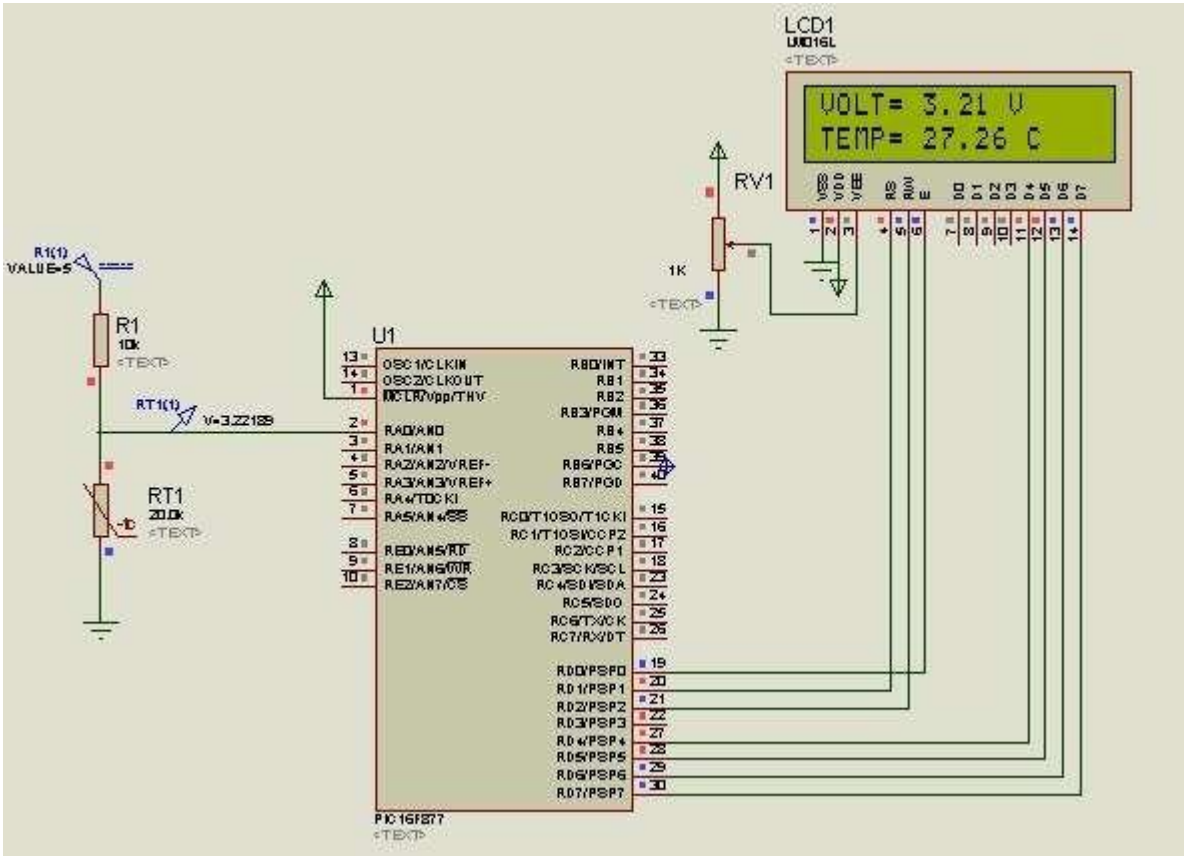
$$T = \frac{1}{\frac{1}{\beta} Ln\frac{RT}{Ro} + \frac{1}{To}}$$

Donde,

- To es la resistencia a una temperatura de referencia,
- RT es la resistencia a la temperatura deseada
- β Temperatura característica del material en grados Kelvin (varía entre 2000 y 5000)
- °K = °C+273.15 (conversión de grados centígrados a kelvin)

En esta simulación se va a utilizar una NTC: NTSA0WB203 que tiene un β = 4050 y una resistencia a 25°C de 20KΩ, el rango de temperatura de operación es entre - 40 a 125°C. Para hacer las operaciones matemáticas se debe utilizar la librería Math.h

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES
   LEER UN SENSOR DE TEMP NTC Y DESPLEGARLO EN LCD LM016
*/

#include<16F877.h>  //incluye todas las caracteristicas del 16F877
#device adc =10
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
#include<math.h>            //libreria matemática
#use standard_io(D)

void main ()     // programa principal
{int16 q;
float VT,TK,TC,R1,V,RT,To,beta,ToK,Ro;

lcd_init ();
setup_adc_ports(AN0);  //selecciona canal 0
setup_adc(ADC_CLOCK_INTERNAL);  // reloj del conversor el interno
set_adc_channel(0);   //habilita canal 0
delay_us(10);

do {
  q=read_adc();     //lee canal 0
  VT=5.0*q/1024.0;   //conversión a voltaje
  R1=10000,  V=5.0;
  RT=VT*R1/(V-VT);
  beta=4050; To=25;
  ToK=To+273.15;
  Ro=20000; //resitencia NTC a 25 oC
  TK=1/((1/beta)*log(RT/Ro)+(1/ToK));
  TC=TK-273.15;
  printf(lcd_putc,"\fVOLT= %2.2f V",VT);  //flotante truncado
  printf(lcd_putc,"\nTEMP= %4.2f C",TC);  //flotante truncado
 }while (true);
 }
```

SIMULACIÓN 2. BARÓMETRO/ALTÍMETRO CON SENSOR DE PRESIÓN

Se va a utilizar el sensor de presión MPX4115 ideal para medir presiones del aire y construir así un barómetro o altímetro. Mide un rango de presiones entre 15 y 115 Kpa (kilopascal) ente 0 y 80°C. Las características las puede encontrar en el siguiente enlace, http://pdf.datasheetcatalog.net/datasheet2/3/07j1jyxe8uwtfocf2owos7ql90fy.pdf El voltaje de salida del sensor se obtiene de la siguiente ecuación:

Vo = Vs*(0.009*P - 0.095) ± (error de presión*factor de temperatura*0.009*Vs) P es la presión en Kpa y Vs es la alimentación del sensor (1Kpa=0.145psi) entre 0 y 80°C el error de presión=1.5Kpa y el factor de temp =1, Vs= 5V

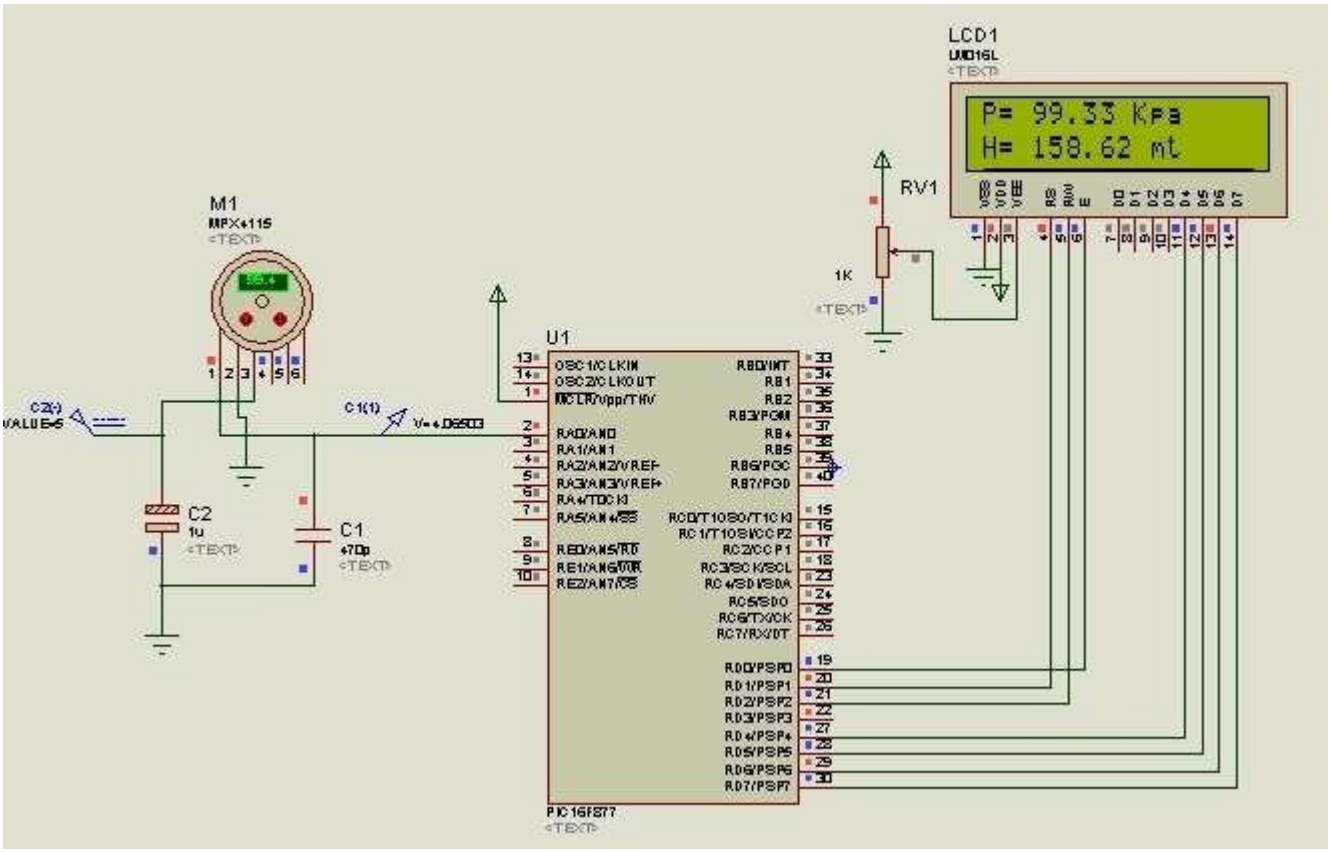
E=error de presión*factor de temperatura*0.009*Vs

Despejando P,

P= (Vo + 0.475-E) / 0.045

La altura sobre el nivel del mar se encuentra con la ecuación, h= - 7990.65*ln(P/Po), Po=101.325 Kpa = 1 atm (nivel del mar)

HARDWARE CON PROTEUS



SOFTWARE CON CCS

```
/* CURSO DE MICROCONTROLADORES
   ALTÍMETRO UTILIZANDO TRANSDUCTOR DE PRESIÓN MPX4115
*/

#include<16F877.h>  //incluye todas las características del 16F877
#device adc =10
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)  //pone el reloj del micro a 4MHz
#include<lcd.c>
#include<math.h>            //libreria matemática
#use standard_io(D)

void main ()     // programa principal
{int16 q;
float Vo,P,Vs,h,Po;
float errorP,factorT,E;
lcd_init ();
setup_adc_ports(AN0);  //selecciona canal 0
setup_adc(ADC_CLOCK_INTERNAL);  // reloj del conversor el interno
set_adc_channel(0);   //habilita canal 0
delay_us(10);

do {
 q=read_adc();     //lee canal 0
 Vo=5.0*q/1024.0;   //conversión a voltaje
 errorP=1.5; factorT=1; Vs=5.0;
 E=errorP*factorT*0.009*Vs;  //tolerancia en la medición 0 a 85oC
 P=(Vo+0.475-E)/0.045;  //presión que lee el transductor en Kpa
 Po=101.325;  //presión a nivel del mar en Kpa
 h=-7990.65*log(P/Po);  //altura según presión en metros
 printf(lcd_putc,"\fP= %2.2f Kpa",P);  //flotante truncado
 printf(lcd_putc,"\nH= %4.2f mt",h);  //flotante truncado
}while (true);
}
```

7.4 CONVESOR A/D – ARDUINO- LABORATORIO

El convertidor Análogo-Digital (ADC por sus siglas en inglés) es un dispositivo que toma una señal análoga (corriente, voltaje, temperatura, presión, etc.), cuantifica la señal y le asigna un valor que se muestra a su salida en formato digital. En general hay tres cosas que nos interesa saber sobre un ADC: La resolución (número de bits), el tiempo de conversión y el rango de trabajo.

La conversión de una señal análoga a su equivalente valor digital no es instantánea, esto significa que el convertidor tarda algún tiempo (usualmente muy pequeño) en realizar el proceso de conversión. Esto limita el número de conversiones que podemos realizar en

una unidad de tiempo, a este número de conversiones que puede realizar un ADC usualmente se le conoce como frecuencia de muestreo.

Una forma simple para elegir un ADC es que la frecuencia de muestreo sea "al menos" el doble de la frecuencia de la señal que queremos convertir (Teorema de Shannon). Por ejemplo si queremos medir una señal que oscila a 60Hz lo menos que necesitaremos es un ADC que logre tomar 120 muestras por segundo.

La tercera característica, pero no la menos importante, que debemos tomar en consideración al elegir un ADC es el rango de voltajes/corrientes que acepta en su entrada, este parámetro es esencial para evitar "quemar" (literalmente) nuestro micro.

La única forma de saber el rango del ADC es revisando su hoja técnica, usualmente encontraremos el rango de valores de voltaje y de corriente aceptados en la entrada como también las tolerancias aceptadas. En el caso de un Arduino Uno, el valor de 0 voltios analógico es expresado en digital como B0000000000 (0) y el valor de 5V analógico es expresado en digital como B1111111111 (1023).Por lo tanto todo valor analógico intermedio es expresado con un valor entre 0 y 1023, es decir, sumo 1 en binario cada 4,883 mV. Arduino Uno tiene una resolución de 10 bits, es decir, unos valores entre 0 y 1023.

Los pines análogos del Arduino para leerlos o escribirles (no necesitan ser configurados como los digitales).

Funciones del conversor:

analogRead(pin): Lee el pin análogo. Pin=A0,A1,A2,A3,A4,A5. devuelve un entero entre 0 y 1023 (10 bits).

analogWrite(pin,valor): Escribe el valor en el pin digital PWM (3,5,6,9,10,11) de 0 (0V) a 255 (5V). val=analogRead(0); //lee la entrada análoga AN0 val=val/4; //convierte 0-1023 a 0-255 analogWrite(10,val); //salida análoga de valor val al pin PMW 10

analogReference: Configura la tensión de referencia para la entrada analógica. Las opciones son:

DEFAULT (5V o 3.3V), INTERNO(Un led de referencia de 1.1V), EXTERNO (Tensión aplicada al pin AREF de 0 a 5V)

EQUIPO Y MATERIAL NECESARIO

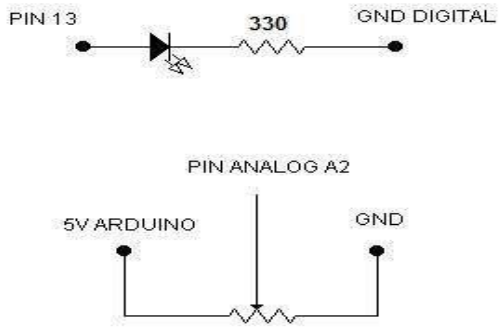
- Un computador
- Placa Arduino Uno
- Cable de conexión para usb al arduino
- Protoboard

- Un pulsador
- 4 LEDs
- 4 resistencias a 1/4W de: 330Ω
- 1 potenciómetro lineal de 10KΩ
- Un display de 7 segmentos cátodo común
- Un display LCD 16x2
- Un sensor de temperatura LM35
- Conectores

1. LEER UNA SEÑAL ANALÓGICA

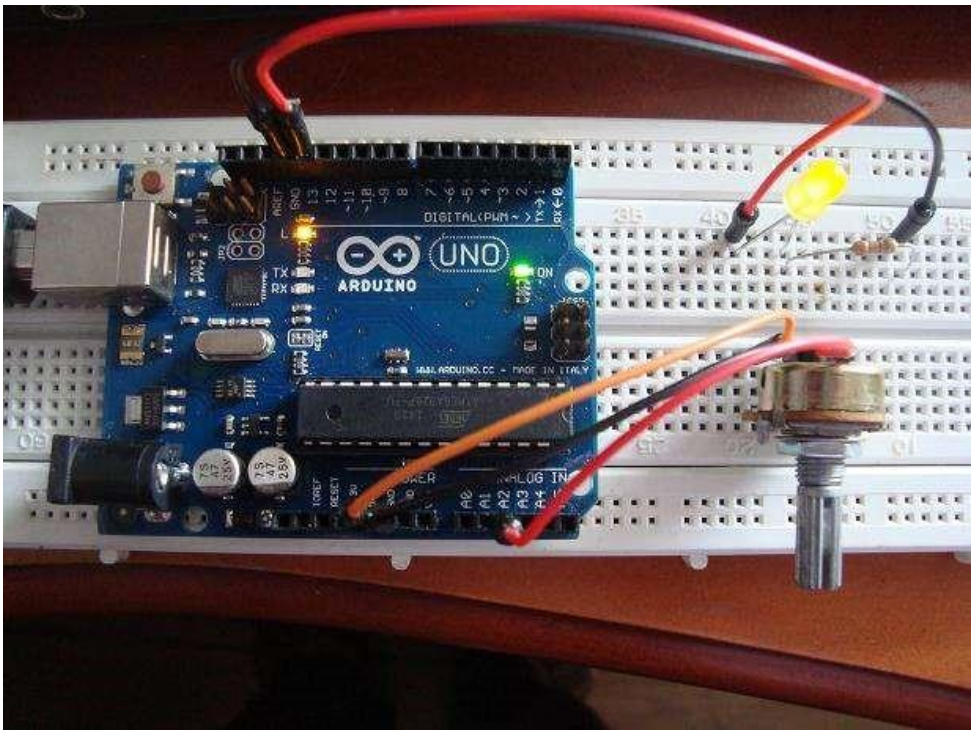
Hacer parpadear un led conectado al pin 13 digital. El tiempo de parpadeo dependerá de un potenciómetro conectado al pin analógico A2.

HARDWARE EN PROTOBOARD:



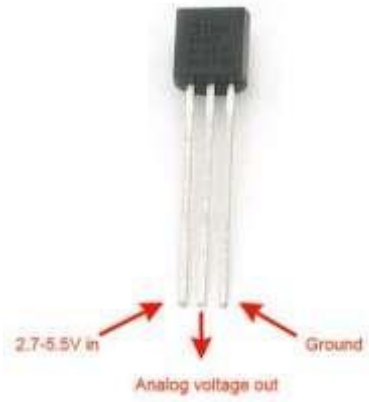
Edite el siguiente programa para realizar el problema propuesto:

```
//PROGRAMA QUE LEE UNA SEÑAL ANÁLOGA
//Variables
int val=0;
//configuración de pines
void setup() {
  pinMode(13,OUTPUT);  //pin 13 como salida para el led
}
//programa principal
void loop() {
 val=analogRead(2);  //valor del pin 2 lo coloca en val
 digitalWrite(13,HIGH);  //pin 13 en alto, enciende led
 delay(val);    //retardo según valor de val (poteniómetro)
 digitalWrite(13,LOW);   //apaga el led
 delay(val);    //retardo igual a val
}
```



2. CONTROL DE TEMPERATURA CON LM35

Usando un LM35 (sensor de temperatura) conectado en el pin análogo A0, realizar un control de temperatura de tal forma que cuando la temperatura sea mayor de 35 grados centígrados un led se conectado en el pin digital 4 se encienda y en caso contrario permanezca apagado.



El LM35 es un sensor de temperatura, esto es, que convierte la temperatura en un voltaje. Para este sensor la sensibilidad es de 10mV por cada grado centígrado. El rango de trabajo es de -55ºC a 150ºC

ControlTemp

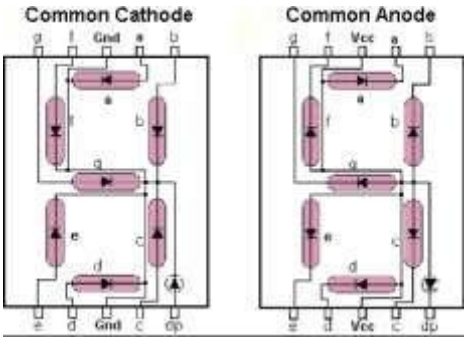
```
/*CONTROL DE TEMPERATURA CON LM35
USO DEL CONVERSOR A/D DE 10 BITS -->1024, EL LED SE PONE EN PIN
4 DIGITAL Y EL LM35 EN PIN ANALOGO A0, AL APSAR TEMP DE 35
GRADOS EL LED SE ENCIENDE DE LO CONTRARIO ESTÁ APAGADO*/

//variables
int sensor=0;  //sensor conectado a pin A0
int ledpin=4;  //led conectado a pin 4
long temp;
long valorsensor;
//configuración pin digital 4
void setup() {
  pinMode(ledpin,OUTPUT);
}
```

```
//programa principal
void loop() {
//convertir voltios a mV,  5V-->1023 (10 bits)
valorsensor=analogRead(sensor)*(5000/1023);
temp=valorsensor/10;  //10mV/grado centígrado
if(temp>=35
){
  digitalWrite(ledpin,HIGH);
}
else{
  digitalWrite(ledpin,LOW);
}

}
```

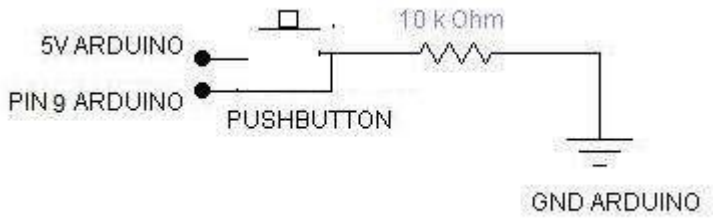
3. CONTADOR DE PULSOS CON DISPLAY DE 7 SEG

El siguiente ejemplo cuenta los pulsos de 0 a 9 en un display de 7 segmentos colocado en los pines digitales del 2 al 8 como salidas cada vez que se presiona un pulsador colocado en el pin 9 como entrada.

CONFIGURACIÓN DEL DISPLAY



CONEXIÓN DEL PULSADOR



PROGRAMA

Contador7seg

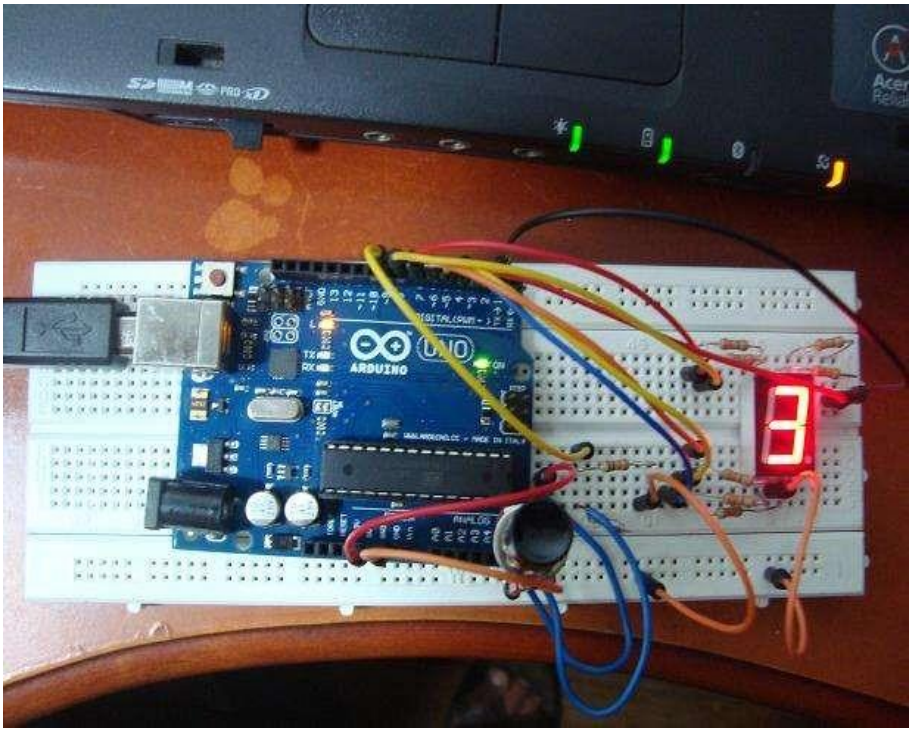
```
/* CONTADOR DE PULSOS POR PIN DIGITAL 9 PARA MOSTRAR EN UN
DISPLAY DE 7 SEGMENTOS DE 0 A 9 */

//CONEXIÓN DEL DISPLAY A LOS PINES DEL 2 AL 8
const int a=2;
const int b=3;
const int c=4;
const int d=5;
const int e=6;
const int f=7;
const int g=8;
//DECLARACIÓN DE VARIABLES
const int pulsa=9;
int valorpulsa;
int cont;
//CONFIGURACION DE LOS PINES
void setup() {
  pinMode;  //pulsador como entrada
  pinMode(a,OUTPUT);    //pin para display como salidas
  pinMode(b,OUTPUT);
  pinMode(c,OUTPUT);
  pinMode(d,OUTPUT);
  pinMode(e,OUTPUT);
  pinMode(f,OUTPUT);
  pinMode(g,OUTPUT);
}

//PROGRAMA PRINCIPAL
void loop() {
  valorpulsa=digitalRead(pulsa);
  if(valorpulsa==HIGH){
    cont++;
    delay(500);
    if(cont>=10){
      cont=0;
    }
  }
  switch (cont){
    case 0:
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,HIGH);
    digitalWrite(g,LOW);
    break;
```

```
    case 1:
    digitalWrite(a,LOW);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,LOW);
    digitalWrite(e,LOW);
    digitalWrite(f,LOW);
    digitalWrite(g,LOW);
    break;
    case 2:
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,LOW);
    digitalWrite(d,HIGH);
    digitalWrite(e,HIGH);
    digitalWrite(f,LOW);
    digitalWrite(g,HIGH);
    break;
    case 3:
    digitalWrite(a,HIGH);
    digitalWrite(b,HIGH);
    digitalWrite(c,HIGH);
    digitalWrite(d,HIGH);
    digitalWrite(e,LOW);
    digitalWrite(f,LOW);
    digitalWrite(g,HIGH);
    break;

case 4:
digitalWrite(a,LOW);
digitalWrite(b,HIGH);
digitalWrite(c,HIGH);
digitalWrite(d,LOW);
digitalWrite(e,LOW);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
break;
case 5:
digitalWrite(a,HIGH);
digitalWrite(b,LOW);
digitalWrite(c,HIGH);
digitalWrite(d,HIGH);
digitalWrite(e,LOW);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
break;
```

```
case 6:
digitalWrite(a,LOW);
digitalWrite(b,LOW);
digitalWrite(c,HIGH);
digitalWrite(d,HIGH);
digitalWrite(e,HIGH);
digitalWrite(f,HIGH);
digitalWrite(g,HIGH);
break;

   case 7:
   digitalWrite(a,HIGH);
   digitalWrite(b,HIGH);
   digitalWrite(c,HIGH);
   digitalWrite(d,LOW);
   digitalWrite(e,LOW);
   digitalWrite(f,LOW);
   digitalWrite(g,LOW);
   break;
   case 8:
   digitalWrite(a,HIGH);
   digitalWrite(b,HIGH);
   digitalWrite(c,HIGH);
   digitalWrite(d,HIGH);
   digitalWrite(e,HIGH);
   digitalWrite(f,HIGH);
   digitalWrite(g,HIGH);
   break;

   case 9:
   digitalWrite(a,HIGH);
   digitalWrite(b,HIGH);
   digitalWrite(c,HIGH);
   digitalWrite(d,LOW);
   digitalWrite(e,LOW);
   digitalWrite(f,HIGH);
   digitalWrite(g,HIGH);
   break;
}
```

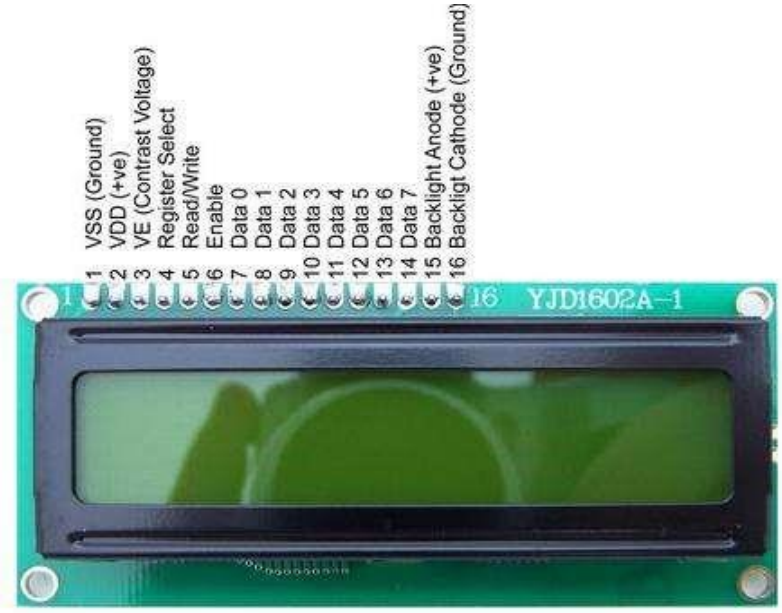
HARDWARE

No olvidar proteger cada uno de los segmentos del display con resistencias de 330 ohm en serie.



4. ESCRIBIR MENSAJE EN LCD 2X16

El display de cristal líquido LCD se maneja a través de la librería LiquidCrystal.h que se llama por medio de la directiva #include. La librería son grupos de funciones pre hechas escritas en este caso para Arduino. esta librería incluye las siguientes funciones, para el manejo del siguiente display:



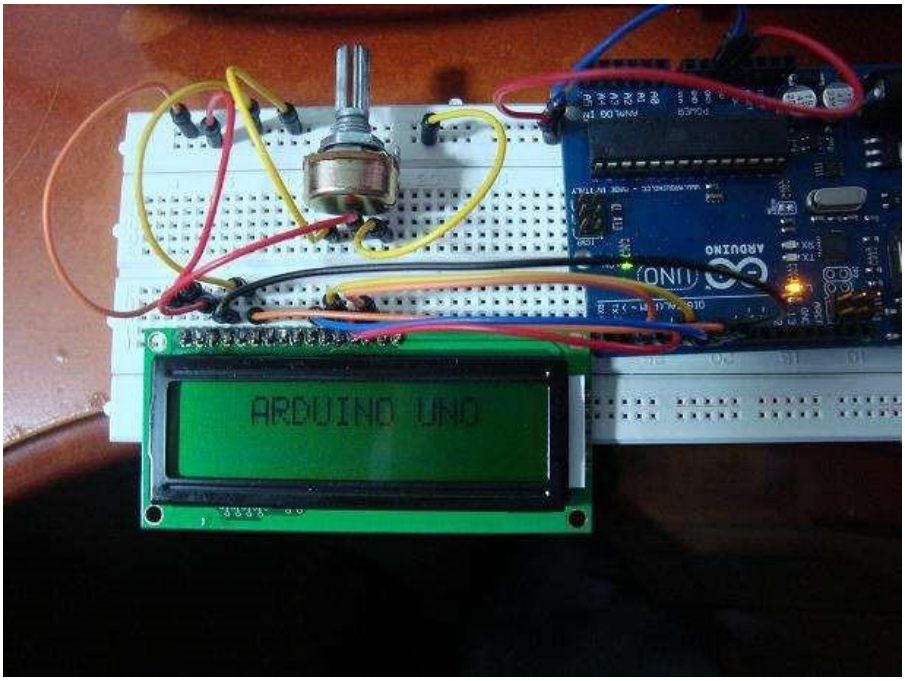
- LiquidCrystal lcd(rs,enable,d4,d5,d6,d7)
- LiquidCrystal lcd(rs,rw,enable,d4,d5,d6,d7)
- LiquidCrystal lcd(rs,enable,d0,d1,d2,d3,d4,d5,d6,d7)
- LiquidCrystal lcd(rs,rw,enable,d0,d1,d2,d3,d4,d5,d6,d7)
- lcd.begin(col,fila) // pone cursor en col, fila indicada
- lcd.clear () // borra el lcd y pone cursor en esquina superior izquierda
- lcd.home () // pone cursor en esquina superior izquierda
- lcd.setCursor (col, fila)
- lcd.write(dato) // escribe caracter de un byte
- lcd.print(dato) // imprime texto: char, byte, int, long, string
- lcd.noCursor () // esconde cursor
- lcd.blink () // parpadeo
- lcd.noBlink () // no parpadeo
- lcd.display () //despliega display si se ha apagado
- lcd.noDisplay () // apaga display
- lcd.scrollDisplayLeft () // corre el mensaje un espacio a la izquierda
- lcd.scrollDisplyRight () //corre el mensaje un espacio a la derecha
- lcd.autoScroll ()
- lcd.leftToRight () //texto de izquierda a derecha
- lcd.rightToLeft () //texto de derecha a izquierda
- lcd.createChar(num,dato) // num=0..7, dato=caracter

EJEMPLO: Desplegar en el display el mensaje ARDUINO UNO y correrlo a la derecha. Use los pines del 2 al 7 para conectar el display. El control de contraste del LCD pin 3 conectarlo a un pot de 10K.

Display7seg

```
#include <LiquidCrystal.h>


// DESPLEGAR EL MENSAJE "ARDUINO UNO" EN DISPLAY LCD 2X16

//#include <LiquidCrystal.h>
LiquidCrystal lcd(7,6,5,4,3,2); //pin para rs,e,d4,d5,d6,d7
```

```
void setup() {
  lcd.begin(16,2);  // lcd de 16 col y 2 filas
  lcd.print("ARDUINO UNO"); //mensaje a imprimir
}

void loop() {
  for(int cont=0;cont<12;cont++){
    lcd.scrollDisplayRight ();
    delay(500);
  }
}
```

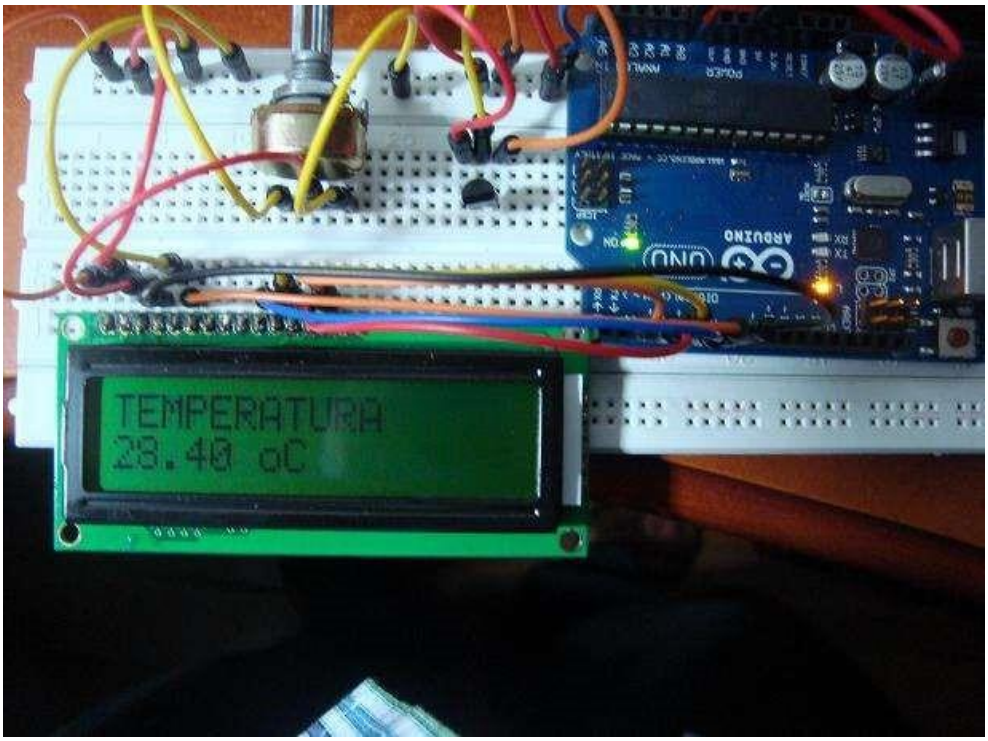


5. LEER LA TEMPERATURA DEL LM35 EN LCD 2X16

LeerTempLcd

```
// LEER LA TEMPERATURA DEL LM35 EN DISPLAY LCD 2X16

LiquidCrystal lcd(7,6,5,4,3,2); //pin para rs,e,d4,d5,d6,d7
const int sensor=0;  //sensor en analog A0
float temp;
float valorsensor;
```

```
void setup() {
  lcd.begin(16,2);  // lcd de 16 col y 2 filas
  lcd.print("TEMPERATURA"); //mensaje a imprimir
}

void loop() {
  valorsensor=(analogRead(sensor))*(5000/1023);
  temp=valorsensor/10;
  lcd.setCursor(0,2);
  lcd.print(temp);
  lcd.print(" oC");
}
```



CAPÍTULO 8. COMUNICACIÓN SERIE- ARDUINO

Un receptor asíncrono / transmisor universal (UART) es un bloque de circuitería responsable de implementar la comunicación serie. En esencia, la UART actúa como intermediario entre las interfaces paralelo y serie. En un extremo de la UART es un bus de ocho o lo que las líneas de datos (además de algunos pines de control), en el otro es los dos cables de serie - RX y TX.

El UART es el responsable del envío y recepción de datos en serie. Por el lado de transmisión, una UART debe crear el paquete de datos - añadiendo sincronización y bits de paridad - y enviar ese paquete por la línea TX con sincronización precisa (según la velocidad de transmisión). En el extremo de recepción, el UART tiene que probar la línea RX a tasas de acuerdo a la velocidad de transmisión y esperar los datos.

Las placas Arduino tienen al menos un puerto serie (también conocido como un UART o USART). Se comunica con los pines digitales 0 (RX) y 1 (TX), así como con el computador a través de USB. Por lo tanto, si utiliza estas funciones, no se puede también utilizar los pines 0 y 1 para la entrada o salida digital.

FUNCIONES

Serial.begin(tasa): Inicializa el puerto serie y asigna la tasa de baudios para la transmisión de datos serie. La típica tasa de baudios para comunicarse con el computador es 9600 aunque también soporta otras velocidades. Se puede utilizar el monitor incorporado de serie del entorno Arduino para comunicarse con una placa Arduino. En la ventana de edición:

Herramientas-->Monitor serie

seleccione la misma velocidad de transmisión utilizada en la llamada para comenzar (). Se utiliza para la comunicación entre la placa Arduino y un computador u otros dispositivos.

Nota: Cuando se usa la comunicación serie, los pines digitales 0 (Rx) y 1 (Tx) no pueden ser usados al mismo tiempo.

Serial.print(dato), Serial.println (): Envían datos por el puerto serie o el valor de una variable. Println finaliza una línea cuando ha terminado de transmitir.

Serial.available(). Devuelve el número de bytes a la espera de leerse.

Serial.read (). Devuelve un byte leído por el puerto serie, -1 si no hay datos.

Serial.write (). Escribe datos por el puerto serie.

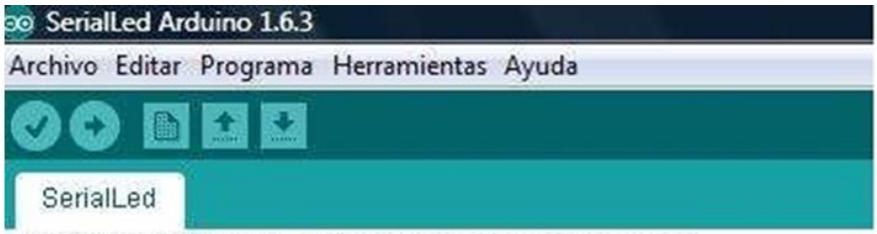
El objetivo de la siguiente práctica es adquirir habilidades para programar el Arduino Uno realizando algunas experiencias de comunicación serie de este sistema de desarrollo para el microcontrolador ATMEGA328 de ATMEL.

EQUIPO Y MATERIAL NECESARIO

- Un computador
- Placa Arduino Uno
- Cable de conexión para usb al arduino
- Protoboard
- Un LED
- Un sensor de temperatura LM35
- Un motor DC de 6V
- Un teclado matricial 4x4
- Una resistencia a 1/4W de 330Ω
- 1 potenciómetro lineal de 10KΩ
- Conectores

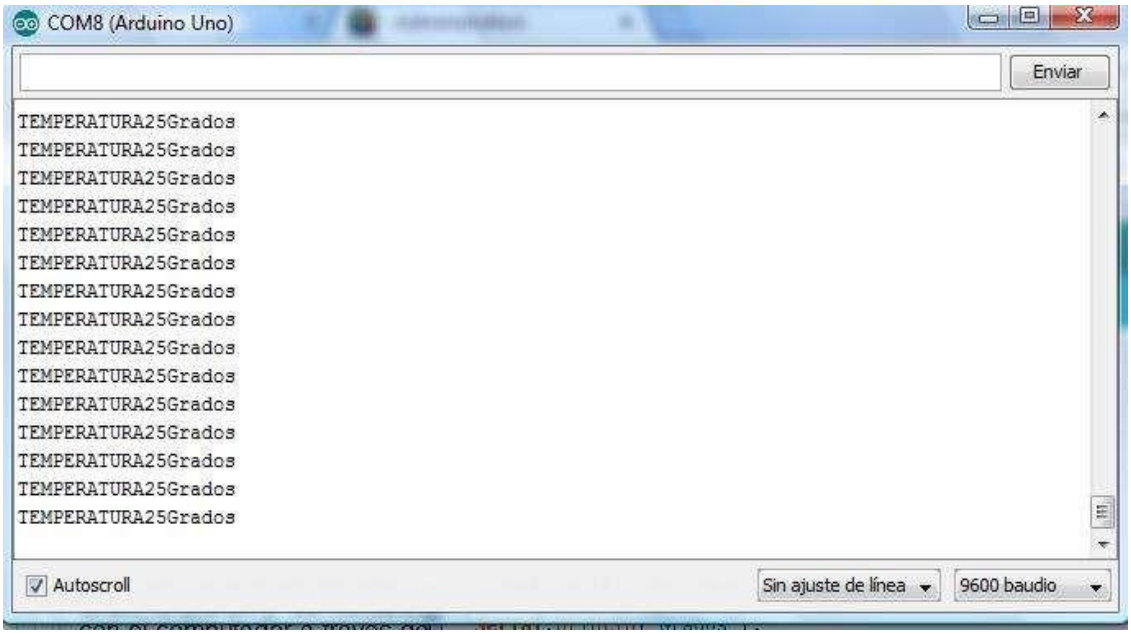
MONITOR SERIE DEL ARDUINO

En el siguiente ejemplo se va a leer la temperatura que mide el sensor LM35 en el Monitor serie del Arduino. Se usa un led en pin 4 para indicar el control de temperatura y el sensor en la entrada análoga A0.




```
/*LEER TEMPERATURA DEL SENSOR LM35 EN MONITOR
SERIE DEL ARDUINO, SENSOR EN PIN A0 Y LED EN PIN 4 */

const int sensor=0;
const int ledpin=4;
long temp;
long valorsensor;
```

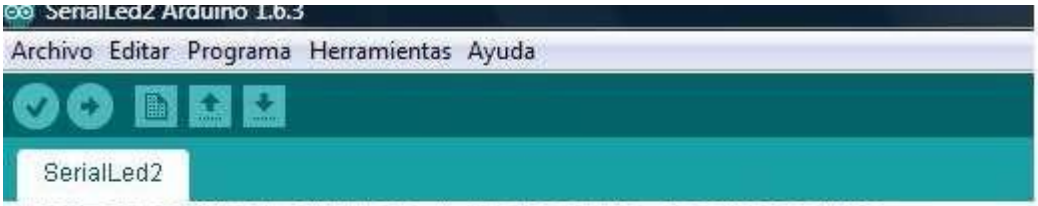
```
void setup() {
  Serial.begin(9600);
  pinMode(ledpin,OUTPUT);
}

void loop() {
  valorsensor=(analogRead(sensor))*(5000/102
  temp=valorsensor/10;
  Serial.print("TEMPERATURA");
  Serial.print(temp);
  Serial.println("Grados");
  delay(1000);
  if(temp>=26){
    digitalWrite(ledpin,HIGH);
  }
  else{
    digitalWrite(ledpin,LOW);
  }

}
```



Observar el monitor y comprobar que al pasar la temperatura de 26 grados el led se enciende y al bajar se apaga.

CONTROL DE UN LED POR EL MONITOR SERIE

El ejemplo consiste en encender o apagar un led colocado en el pin 4 desde el Monitor serie del Arduino, de tal forma que al enviarle un carácter 'V' se enciende y al enviarle el carácter 'F' se apaga.




```
/*CONTROL DE UN LED ENCENDIDO Y APAGADO DESDE EL MONITOR SERIE
DEL ARDUINO, LED EN PIN 4. CON 'V' SE ENCIENDE Y CON 'F' SE APAGA */

const int ledpin=4;
char datoserie;

void setup() {
  Serial.begin(9600);
  pinMode(ledpin,OUTPUT);
}
```

```
void loop() {
  //comprobar si hay dato serie de entrada
  if(Serial.available()>0){
    datoserie=Serial.read();
    if(datoserie=='V'){
      digitalWrite(ledpin,HIGH);
    }
    if(datoserie=='F'){
      digitalWrite(ledpin,LOW);
    }
  }
}
```

MANEJO DE STRING

Cuando se trabaja con el puerto serie es muy común transmitir textos. Arduino utiliza la función sprintf () que tiene la siguiente sintaxis:

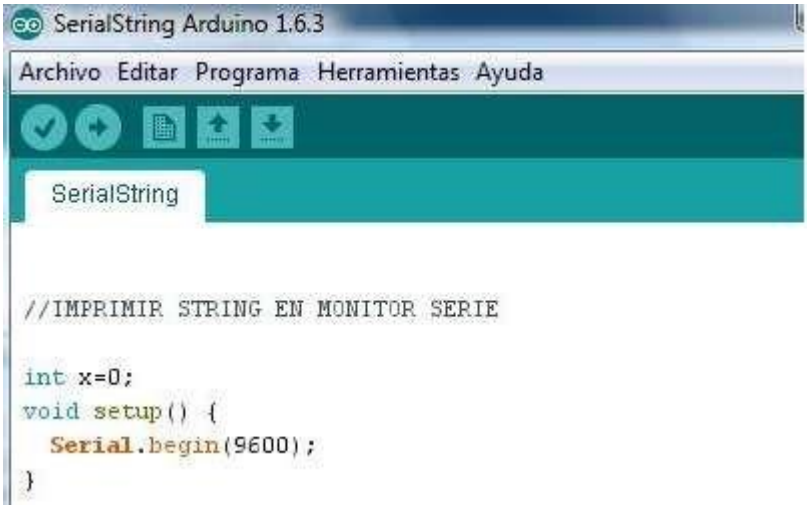
sprintf(stringResultado, stringFormato, argumentos);

StringResultado, es un arreglo char donde se almacenará el resultado.

StringFormato, especifica el formato, comienza con % y una letra para indicar el tipo (%d para enteros, %x para hexadecimal, %c para char).

Argumentos, lista de variables.

EJEMPLO:



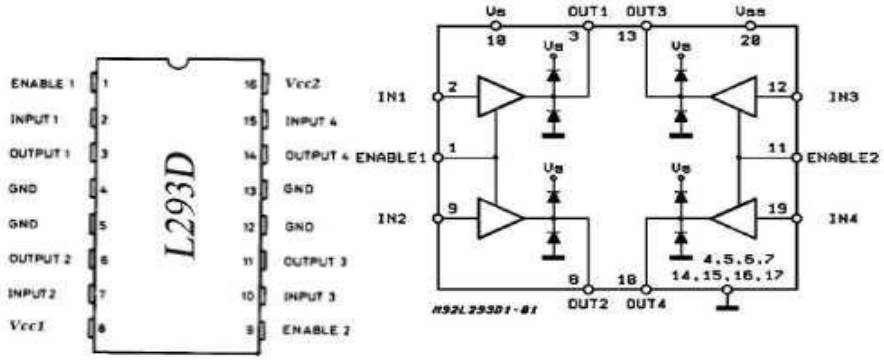

```
void loop() {
  char texto[20];
  sprintf(texto,"Variable: %d\n",x);
  Serial.println(texto); //imprime Variable: 0
  x++;
  delay(1000);
}
```

```
    }
    if(datoserie=='F'){
      digitalWrite(ledpin,LOW);
    }
  }
}
```

INVERTIR GIRO DE UN MOTOR DC

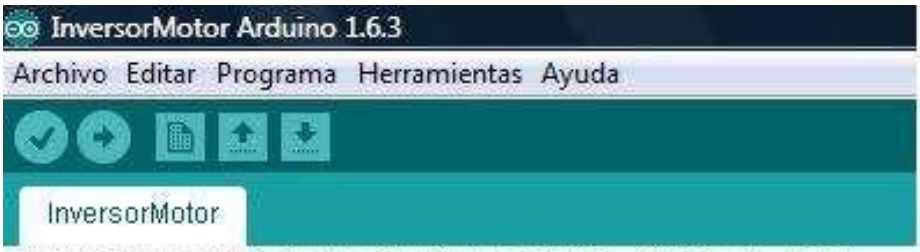
Invertir el giro de un motor de corriente continua usando el driver L293D para conexión del motor que soporta hasta 600 mA por canal. Se utiliza como control de giro un potenciómetro de 10K colocado en la entrada análoga A0 del conversor A/D de tal forma que cuando lea de 0 a 500 gire contrario al reloj, de 500 a 580 se pare y una lectura de

más de 580 gire en al sentido del reloj (recuerde que el conversor A/D traduce la lectura análoga de 0 a 1023). A continuación, se dan las características del driver L293D.



IN1	IN2	Function
High	High	Turn Anti-clockwise (Reverse)
High	Low	Turn clockwise (Forward)
High	High	Stop
High	Low	Stop
Low	X	Stop

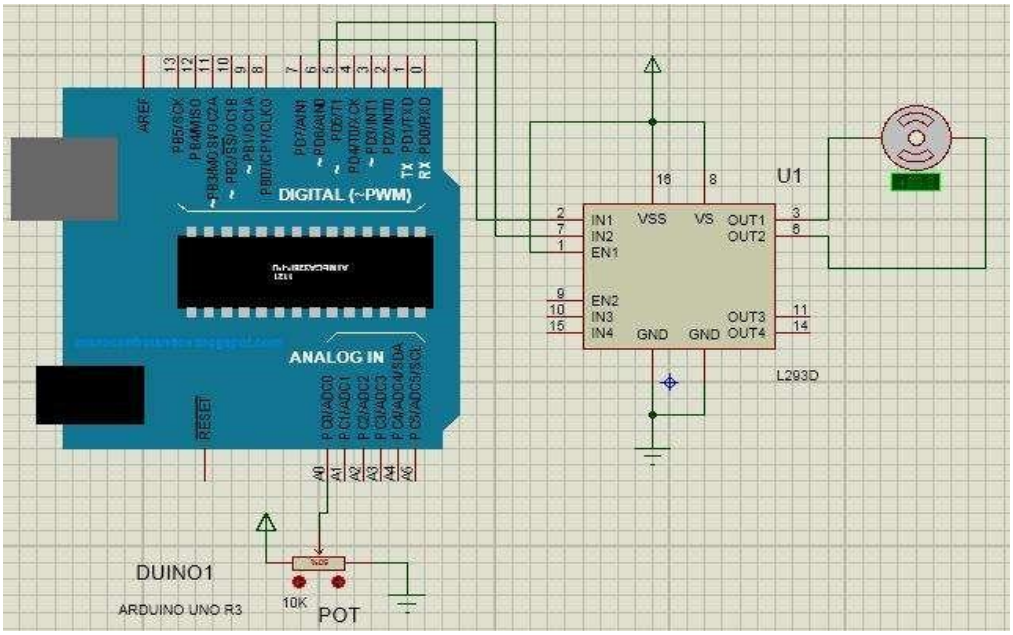
A continuación, se presenta el programa (sketch) para el Arduino.




```
/*INVERTIR GIRO DE UN MOTOR DC USANDO PUENTE H L293D
UTILIZANDO UN POT DE 10K CONECTADO EN EL PIN A0*/

const int pot=0;  //pot conectado a A0
const int pinmotor1=6;  // IN1 del L293 al pin 6
const int pinmotor2=5;  // IN2 del L293 al pin 5
int valorpot;

void setup() {
  pinMode(pinmotor1,OUTPUT);
  pinMode(pinmotor2,OUTPUT);
  //motor apagado inicialmente
  digitalWrite(pinmotor1,LOW);
  digitalWrite(pinmotor2,LOW);
}
```

```
void loop() {
  valorpot=analogRead(pot);
  if(valorpot<500){
    //giro contrario al reloj
    digitalWrite(pinmotor1,HIGH);
    digitalWrite(pinmotor2,LOW);
  }
  if(valorpot>580){
    //giro en sentido del reloj
    digitalWrite(pinmotor1,LOW);
    digitalWrite(pinmotor2,HIGH);
  }
    else{
      //motor apagado pot entre 500 y 580
      digitalWrite(pinmotor1,LOW);
      digitalWrite(pinmotor2,LOW);
    }
  }
```

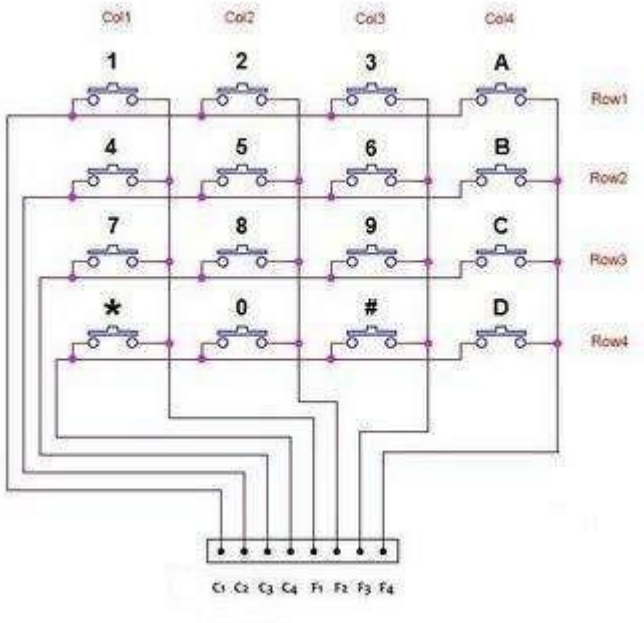
La implementación hardware se presenta a continuación. Se ha simulado en Proteus. Para descargar Proteus Versión 8 en este video está la forma de descargarlo e instalarlo.
https://www.youtube.com/watch?v=ls2MJla_gno
En este enlace se encuentra la forma de simular Arduino con Proteus.
Simulación de Arduino con Proteus



LEER TECLADO MATRICIAL

El Arduino Uno no tiene la librería del teclado hay que bajarla copiarla y pegarla al a la carpeta C --> archivos de programa-->arduino-->libraries el archivo Keypad que está en siguiente enlace.

http://playground.arduino.cc/code/keypad



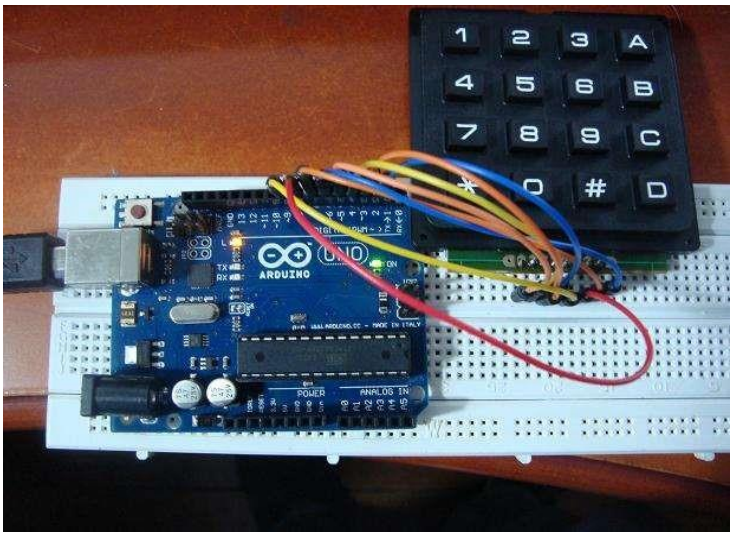
El siguiente ejemplo se lee el teclado por puerto serie:

```
//programa-->include library-->Keypad
#include <Keypad.h>
const byte rows=4;
const byte cols=4;
//configuración del teclado
char keys [rows][cols]={
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

```
byte rowspin[rows]={9,8,7,6};  //pines a las filas f1 f2 f3 f4
byte colspin[cols]={5,4,3,2};  //pines a las columnas c1 c2 c3 c4
Keypad keypad=Keypad(makeKeymap(keys),rowspin,colspin,rows,cols);

void setup() {
  Serial.begin(9600);
}

void loop() {
   char tecla=keypad.getKey();
   if(tecla){
     Serial.print("tecla=");
     Serial.println(tecla);
   }
   delay(10);
}
```

Comprobación de la configuración del teclado en el Monitor serie del Arduino.

HARDWARE IMPLEMENTADO.



BIBLIOGRAFÍA RECOMENDADA

1. LIBROS DE TEXTO Y REFERENCIAS ACADÉMICAS

- Floyd, Thomas L. (2015). Fundamentos de Sistemas Digitales (11ª ed.). México: Pearson Educación.
- Tocci, Ronald J.; Widmer, Neal S. & Moss, Gregory L. (2013). Sistemas Digitales: Principios y Aplicaciones (11ª ed.). México: Pearson.
- Mano, M. Morris & Kime, Charles R. (2016). Fundamentos de Diseño Lógico y de Computadoras (5ª ed.). Madrid: Cengage Learning.
- Roth, Charles H. & Kinney, Larry L. (2014). Fundamentos de Diseño Lógico (7ª ed.). México: Cengage Learning.
- Wakerly, John F. (2006). Diseño Digital: Principios y Prácticas (4ª ed.). México: Pearson.

2. SIMULACIÓN, PRÁCTICA Y LABORATORIO

- Proteus Design Suite. (2023). User Manual & Simulation Guide. LabCenter Electronics.
- Kleitz, William. (2017). Electrónica Digital: Un Enfoque Práctico (8ª ed.). Madrid: Pearson.

3. HOJAS DE DATOS (DATASHEETS) Y REFERENCIAS DE FABRICANTES

- Texas Instruments. (2023). Logic Guide (SCLD001). Dallas: TI Semiconductor.
- NXP Semiconductors. (2022). CMOS Logic Data Handbook. Eindhoven: NXP.
- ON Semiconductor. (2021). TTL/CMOS Logic IC Selection Guide. Phoenix: ON Semi.

4. PUBLICACIONES DEL AUTOR EN AMAZON KDP

Esta serie complementa el presente texto, formando una ruta formativa integral desde los fundamentos eléctricos hasta el control digital avanzado.

Polanía Puentes, Jorge Antonio. (2025). Fundamentos de Circuitos Eléctricos: Teoría y Práctica. Amazon KDP. Ley de Ohm, Kirchhoff, teoremas de circuitos y corriente alterna.

Polanía Puentes, Jorge Antonio. (2025). Medidores Eléctricos: Voltímetros, Amperímetros y Óhmetros. Amazon KDP. Diseño y uso de instrumentos de medición eléctrica.

Polanía Puentes, Jorge Antonio. (2025). Fundamentos de Electrónica: Semiconductores, Diodos y Transistores. Amazon KDP. Dispositivos semiconductores, rectificadores y circuitos transistorizados.

Polanía Puentes, Jorge Antonio. (2025). Amplificadores Transistorizados: Multietapas, Video y Sintonizados. Amazon KDP. Diseño de amplificadores analógicos y de potencia.

Polanía Puentes, Jorge Antonio. (2025). Osciladores y Multivibradores Transistorizados. Amazon KDP. Generación de señales y circuitos de conmutación.

Polanía Puentes, Jorge Antonio. (2025). Señales y Sistemas Continuos. Amazon KDP. Análisis de sistemas continuos, respuesta transitoria y diagramas de Bode.

Polanía Puentes, Jorge Antonio. (2025). Señales y Sistemas Discretos. Amazon KDP. Análisis de sistemas discretos, convolución y FFT.

Polanía Puentes, Jorge Antonio. (2025). Aprende Control con MATLAB. Amazon KDP. Implementación práctica de sistemas de control con MATLAB.

Polanía Puentes, Jorge Antonio. (2025). Control de Motores con MATLAB. Amazon KDP. Modelado y control de motores eléctricos con herramientas computacionales.

Polanía Puentes, Jorge Antonio. (2025). Instalaciones Eléctricas Residenciales. Amazon KDP. Diseño de instalaciones eléctricas domiciliarias y normatividad.

Polanía Puentes, Jorge Antonio. (2025). Dispositivos de Control Electrónico. Amazon KDP. Dispositivos semiconductores de potencia y control.

SOBRE EL AUTOR

1. FORMACIÓN ACADÉMICA

Secundaria: Bachiller, 1969. Colegio Nacional Santa Librada de Neiva Pregrado: Ingeniero Electrónico,1976. Universidad Distrital de Santafé de Bogotá. Tesis Meritoria. Matrícula profesional: CN206-45500 del Consejo Profesional Nacional de Ingenierías Eléctrica, Mecánicas y Profesiones afines. Postgrado: Magister en Ingeniería Electrónica, 1991. Universidad Nacional Autónoma de México. Mención Honorífica. Convalidación Resolución No 823 de 1994 del ICFES. Cursos de postgrado: Microprocesadores y sistemas de desarrollo. Sistemas de comunicación de datos. Interconexión de redes usando TCP/IP. Sistemas de telecomunicaciones por satélite. HTML Scripting. Autoevaluación institucional. Programación Turbo Pascal. Planeación institucional. Planeación y desarrollo institucional.

2. EXPERIENCIA ACADÉMICA

Experiencia académica de 30 años como Profesor de Tiempo Completo de la Universidad Surcolombiana adscrito a la Facultad de Ingeniería, del 7 de febrero de 1977 al 30 de marzo de 2007. Profesor Titular desde el 1 de diciembre de 1993. Profesor de Pregrado en el Programa de Ingeniería Electrónica de las asignaturas: Electrónica Analógica, Arquitectura de computadores. Electrónica Digital, Microcontroladores. Programación en Matlab. Control digital. Señales y sistemas. Procesamiento digital de señales. Energías Alternativas. Profesor de Postgrado en la Maestría de Gestión e Ingeniería Ambiental de las asignaturas: Energética Ambiental y Simulación de Sistemas Ambientales.

3. EXPERIENCIA ADMINISTRATIVA

Experiencia administrativa de más de 10 años en los siguientes cargos: Decano de la Facultad de Ingeniería, Universidad Surcolombiana. 1984-1986 Decano de la Facultad de Ingeniería, Universidad Surcolombiana.1987–1988 Jefe de la Oficina de Planeación, Universidad Surcolombiana. 1988 – 1989 Rector de la Universidad Surcolombiana. 1997 – 2000 Director del Postgrado de Automatización industrial. 2000 Jefe de Programa de Ingeniería Electrónica. 2000 – 2001 Director del Departamento de Ingeniería Electrónica. 2006

4. EXPERIENCIA INVESTIGATIVA

Coordinador del grupo de investigación Nuevas tecnologías. Categoría C Colciencias. 2005. Proyectos de investigación: “Simulación de algoritmos de control digital con Matlab”. 2005. “Diseño y construcción de un electrocardiógrafo digital inalámbrico”. 2006. Diseño e implementación de un sistema de riego inteligente para un cultivo de mango”. 2007. Hoja de vida CvLAC COLCIENCIAS

5. PRODUCCIÓN INTELECTUAL

Cursos publicados en forma virtual en la web www.ceduvirt.com: Circuitos eléctricos. Semiconductores. Electrónica básica. Electrónica industrial. Electrónica digital. Microcontroladores. Señales y sistemas. Control digital con Matlab, Procesamiento de señales. Procesos con Matlab y Simulink. Teoría de sistemas. Energía solar. Aplicaciones en Ingeniería ambiental.