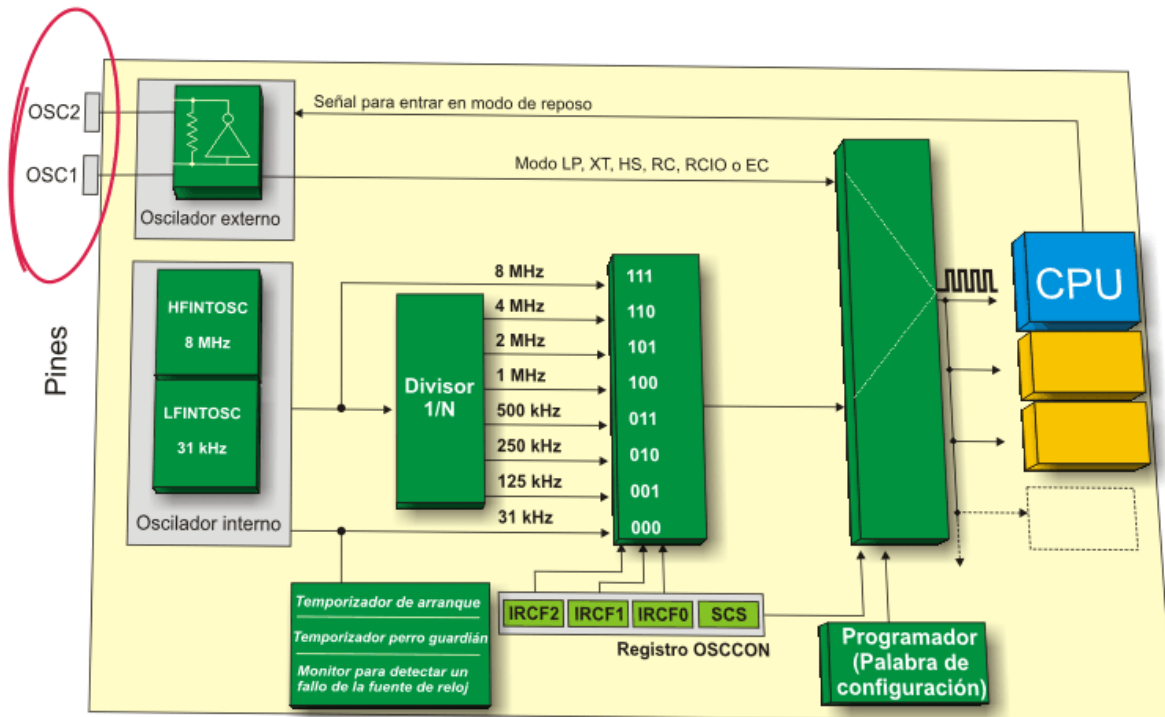


3.10 Oscilador de Reloj

GO

Como se muestra en la siguiente figura, la señal de reloj se genera por uno de los dos osciladores integrados.



Un **oscilador externo** está instalado fuera del microcontrolador y conectado a los pines OSC1 y OSC2. Es denominado 'externo' porque utiliza componentes externos para generar una señal de reloj y estabilizar la frecuencia. Estos son: cristal de cuarzo, resonador cerámico o circuito resistor - capacitor. El modo de funcionamiento del oscilador se selecciona por los bits, que se envían durante la programación, denominados Palabra de Configuración. El **oscilador interno** consiste en dos osciladores internos separados: El HFINTOSC es un oscilador interno de alta frecuencia calibrado a 8MHz. El microcontrolador puede utilizar una señal de reloj generada a esta frecuencia o después de haber sido dividida en el pre-escalador. El LFINTOSC es un oscilador interno de baja frecuencia calibrado a 31 kHz. Sus pulsos de reloj se utilizan para funcionamiento de los temporizadores de encendido y perro guardián, asimismo puede utilizarse como fuente de señal de reloj para el funcionamiento de todo el microcontrolador. El bit *System Clock Select* (bit de selección del reloj del sistema - SCS) del registro OSCCON determina si una fuente de señal de reloj del microcontrolador será interna o externa.

Registro OSCCON

El registro OSCCON gobierna el microcontrolador y las opciones de selección de frecuencia. Contiene los siguientes bits: bits de selección de frecuencia (IRCF2, IRCF1, IRCF0), bits de estado de frecuencia (HTS, LTS), bits de control de reloj del sistema (OSTA, SCS).

OSCCON		R/W (1)	R/W (1)	R/W (0)	R (1)	R (0)	R (0)	R/W (0)	Características
	-	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
(0)	Después del reinicio, el bit se pone a cero
(1)	Después del reinicio, el bit se pone a uno

IRCF2-0 - Internal Oscillator Frequency Select bits. (bits de selección de frecuencia del oscilador interno). El valor del divisor de frecuencias depende de la combinación de estos tres bits. La frecuencia de reloj del oscilador interno se determina de la misma manera.

IRCF2	IRCF1	IRCF0	FRECUENCIA	OSC.
1	1	1	8 MHz	HFINTOS
1	1	0	4 MHz	HFINTOS
1	0	1	2 MHz	HFINTOS
1	0	0	1 MHz	HFINTOS
0	1	1	500 kHz	HFINTOS
0	1	0	250 kHz	HFINTOS
0	0	1	125 kHz	HFINTOS
0	0	0	31 kHz	LFINTOS

OSTS - Oscillator Start-up Time-out Status bit (bit de estado del temporizador de encendido) indica cuál fuente de reloj está actualmente en uso. Es un bit de sólo lectura.

- 1 - Se utiliza el oscilador de reloj externo.

- 0 - Se utiliza uno de los osciladores de reloj interno (HFINTOSC o LFINTOSC).

HTS - HFINTOSC Status bit (8 MHz - 125 kHz) (bit de estado del HFINTOSC) indica si el oscilador interno de alta frecuencia funciona en modo estable.

- 1 - HFINTOSC está estable.
- 0 - HFINTOSC no está estable.

LTS - LFINTOSC Stable bit (31 kHz) (bit de estado del LFINTOSC) indica si el oscilador de baja frecuencia funciona en modo estable.

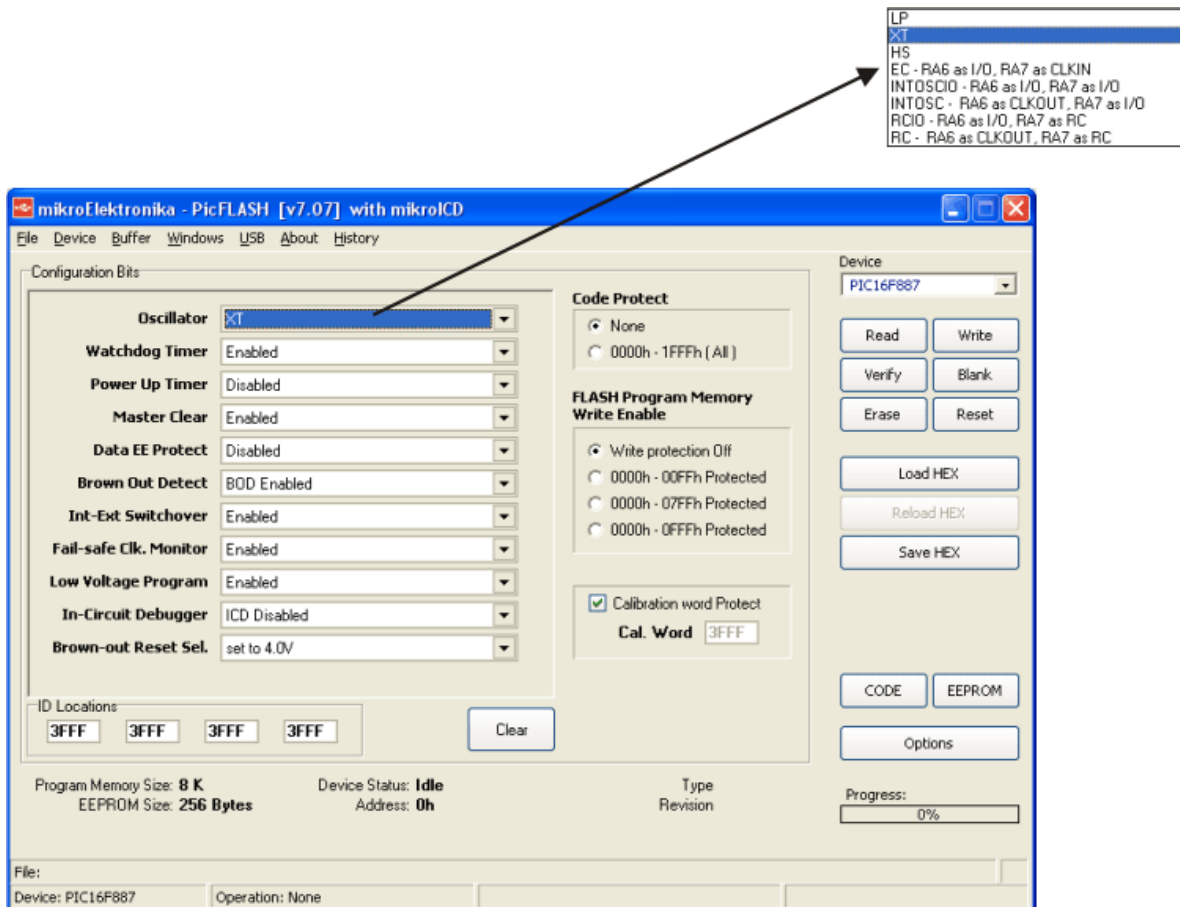
- 1 - LFINTOSC está estable.
- 0 - LFINTOSC no está estable.

SCS - System Clock Select bit (bit de selección del reloj del sistema) determina cuál oscilador se utilizará como una fuente de reloj.

- 1 - Oscilador interno se utiliza como reloj del sistema.
- 0 - Oscilador externo se utiliza como reloj del sistema. El modo del oscilador se configura por medio de los bits, denominados *Palabra de Configuración*, que se escribe en la memoria del microcontrolador durante el proceso de la programación.

MODOS DE RELOJ EXTERNO

El oscilador externo se puede configurar para funcionar en uno de varios modos, lo que habilita que funcione a diferentes velocidades y utilice diferentes componentes para estabilizar la frecuencia. El modo de funcionamiento se selecciona durante el proceso de escribir un programa en el microcontrolador. Antes que nada, es necesario activar el programa en una PC que se utilizará para programar el microcontrolador. En este caso, es el programa PICflash. Pulse sobre la casilla del oscilador y seleccione uno de la lista desplegable. Los bits apropiados se pondrán a uno automáticamente, formando parte de varios bytes, denominados Palabra de Configuración. Durante el proceso de la programación del microcontrolador, los bytes de la Palabra de Configuración se escriben en la memoria ROM del microcontrolador y se almacenan en los registros especiales no disponibles al usuario. A base de estos bits, el microcontrolador “sabe” qué hacer, aunque eso no se indica explícitamente en el programa.



Modo de funcionamiento se selecciona después de escribir y compilar un programa

OSCILADOR EXTERNO EN MODO EC

El modo de reloj externo (EC - external clock) utiliza un oscilador externo como una fuente de señal de reloj. La máxima frecuencia de señal de reloj está limitada a 20 MHz.

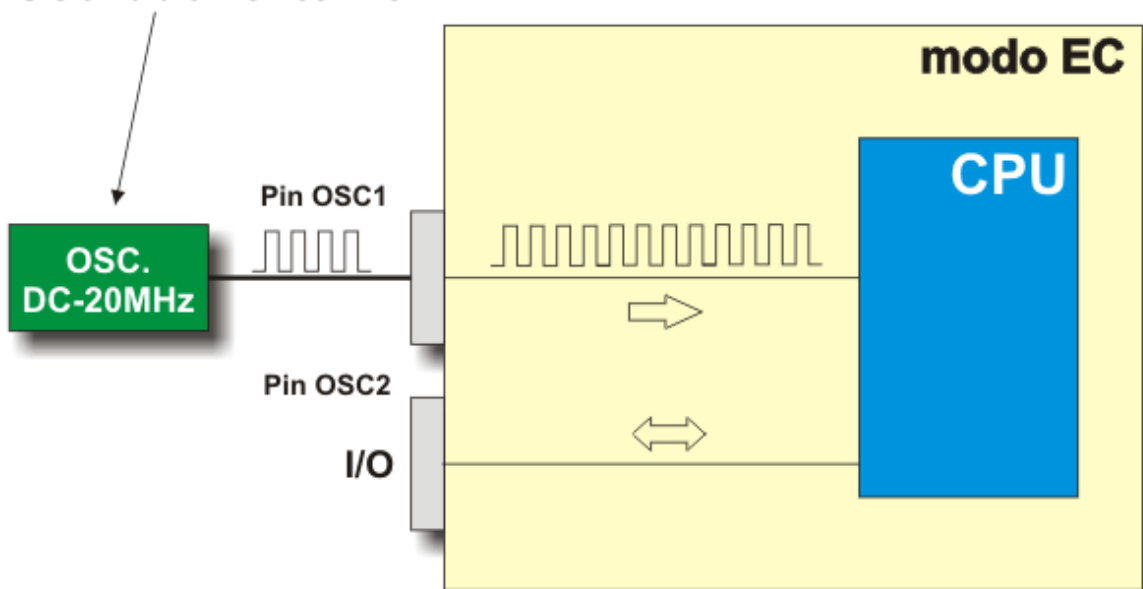


Las ventajas del funcionamiento del oscilador externo en modo EC son las siguientes:

- La fuente de reloj externa independiente está conectada al pin de entrada OSC1. El pin OSC2 está disponible como pin de E/S de propósito general;

- Es posible sincronizar el funcionamiento del microcontrolador con los demás componentes incorporados en el dispositivo;
- En este modo el microcontrolador se pone a funcionar inmediatamente después de encenderlo. No se requiere esperar para estabilizar la frecuencia.
- Al deshabilitar temporalmente la fuente de reloj externa, se detiene el funcionamiento del dispositivo, dejando todos los datos intactos. Después de reiniciar el reloj externo, el dispositivo sigue funcionando como si no hubiera pasado nada.

Oscilador externo



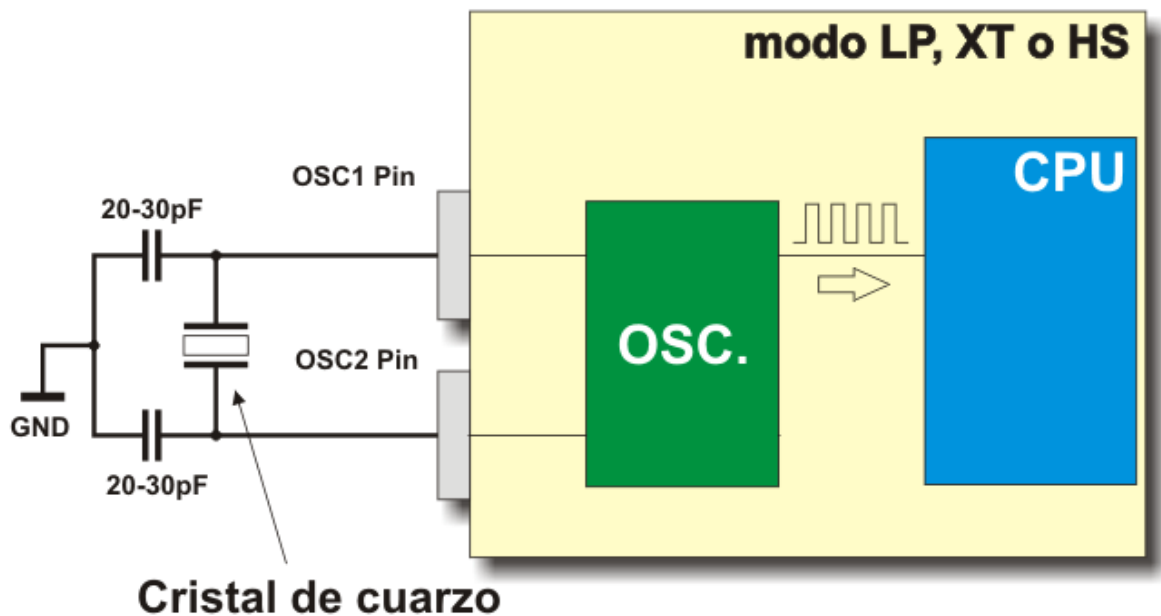
OSCILADOR EXTERNO EN MODO LP, XT O HS



Los modos LP, XT y HS utilizan un oscilador externo como una fuente de reloj cuya frecuencia está determinada por un cristal de cuarzo o por resonadores

cerámicos conectados a los pines OSC1 y OSC2. Dependiendo de las características de los componentes utilizados, seleccione uno de los siguientes modos:

- **Modo LP** - (Baja potencia) se utiliza sólo para cristal de cuarzo de baja frecuencia. Este modo está destinado para trabajar con cristales de 32.768 KHz normalmente embebidos en los relojes de cristal. Es fácil de reconocerlos por sus dimensiones pequeñas y una forma cilíndrica. Al utilizar este modo el consumo de corriente será menor que en los demás modos.
- **Modo XT** se utiliza para cristales de cuarzo de frecuencias intermedias hasta 8 MHz. El consumo de corriente es media en comparación con los demás modos.
- **Modo HS** - (Alta velocidad) se utiliza para cristales de reloj de frecuencia más alta de 8 MHz. Al utilizar este modo el consumo de corriente será mayor que en los demás modos.



RESONADORES CERÁMICOS EN MODO XT O HS

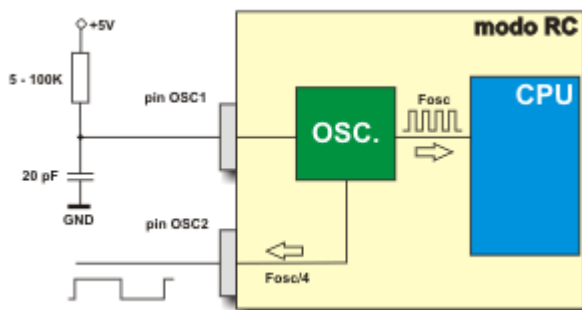
Los resonadores cerámicos son similares a los cristales de cuarzo según sus características, por lo que se conectan de la misma manera. A diferencia de los cristales de cuarzo, son más baratos y los osciladores que hacen uso de ellos son de calidad más baja. Se utilizan para las frecuencias de reloj entre 100 kHz y 20 MHz.



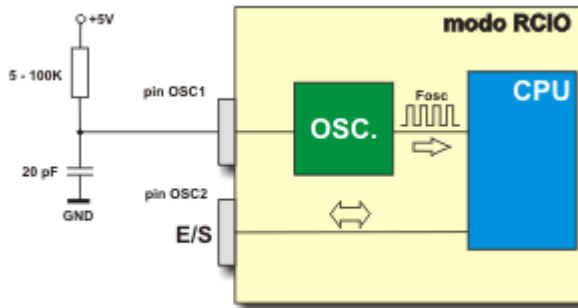
OSCILADOR EXTERNO EN MODOS RC Y RCIO

El uso de los elementos para estabilizar la frecuencia sin duda alguna tiene muchas ventajas, pero a veces realmente no es necesario. En la mayoría de casos el oscilador puede funcionar a frecuencias que no son precisamente definidas, así que sería una pérdida de dinero embeber tales elementos. La solución más simple y más barata en estas situaciones es utilizar una resistencia y un capacitor para el funcionamiento del oscilador. Hay dos modos:

Modo RC. Cuando el oscilador externo se configura a funcionar en modo RC, el pin OSC1 debe estar conectado al circuito RC como se muestra en la figura a la derecha. La señal de frecuencia del oscilador RC dividida por 4 está disponible en el pin OSC2. Esta señal se puede utilizar para la calibración, sincronización o para otros propósitos.



Modo RCIO. De manera similar, el circuito RC está conectado al pin OSC1. Esta vez, el pin OSC2 está disponible para ser utilizado como pin de E/S de propósito general.



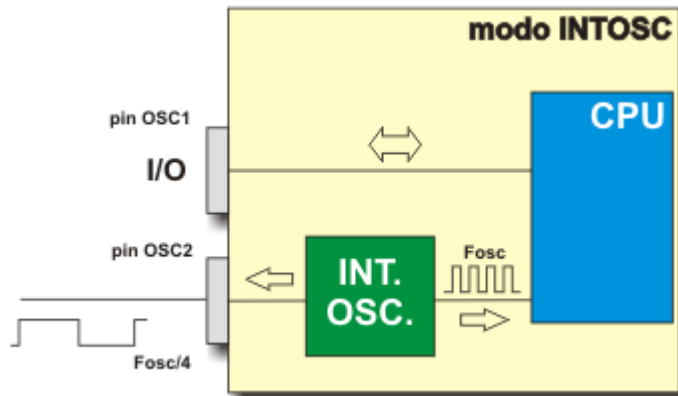
En ambos casos se le recomienda utilizar los componentes como se muestra en la figura. La frecuencia de este oscilador se calcula por medio de la fórmula $f = 1/T$ según la que:

- f = frecuencia [Hz];
- $T = R * C$ = constante de tiempo [s];
- R = resistencia eléctrica [Ω]; y
- C = capacitancia del condensador [F].

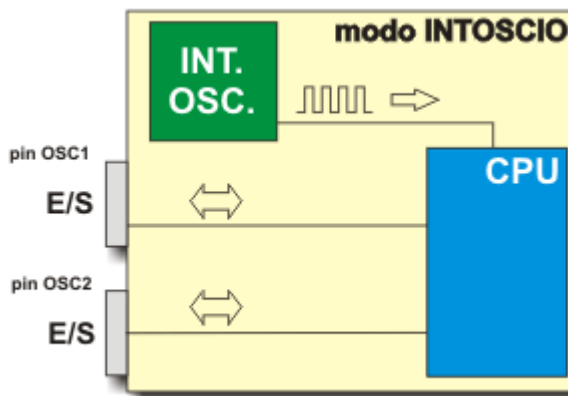
MODOS DE RELOJ INTERNO

El circuito del oscilador interno consiste en dos osciladores separados que se pueden seleccionar como la fuente del reloj del microcontrolador: El oscilador **HFINTOSC** está calibrado de fábrica y funciona a 8Mhz. La frecuencia de este oscilador se puede configurar por el usuario por medio de software utilizando los bits del registro OSCTUNE. El oscilador **LFINTOSC** no está calibrado de fábrica y funciona a 31kHz. Similar al oscilador externo, el interno también puede funcionar en varios modos. El modo de funcionamiento se selecciona de la misma manera que en el oscilador externo - por medio de los bits que forman *Palabra de configuración*. En otras palabras, todo se lleva a cabo dentro del software de PC antes de escribir un programa en el microcontrolador.

OSCILADOR INTERNO EN MODO INTOSC En este modo, el pin OSC1 está disponible para ser utilizado como pin de E/S de propósito general. La señal de frecuencia del oscilador interno dividida por 4 está disponible en el pin OSC2.

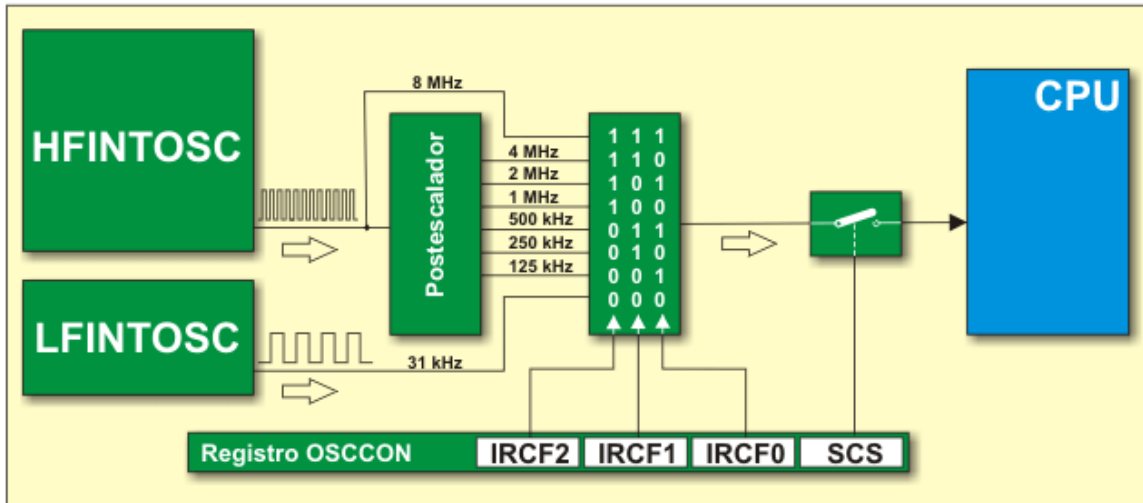


OSCILADOR INTERNO EN MODO INTOSCIO En este modo, los dos pines están disponibles como pines de E/S de propósito general.



CONFIGURACIÓN DEL OSCILADOR INTERNO

El oscilador interno consiste en dos circuitos separados: 1. El oscilador interno de alta frecuencia HFINTOSC está conectado al post-escalador (divisor de frecuencias). Está calibrado de fábrica y funciona a 8 Mhz. Al utilizar el post-escalador, este oscilador puede producir una señal de reloj a una de siete frecuencias. La selección de frecuencia se realiza dentro del software utilizando los pines IRCF2, IRCF1 y IRCF0 del registro OSCCON. El HFINTOSC está habilitado al seleccionar una de siete frecuencias (entre 8 Mhz y 125 kHz) y poner a uno el bit de la fuente de reloj del sistema (SCS) del registro OSCCON. Como se muestra en la siguiente figura , todo el procedimiento se realiza por medio de los bits del registro OSCCON.

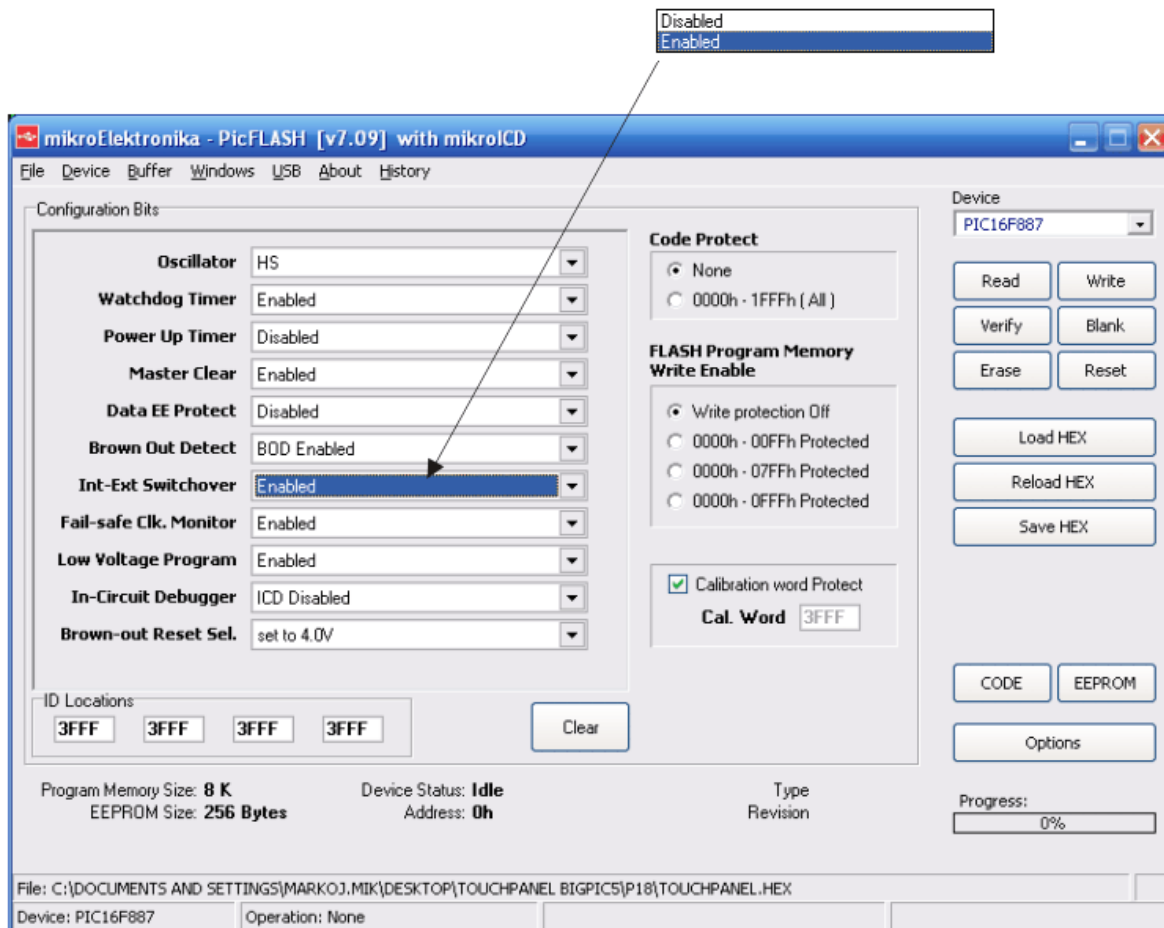


2. El oscilador de baja frecuencia LFINTOSC no está calibrado de fábrica y funciona a 31 kHz. Está habilitado al seleccionar la frecuencia (bits del registro OSCCON) y poner a uno el bit SCS del mismo registro.

MODO DE CAMBIO AUTOMÁTICO DE VELOCIDAD DE RELOJ (TWO-SPEED CLOCK START-UP MODE)

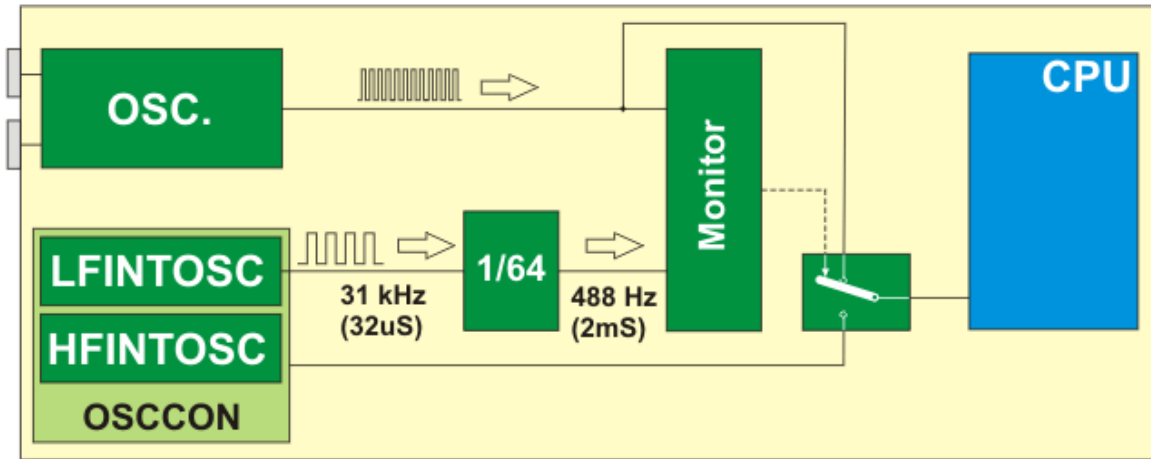
El modo de cambio automático de velocidad de reloj se utiliza para reducir el consumo de corriente cuando el microcontrolador funciona en modo de reposo. ¿De qué se trata todo esto? Cuando se configura en modo LP, XT o HS, el oscilador externo se desactiva al pasar a modo de reposo para reducir el consumo de corriente total del dispositivo. Cuando se cumplen las condiciones de "despertamiento", el microcontrolador no se pone a funcionar inmediatamente puesto que tiene que esperar a que se establezca la frecuencia de señal de reloj. Este tiempo muerto dura exactamente 1024 pulsos, después de que el microcontrolador continúa con la ejecución del programa. El caso es que se ejecutan sólo unas pocas instrucciones antes de que el microcontrolador vuelva al modo de reposo. Eso significa que la mayoría de tiempo así como la mayoría de corriente de baterías se ha perdido en vano. El caso se soluciona utilizando el oscilador interno para ejecutar el programa durante la duración de 1024 pulsos. Tan pronto como se establece la frecuencia del oscilador externo, él retoma automáticamente "el papel principal". Todo el procedimiento se habilita al poner a uno el bit de palabra de

configuración. Para programar el microcontrolador, es necesario seleccionar la opción Int-Ext Switchover (conmutación interna/externa) por software.

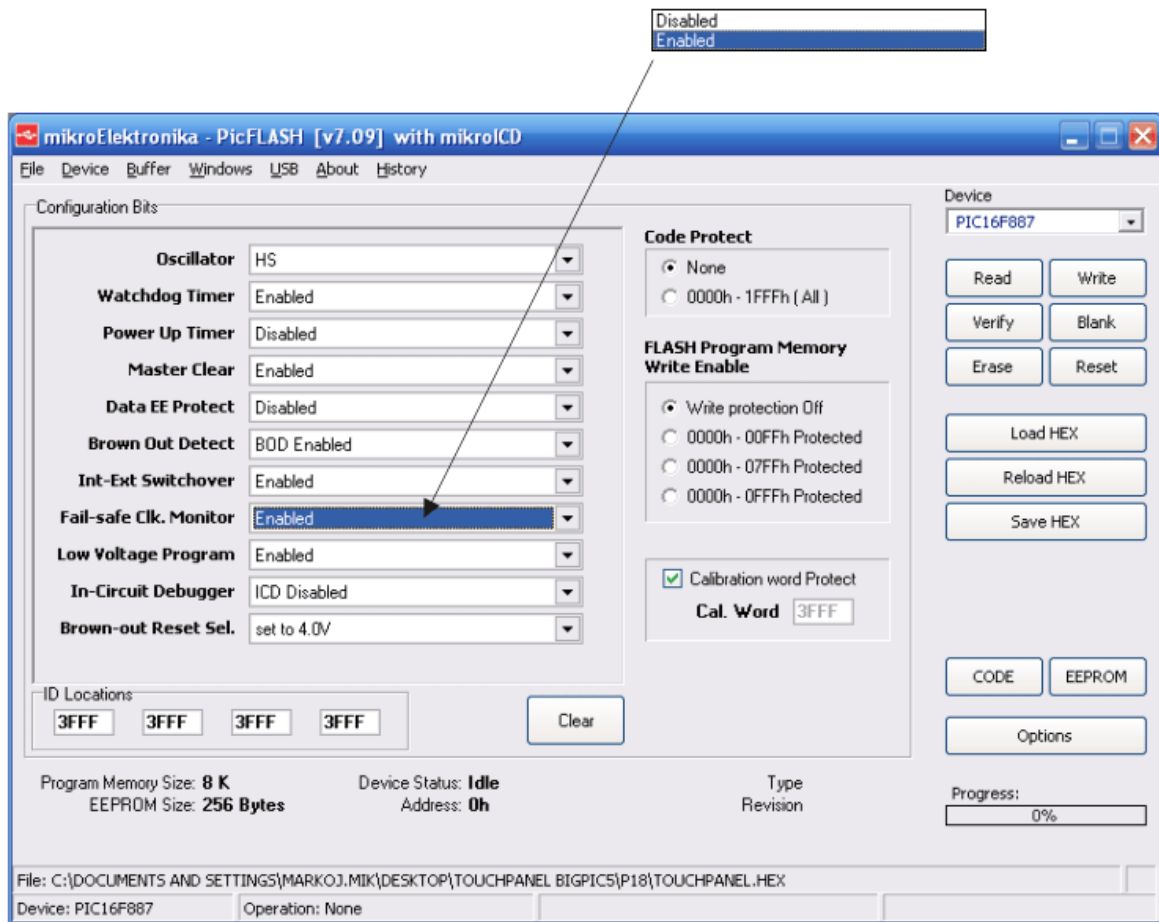


MONITOR PARA DETECTAR UN FALLO DE LA FUENTE DE RELOJ (FAIL-SAFE CLOCK MONITOR)

Como indica su nombre, el monitor para detectar un fallo de la fuente de reloj (Fail-Safe Clock Monitor - FSCM) monitorea el funcionamiento externo y permite al microcontrolador continuar con la ejecución de programa en caso de que el oscilador falle por alguna razón. En tal caso, el oscilador interno toma su función.



El monitor detecta un fallo al comparar las fuentes de reloj interno y externo. Si los pulsos del oscilador externo tardan más de 2mS en llegar, la fuente de reloj será automáticamente cambiada por la interna. Así, el oscilador interno sigue funcionando controlado por los bits del registro OSCCON. Si el bit OSFIE del registro PIE2 está a uno, se producirá una interrupción. El reloj interno sigue siendo la fuente del reloj del sistema hasta que el dispositivo reinicie con éxito el oscilador externo que vuelve a ser la fuente de reloj del sistema. De manera similar a casos anteriores, este módulo está habilitado al cambiar la palabra de configuración justamente antes de que se inicie el proceso de programar el chip. Esta vez, esto se realiza al seleccionar la opción Fail-Safe Clock Monitor.



Registro OSCTUNE

Los cambios del registro OSCTUNE afectan a la frecuencia del oscilador HFINTOSC, pero no a la frecuencia del LFINTOSC. No hay ninguna indicación de que haya ocurrido desplazamiento de frecuencia durante el funcionamiento del microcontrolador.

OSCTUNE				R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Características
	Bit 7	Bit 6	Bit 5	TUN4	TUN3	TUN2	TUN1	TUN0	Nombre de bit
	-	-	-						

Leyenda

- Bit no implementado
- R/W Bit de lectura/escritura
- (0) Después del reinicio, el bit se pone a cero

TUN4 - TUN0 Frequency Tuning bits. (bits de calibrar la frecuencia). Al combinar estos cinco bits, la frecuencia del oscilador de 8Mhz se reduce o se aumenta. De este modo, las frecuencias obtenidas por la división en el post-escalador cambian también.

TUN4	TUN3	TUN2	TUN1	TUN0	FR
0	1	1	1	1	Má
0	1	1	1	0	
0	1	1	0	1	
0	0	0	0	1	
0	0	0	0	0	Ca
1	1	1	1	1	
1	0	0	1	0	
1	0	0	0	1	
1	0	0	0	0	Mí

La EEPROM es un segmento de memoria separado, que no pertenece a la memoria de programa (ROM), tampoco a la memoria de datos (RAM). Aunque a estas localidades de memoria no se les puede acceder fácil y rápidamente, su propósito es insustituible. Los datos almacenados en la EEPROM están permanentemente guardados incluso al apagar la fuente de alimentación, y pueden ser cambiados en cualquier momento. Por estas características excepcionales cada byte de la EEPROM se considera valioso.

3.11 Memoria EEPROM

GO

El microcontrolador PIC16F887 dispone de 256 localidades de memoria EEPROM controlados por los bits de los siguientes registros:

- EECON1 (registro de control);
- EECON2 (registro de control);
- EEDAT (almacena los datos listos para escritura y lectura); y
- EEADR (almacena la dirección de la EEPROM a la que se accede).

Además, el registro EECON2 no es un registro verdadero, no existe físicamente en el chip. Se utiliza sólo durante la escritura de los datos en la memoria. Los registros EEDATH y EEADRH se utilizan durante la escritura y lectura de la EEPROM. Los dos se utilizan también durante la escritura y lectura de la memoria de programa (FLASH). Por considerar esto una zona de riesgo (por supuesto usted no quiere que el microcontrolador borre su propio programa por casualidad), no vamos a discutirlo aquí, no obstante le avisamos que tenga cuidado.

Registro EECON1

EECON1	R/W (x)				R/W (x)	R/W (0)	R/S (0)	R/S (0)	Características
	EEPGD	-	-	-	WRERR	WREN	WR	RD	Nombre de bit
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

Leyenda

-	Bit no implementado
R/W	Bit de lectura/escritura
R	Bit de lectura
S	Bit se puede poner sólo a uno
(0)	Después del reinicio, el bit se pone a cero
(x)	Después del reinicio, el estado de bit es desconocido

EEPGD - Program/Data EEPROM Select bit (bit de selección de memorias)

- 1 - Acceso a la memoria Flash de programa.
- 0 - Acceso a la memoria de datos EEPROM.

WRERR - EEPROM Error Flag bit (bit de error de escritura)

- 1 - Se produce un error de escritura de forma prematura y ha ocurrido un error.
- 0 - Se ha completado la operación de escritura.

WREN - EEPROM Write Enable bit (bit de habilitación de escritura)

- 1 - Escritura de datos en la EEPROM habilitada.
- 0 - Escritura de datos en la EEPROM deshabilitada.

WR - Write Control bit (bit de control de escritura)

- 1 - Se ha iniciado una operación de escritura de datos en la EEPROM.
- 0 - Se ha completado una operación de escritura de datos en la EEPROM.

RD - Read Control bit (bit de control de lectura)

- 1 - Inicia una lectura de la memoria EEPROM.
- 0 - Lectura de la memoria EEPROM deshabilitada.

LECTURA DE LA MEMORIA EEPROM

Para leer los datos de la memoria EEPROM, siga los siguientes pasos:

- **Paso 1:** Escribir la dirección (00h - FFh) en el registro EEADR.
- **Paso 2:** Seleccionar el bloque de memoria EEPROM al poner a cero el bit EEPGD del registro EECON1.
- **Paso 3:** Poner a uno el bit RD del mismo registro para leer el contenido de la localidad.
- **Paso 4:** El dato se almacena en el registro EEDAT y está listo para su uso.

El siguiente ejemplo muestra el procedimiento anteriormente descrito al escribir un programa en lenguaje ensamblador:

```
BSF STATUS,RP1    ;
BCF STATUS,RP0    ; Acceder al banco 2
MOVF ADDRESS,W    ; Mover la dirección al registro W
MOVWF EEADR       ; Escribir la dirección
BSF STATUS,RP0    ; Acceder al banco 3
BCF EECON1,EEPGD  ; Seleccionar la EEPROM
BSF EECON1,RD     ; Leer los datos
BCF STATUS,RP0    ; Acceder al banco 2
MOVF EEDATA,W     ; Dato se almacena en el registro W
```

La misma secuencia de programa escrita en C se parece a lo siguiente:

```
W = EEPROM_Read(ADDRESS);
```

Las ventajas del uso del lenguaje C se han hecho más obvias, no lo cree?

ESCRITURA EN LA MEMORIA EEPROM

Antes de escribir los datos en la memoria EEPROM es necesario escribir la dirección en el registro EESADR y los datos en el registro EESAT. Sólo ha quedado seguir a una secuencia especial para iniciar la escritura para cada byte. Durante el proceso de escritura las interrupciones deben estar deshabilitadas. El ejemplo que sigue muestra el procedimiento anteriormente descrito al escribir un programa en lenguaje ensamblador:

```
BSF STATUS,RP1
BSF STATUS,RP0
BTFSC EECON,WR1 ; Esperar a que se complete la escritura anterior
GOTO $-1 ;
BCF STATUS,RP0 ; Banco 2
MOVF ADDRESS,W ; Mover la dirección a W
MOVWF EEADR ; Escribir la dirección
MOVF DATA,W ; Mover los datos a W

MOVWF EEDATA ; Escribir los datos
BSF STATUS,RP0 ; Banco 3
BCF EECON1,EEPGD ; Seleccionar la EEPROM
BSF EECON1,WREN ; Escritura a la EEPROM habilitada
BCF INCON,GIE ; Todas las interrupciones deshabilitadas

MOVLW 55h
MOVWF EECON2
MOVLW AAh
MOVWF EECON2
BSF EECON1,WR

BSF INTCON,GIE ; Interrupciones habilitadas
BCF EECON1,WREN ; Escritura a la EEPROM deshabilitada
```

La misma secuencia de programa escrita en C se parece a lo siguiente:

```
W = EEPROM_Write(ADDRESS, W);
```

No hace falta comentar nada. **Vamos a hacerlo en mikroC...**

```
// El ejemplo muestra cómo utilizar la librería EEPROM en el compilador mikroC PRO for PIC.
```

```
char ii; // La variable ii utilizada en el bucle
```

```

void main(){
    ANSEL = 0;                // Configuración de los pines AN como E/S digitales
    ANSELH = 0;
    PORTB = 0;
    PORTC = 0;
    PORTD = 0;
    TRISB = 0;
    TRISC = 0;
    TRISD = 0;

    for(ii = 0; ii < 32; ii++)    // Llenar el búfer con los datos
        EEPROM_Write(0x80+ii, ii);    // Escribir los datos en la dirección 0x80+ii

    EEPROM_Write(0x02,0xAA);        // Escribir un dato en la dirección 2 de la EEPROM
    EEPROM_Write(0x50,0x55);        // Escribir un dato en la dirección 0x50

    // de la EEPROM

    Delay_ms(1000);                // Diodos en los puertos PORTB y PORTC
    PORTB = 0xFF;                  // para indicar el comienzo de la lectura
    PORTC = 0xFF;
    Delay_ms(1000);
    PORTB = 0x00;
    PORTC = 0x00;
    Delay_ms(1000);
    PORTB = EEPROM_Read(0x02);      // Leer los datos de la dirección 2 de la EEPROM y

    // visualizarla en el puerto PORB
    PORTC = EEPROM_Read(0x50);      // Leer los datos de la dirección 0x50 de la EEPROM y

    // visualizarla en el puerto PORC
    Delay_ms(1000);

    for(ii = 0; ii < 32; ii++) {    // Leer el bloque de 32 bytes de la dirección
        PORTD = EEPROM_Read(0x80+ii); // 0x80 y visualizarla en el puerto PORTD

    Delay_ms(250);
    }
}

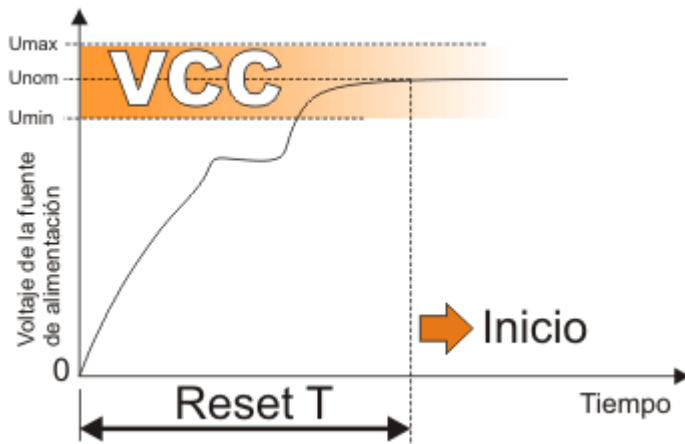
```

A primera vista, basta con encender una fuente de alimentación para hacer funcionar un microcontrolador. A primera vista, basta con apagar una fuente de alimentación para detenerlo. Sólo a primera vista. En realidad, el arranque y el final del funcionamiento son las fases críticas de las que se encarga una señal especial denominada RESET.

3.12 ¡Reinicio! ¿Black-out, Brown-out o Ruidos?

Al producirse un reinicio el microcontrolador detiene su funcionamiento inmediatamente y borra sus registros. Una señal de reinicio se puede generar externamente en cualquier momento (nivel lógico bajo en el pin MCLR). Si se necesita, una señal también puede ser generada por la lógica de control interna. Al encender una fuente de alimentación siempre se produce un reinicio. Por muchos eventos de transición que ocurren al encender una fuente de alimentación (centelleos y fogonazos de contactos eléctricos en interruptores, subida de voltaje lenta, estabilización de la frecuencia de señal de reloj graduada etc.) es necesario proporcionar un cierto tiempo muerto antes de que el microcontrolador se ponga a funcionar. Dos temporizadores internos PWRT y OST se encargan de eso. El PWRT puede estar habilitado/deshabilitado durante el proceso de escribir un programa. Vamos a ver cómo funciona todo.

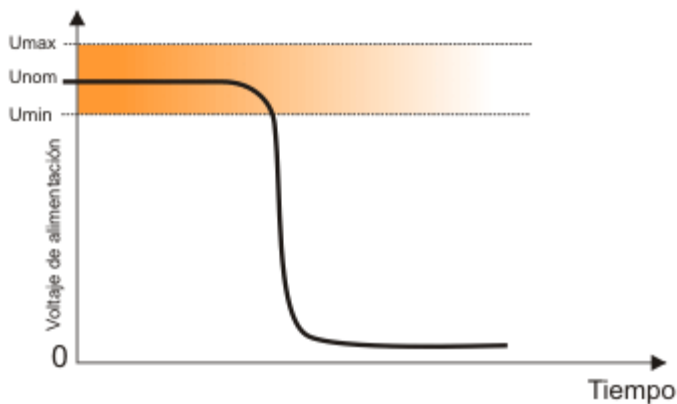
Cuando el voltaje de la fuente de alimentación alcanza entre 1.2 y 1.7V, un circuito denominado temporizador de arranque (Power-up timer) mantiene al microcontrolador reiniciado durante unos 72mS. Tan pronto como transcurra el tiempo, otro temporizador denominado temporizador de encendido del oscilador (Oscillator start-up timer) genera otra señal de reinicio durante la duración de 1024 períodos del oscilador de cuarzo. Al expirar el tiempo muerto (marcado con Reset T en la Figura) y al poner a alto el pin MCLR, todas las condiciones se han cumplido y el microcontrolador se pone a ejecutar la primera instrucción en el programa.



Aparte de este reinicio "controlado" que ocurre al encender una fuente de alimentación, hay dos tipos de reinicio denominados Black-out y Brown-out que pueden producirse durante el funcionamiento del microcontrolador así como al apagar una fuente de alimentación.

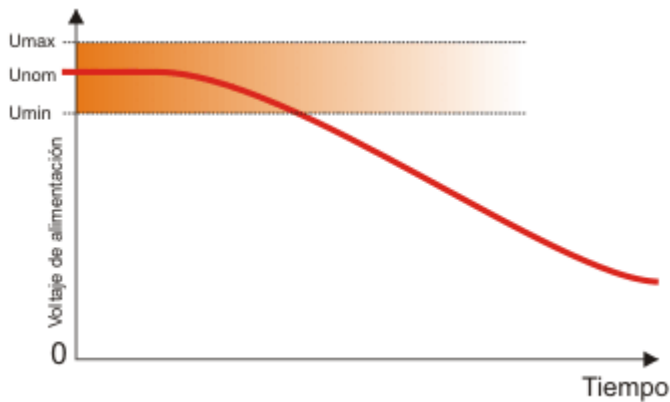
REINICIO BLACK-OUT

El reinicio black out ocurre al apagar una fuente de alimentación correctamente. El microcontrolador no tiene tiempo para hacer nada imprevisible puesto que el voltaje cae muy rápidamente por debajo de su valor mínimo. En otras palabras, ¡se apaga la luz, las cortinas bajan y el espectáculo ha terminado!



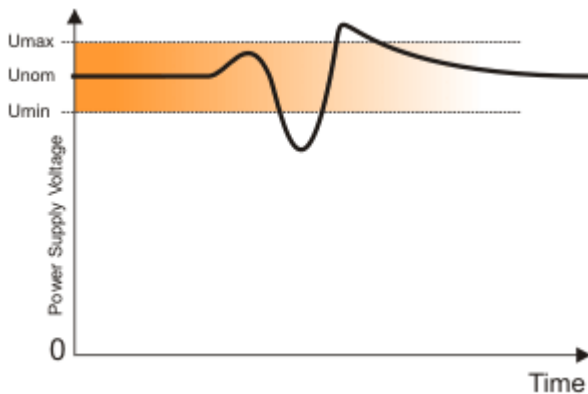
REINICIO BROWN-OUT

Cuando el voltaje de la fuente de alimentación cae lentamente (un ejemplo típico es descarga de baterías, aunque el microcontrolador experimentaría unas caídas mucho más rápidas como un proceso lento) los componentes internos detienen su funcionamiento gradualmente y ocurre el así llamado reinicio Brownout. En tal caso, antes de que el microcontrolador detenga su funcionamiento completamente, hay un peligro real de que los circuitos que funcionan a frecuencias altas se pongan a funcionar de forma imprevisible. El reinicio brown-out puede causar cambios fatales en el programa ya que se almacena en la memoria flash incorporada en el chip.



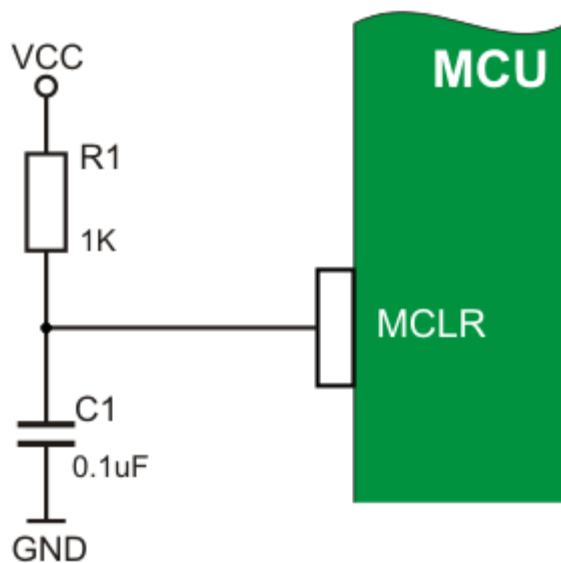
RUIDO ELÉCTRICO

Es un tipo especial del reinicio Brownout que ocurre en un ambiente industrial cuando voltaje de alimentación "parpadea" por un momento y cae por debajo del valor mínimo. Aunque es corto, este ruido producido en una línea de conducción eléctrica puede afectar desfavorablemente al funcionamiento del dispositivo.



PIN MCLR

Un cero lógico (0) al pin MCLR causa un reinicio inmediato y regular. Es recomendable conectarlo de la forma mostrada en la Figura a la derecha. La función de los componentes adicionales es de mantener un uno lógico "puro" durante el funcionamiento normal. Si sus valores se seleccionan de manera que proporcionen un nivel lógico alto en el pin después de que haya transcurrido el tiempo muerto reset T , el microcontrolador se pondrá a funcionar inmediatamente. Esto puede ser muy útil cuando se necesita sincronizar el funcionamiento del microcontrolador con los componentes adicionales o con el funcionamiento de varios microcontroladores.

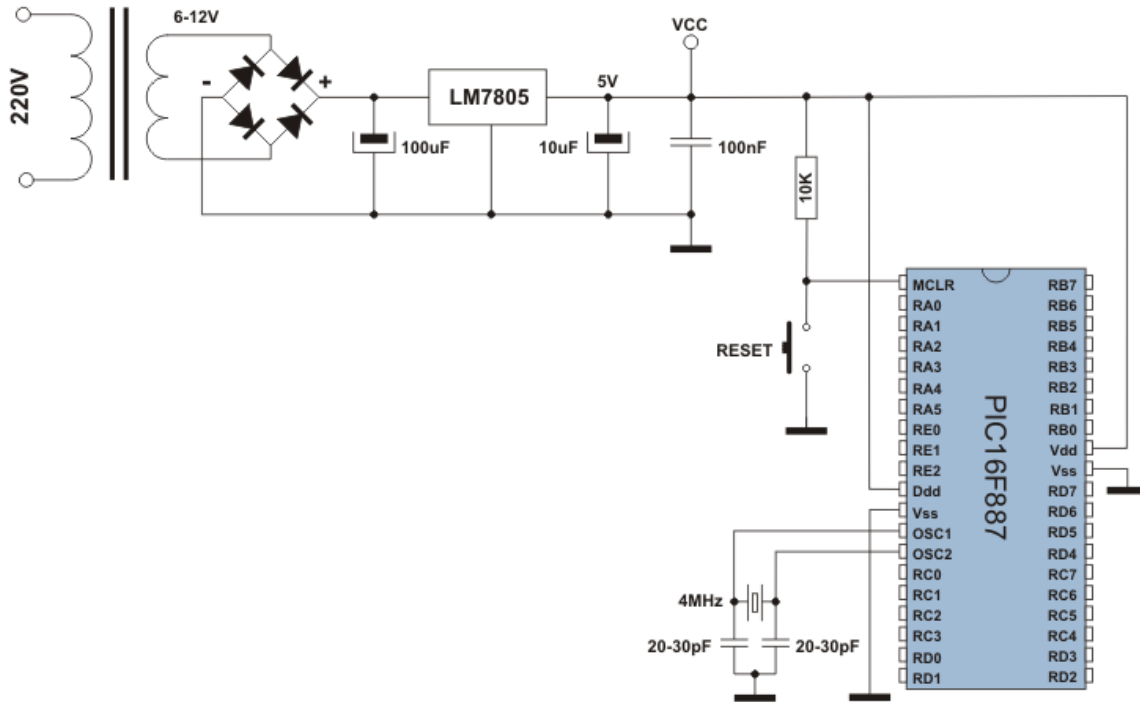


Para evitar posibles errores al producirse el reinicio Brown-out, el PIC 16F887 tiene un "mecanismo de protección" incorporado. Es un circuito simple, pero eficaz que reacciona cada vez que el voltaje de alimentación cae por debajo de 4V (si se utiliza un voltaje de alimentación de 5V) y mantiene este nivel de voltaje por más de 100 microsegundos. Este circuito genera una señal después de que todo el microcontrolador funcionará como si hubiera sido encendido por primera vez.

4.1 Conexión Básica

Para que un microcontrolador funcione apropiadamente es necesario proporcionar lo siguiente:

- Alimentación;
- Señal de reinicio; y
- Señal de reloj.



Como se muestra en la figura anterior, se trata de circuitos simples, pero no tiene que ser siempre así. Si el dispositivo destino se utiliza para controlar las máquinas caras o para mantener funciones vitales, todo se vuelve mucho más complicado.

ALIMENTACIÓN

Aunque el PIC16F887 es capaz de funcionar a diferentes voltajes de alimentación, no es recomendable probar la ley de Murphy. Lo más adecuado es proporcionar un voltaje de alimentación de 5V DC. Este circuito, mostrado en la página anterior, utiliza un regulador de voltaje positivo de tres terminales LM7805. Es un regulador integrado y barato que proporciona una estabilidad de voltaje de alta calidad y suficiente corriente para habilitar el

funcionamiento apropiado del controlador y de los periféricos (aquí suficiente significa una corriente de 1A).

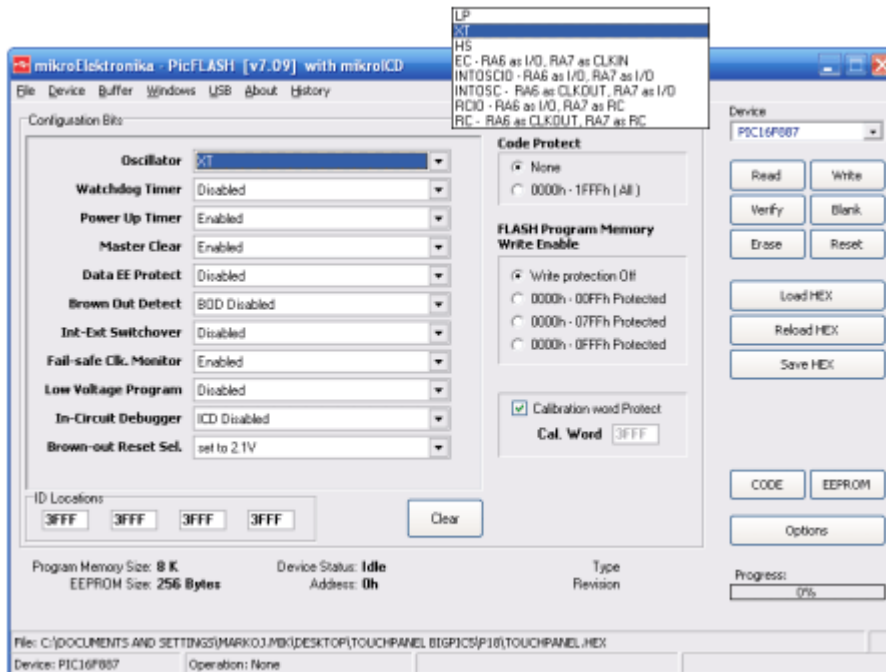
SEÑAL DE REINICIO

Para que un microcontrolador pueda funcionar apropiadamente, un uno lógico (VCC) se debe colocar en el pin de reinicio. El botón de presión que conecta el pin MCLR a GND no es necesario. Sin embargo, este botón casi siempre está proporcionado ya que habilita al microcontrolador volver al modo normal de funcionamiento en caso de que algo salga mal. Al pulsar sobre el botón RESET, el pin MCLR se lleva un voltaje de 0V, el microcontrolador se reinicia y la ejecución de programa comienza desde el principio. Una resistencia de 10k se utiliza para impedir un corto circuito a tierra al presionar este botón.

SEÑAL DE RELOJ

A pesar de tener un oscilador incorporado, el microcontrolador no puede funcionar sin componentes externos que estabilizan su funcionamiento y determinan su frecuencia (velocidad de operación del microcontrolador). Dependiendo de los elementos utilizados así como de las frecuencias el oscilador puede funcionar en cuatro modos diferentes:

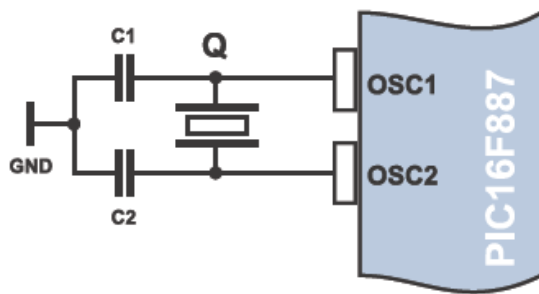
- LP - Cristal de bajo consumo;
- XT - Cristal / Resonador;
- HS - Cristal/Resonador de alta velocidad; y
- RC - Resistencia / Condensador.



¿Por qué son estos modos importantes? Como es casi imposible construir un oscilador estable que funcione a un amplio rango de frecuencias, el microcontrolador tiene que “saber” a qué cristal está conectado, para poder ajustar el funcionamiento de sus componentes internos. Ésta es la razón por la que todos los programas utilizados para escribir un programa en el chip contienen una opción para seleccionar el modo de oscilador. Vea la figura de la izquierda.

Cristal de cuarzo

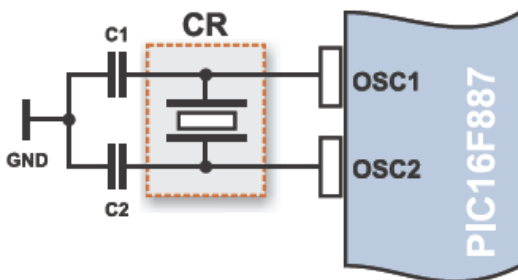
Al utilizar el cristal de cuarzo para estabilizar la frecuencia, un oscilador incorporado funciona a una frecuencia determinada, y no es afectada por los cambios de temperatura y de voltaje de alimentación. Esta frecuencia se etiqueta normalmente en el encapsulado del cristal. Aparte del cristal, los condensadores C1 y C2 deben estar conectados como se muestra en el siguiente esquema. Su capacitancia no es de gran importancia. Por eso, los valores proporcionados en la siguiente tabla se deben tomar como recomendación y no como regla estricta.



Modo	Frecuencia	C1, C2
LP	32 KHz	33pF
	200 KHz	15pF
XT	200 KHz	47-68 pF
	1 MHz	15 pF
	4 MHz	15 pF
HS	4 MHz	15 pF
	8 MHz	15-33 pF
	20 MHz	15-33 pF

Resonador cerámico

Un resonador cerámico es más barato y muy similar a un cuarzo por la función y el modo de funcionamiento. Por esto, los esquemas que muestran su conexión al microcontrolador son idénticos. No obstante, los valores de los condensadores difieren un poco debido a las diferentes características eléctricas. Refiérase a la tabla que está a continuación.



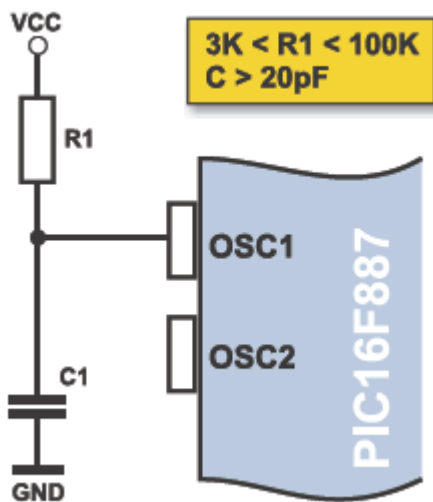
Modo	Frecuencia	C1, C2
XT	455 KHz	68-100 pF
	2 MHz	15-68 pF
	4 MHz	15-68 pF
HS	8 MHz	10-68 pF
	16 MHz	10-22 pF

Estos resonadores se conectan normalmente a los osciladores en caso de que no sea necesario proporcionar una frecuencia extremadamente precisa.

Oscilador RC

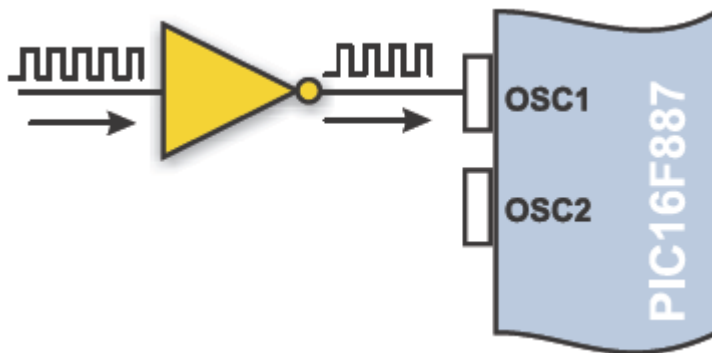
Si la frecuencia de operación no es de importancia, entonces no es necesario utilizar los componentes caros y adicionales para la estabilización. En vez de eso, basta con utilizar una simple red RC, mostrada en la siguiente figura. Como aquí es utilizada sólo la entrada del oscilador local, la señal de reloj con la frecuencia $F_{osc}/4$ aparecerá en el pin OSC2. Ésta es la

frecuencia de operación del microcontrolador, o sea la velocidad de ejecución de instrucciones.



Oscilador externo

Si se requiere sincronizar el funcionamiento de varios microcontroladores o si por alguna razón no es posible utilizar ninguno de los esquemas anteriores, una señal de reloj se puede generar por un oscilador externo. Refiérase a la siguiente figura.



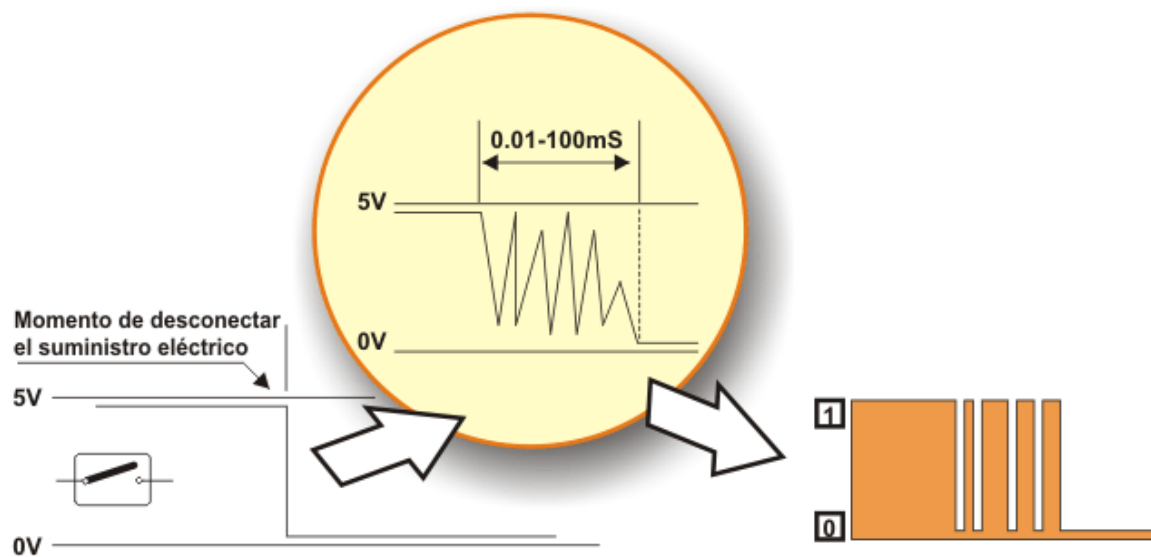
Apesar del hecho de que el microcontrolador es un producto de la tecnología moderna, no es tan útil sin estar conectado a los componentes adicionales. Dicho de otra manera, el voltaje llevado a los pines del microcontrolador no sirve para nada si no se utiliza para llevar a cabo ciertas operaciones como son encender/apagar, desplazar, visualizar etc.

4.2 Componentes Adicionales

Esta parte trata los componentes adicionales utilizados con más frecuencia en la práctica, tales como resistencias, transistores, diodos LED, visualizadores LED, visualizadores LCD y los circuitos de comunicación RS-232.

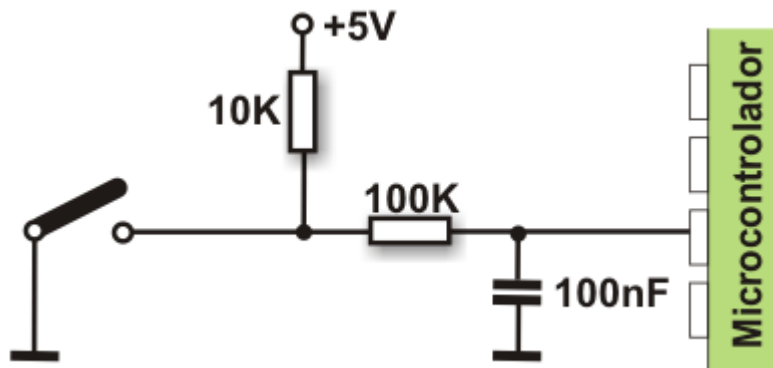
INTERRUPTORES Y BOTONES DE PRESIÓN

Los interruptores y los botones de presión son los dispositivos simples para proporcionar la forma más simple de detectar la aparición de voltaje en un pin de entrada del microcontrolador. No obstante, no es tan simple como parece... Es por un rebote de contacto. El rebote de contacto es un problema común en los interruptores mecánicos.

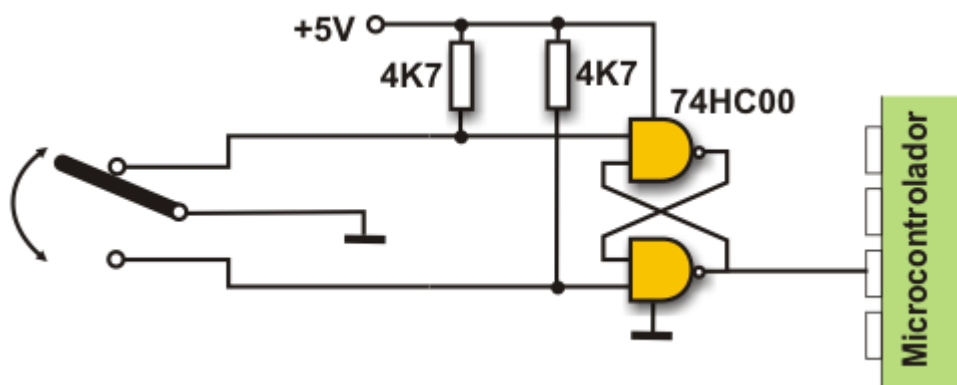


Al tocarse los contactos, se produce un rebote por su inercia y elasticidad. Por consiguiente, la corriente eléctrica es rápidamente pulsada en lugar de tener una clara transición de cero a la corriente máxima. Por lo general, esto ocurre debido a las vibraciones, los desniveles suaves y la suciedad entre los contactos. Este efecto no se percibe normalmente al utilizar estos componentes en la vida cotidiana porque el rebote ocurre demasiado rápido para afectar a la mayoría de los dispositivos eléctricos. Sin embargo, pueden surgir problemas en algunos circuitos lógicos que responden lo suficientemente rápido de manera que malinterpreten los pulsos producidos al tocarse los contactos como un flujo de datos. De todos modos, el proceso entero no dura mucho (unos pocos micro - o milisegundos), pero dura lo

suficiente para que lo detecte el microcontrolador. Al utilizar sólo un botón de presión como una fuente de señal de contador, en casi 100% de los casos ocurren los errores. El problema se puede resolver con facilidad al conectar un simple circuito RC para suprimir rápidos cambios de voltaje. Como el período del rebote no está definido, los valores de los componentes no están precisamente determinados. En la mayoría de los casos es recomendable utilizar los valores que se muestran en la siguiente figura.



Si se necesita una estabilidad completa, entonces hay que tomar medidas radicales. La salida del circuito, mostrado en la siguiente figura (biestable RS, también llamado flip flop RS), cambiará de estado lógico después de detectar el primer pulso producido por un rebote de contacto. Esta solución es más cara (interruptor SPDT), pero el problema es resuelto.



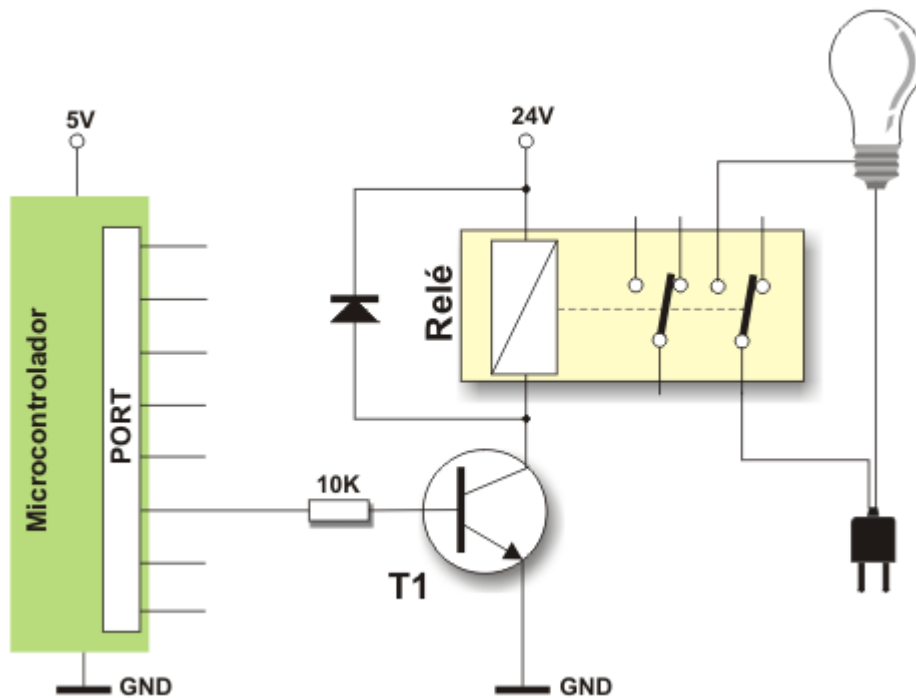
Aparte de estas soluciones de hardware, hay también una simple solución de software. Mientras el programa prueba el estado de circuito lógico de un pin de entrada, si detecta un cambio, hay que probarlo una vez más después de un cierto tiempo de retardo. Si el programa confirma el cambio, esto significa que un interruptor/botón de presión ha cambiado de posición. Las ventajas de esta solución son obvias: es gratuita, se borran los efectos del rebote de contacto y se puede aplicar a los contactos de una calidad más baja también.

RELÉ

Un relé es un interruptor eléctrico que se abre y se cierra bajo el control de otro circuito electrónico. Por eso está conectado a los pines de salida del microcontrolador y utilizado para encender/apagar los dispositivos de alto consumo tales como: motores, transformadores, calefactores, bombillas etc. Estos dispositivos se colocan casi siempre lejos de los componentes sensibles de la placa. Hay varios tipos de relés, pero todos funcionan de la misma manera. Al fluir la corriente por la bobina, el relé funciona por medio de un electromagneto, abriendo y cerrando uno o más conjunto de contactos. Similar a los optoacopladores no hay conexión galvánica (contacto eléctrico) entre los circuitos de entrada y salida. Los relés requieren con frecuencia tanto un voltaje más alto y una corriente más alta para empezar a funcionar. También hay relés miniatura que se pueden poner en marcha por una corriente baja obtenida directamente de un pin del microcontrolador.



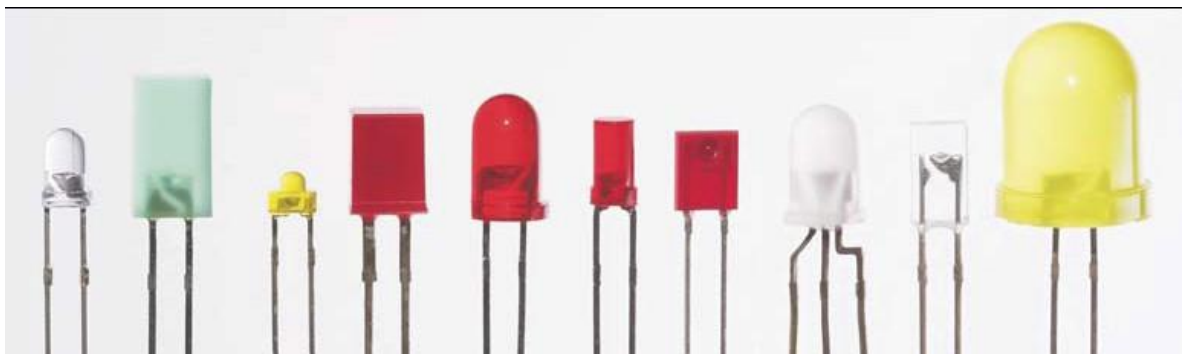
La figura que sigue muestra la solución utilizada con más frecuencia.



Para prevenir la aparición de un alto voltaje de autoinducción, causada por una parada repentina del flujo de corriente por la bobina, un diodo polarizado invertido se conecta en paralelo con la bobina. El propósito de este diodo es de “cortar” este pico de voltaje.

DIODOS LED

Probablemente sepa todo lo que necesita saber sobre los diodos LED, pero también debe pensar en los jóvenes... A ver, ¿cómo destruir un LED? Bueno...muy fácil.



Quemar con rapidez

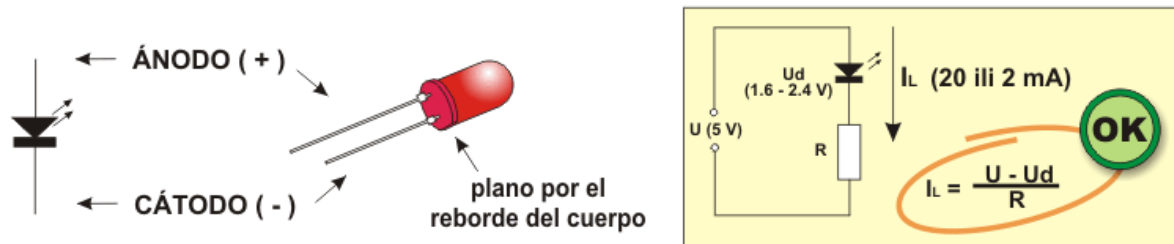
Como cualquier otro diodo, los LEDs tienen dos puntas - un ánodo y un cátodo. Conecte un diodo apropiadamente a la fuente de alimentación y va a emitir luz sin ningún problema. Ponga al diodo al revés y conéctelo a la misma fuente de alimentación (aunque sea por un momento). No emitirá luz - ¡nunca más!

Quemar lentamente

Hay un límite de corriente nominal, o sea, límite de corriente máxima especificada para cada LED que no se deberá exceder. Si eso sucede, el diodo emitirá luz más intensiva, pero sólo por un período corto de tiempo.

Algo para recordar

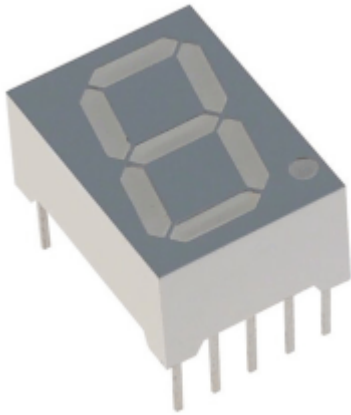
De manera similar, todo lo que tiene que hacer es elegir una resistencia para limitar la corriente mostrada a continuación. Dependiendo de voltaje de alimentación, los efectos pueden ser espectaculares.



VISUALIZADOR LED

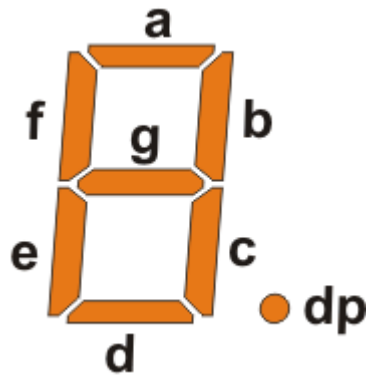
Básicamente, un visualizador LED no es nada más que varios diodos LED moldeados en la misma caja plástica. Hay varios tipos de los visualizadores y algunos de ellos están compuestos por varias docenas de diodos incorporados que pueden visualizar diferentes símbolos. No obstante, el visualizador utilizado con más frecuencia es el visualizador de 7 segmentos. Está compuesto por 8 LEDs. Los siete segmentos de un dígito están organizados en forma de un rectángulo para visualizar los símbolos, mientras que el segmento adicional se utiliza para el propósito de visualizar los puntos decimales. Para simplificar la conexión, los ánodos y los cátodos de todos los diodos se conectan al pin común así que tenemos visualizadores de ánodo común y visualizadores de cátodo común, respectivamente. Los segmentos están etiquetados con letras de a a g y dp, como se muestra en la siguiente figura. Al conectarlos, cada

diodes LED se trata por separado, lo que significa que cada uno dispone de su propia resistencia para limitar la corriente.

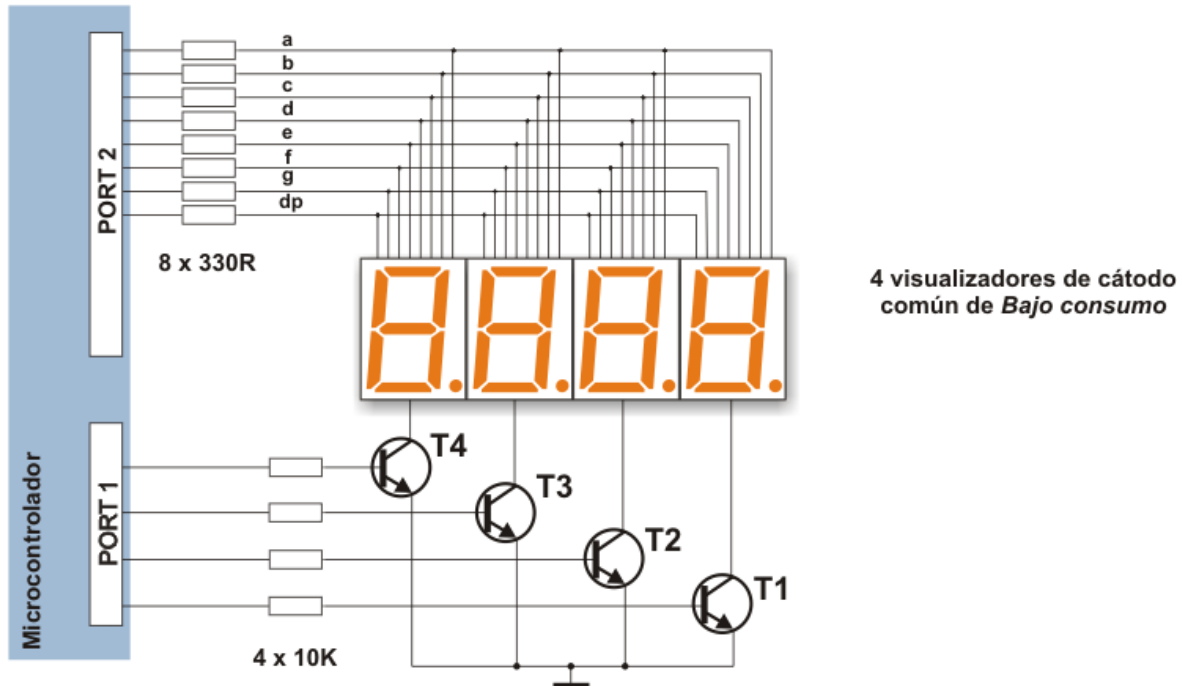


Aquí le presentamos unas cosas importantes a las que debe prestar atención al comprar un visualizador LED:

- Como hemos mencionado, dependiendo de si ánodos o cátodos están conectados al pin común, tenemos visualizadores de ánodo común y visualizadores de cátodo común. Visto de afuera, parece que no hay ninguna diferencia entre estos visualizadores, pues se le recomienda comprobar cuál se va a utilizar antes de instalarlo.
- Cada pin del microcontrolador tiene un límite de corriente máxima que puede recibir o dar. Por eso, si varios visualizadores están conectados al microcontrolador, es recomendable utilizar así llamados LEDs de Bajo consumo que utilizan solamente 2mA para su funcionamiento.
- Los segmentos del visualizador están normalmente etiquetados con letras de a a g, pero no hay ninguna regla estricta a cuáles pines del visualizador estarán conectados. Por eso es muy importante comprobarlo antes de empezar a escribir un programa o diseñar un dispositivo.

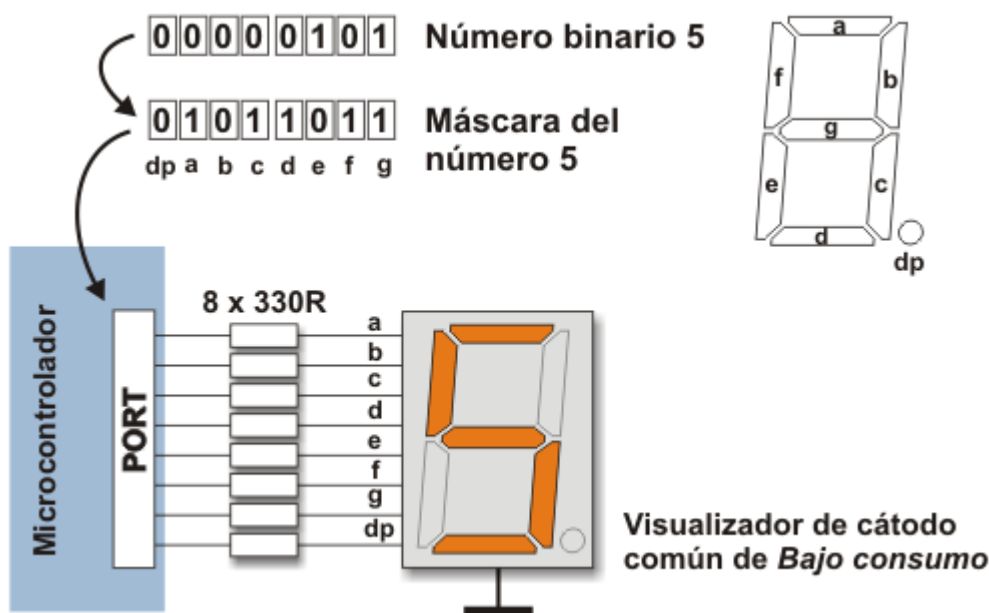


Los visualizadores conectados al microcontrolador normalmente ocupan un gran número de los pines de E/S valiosos, lo que puede ser un problema sobre todo cuando se necesita visualizar los números compuestos por varios dígitos. El problema se vuelve más obvio si, por ejemplo, se necesita visualizar dos números de seis dígitos (un simple cálculo muestra que en este caso se necesitan 96 pines de salida). La solución de este problema es denominada multiplexión. Aquí es cómo se ha hecho una ilusión óptica basada en el mismo principio de funcionamiento como una cámara de película. Un sólo dígito está activo a la vez, pero se tiene la impresión de que todos los dígitos de un número están simultáneamente activos por cambiar tan rápidamente de las condiciones de encendido/apagado.



Veamos la figura anterior. Primero se aplica un byte que representa unidades al puerto PORT2 del microcontrolador y se activa el transistor T1 a la vez. Después de poco tiempo, el transistor T1 se apaga, un byte que representa decenas se aplica al

puerto PORT2 y el transistor T2 se activa. Este proceso se está repitiendo cíclicamente a alta velocidad en todos los dígitos y transistores correspondientes. Lo decepcionante es que el microcontrolador es sólo un tipo de computadora miniatura diseñada para interpretar el lenguaje de ceros y unos, lo que se pone de manifiesto al visualizar cualquier dígito. Concretamente, el microcontrolador no conoce cómo son unidades, decenas, centenas, ni diez dígitos a los que estamos acostumbrados. Por esta razón, cada número a visualizar debe pasar por el siguiente procedimiento: Antes que nada, un número de varios dígitos debe ser dividido en unidades, centenas etc. en una subrutina específica. Luego, cada de estos dígitos se debe almacenar en los bytes particulares. Los dígitos se hacen reconocibles al realizar "enmascaramiento". En otras palabras, el formato binario de cada dígito se sustituye por una combinación diferente de los bits por medio de una subrutina simple. Por ejemplo, el dígito 8 (0000 1000) se sustituye por el número binario 0111 1111 para activar todos los LEDs que visualizan el número 8. El único diodo que queda inactivo aquí está reservado para el punto decimal. Si un puerto del microcontrolador está conectado al visualizador de tal manera que el bit 0 active el segmento 'a', el bit 1 active el segmento 'b', el bit 2 active el segmento 'c' etc, entonces la tabla que sigue muestra la "máscara" para cada dígito.



DÍGITOS A VISUALIZAR	SEGMENTOS DEL VISUALIZADOR					
	dp	a	b	c	d	e

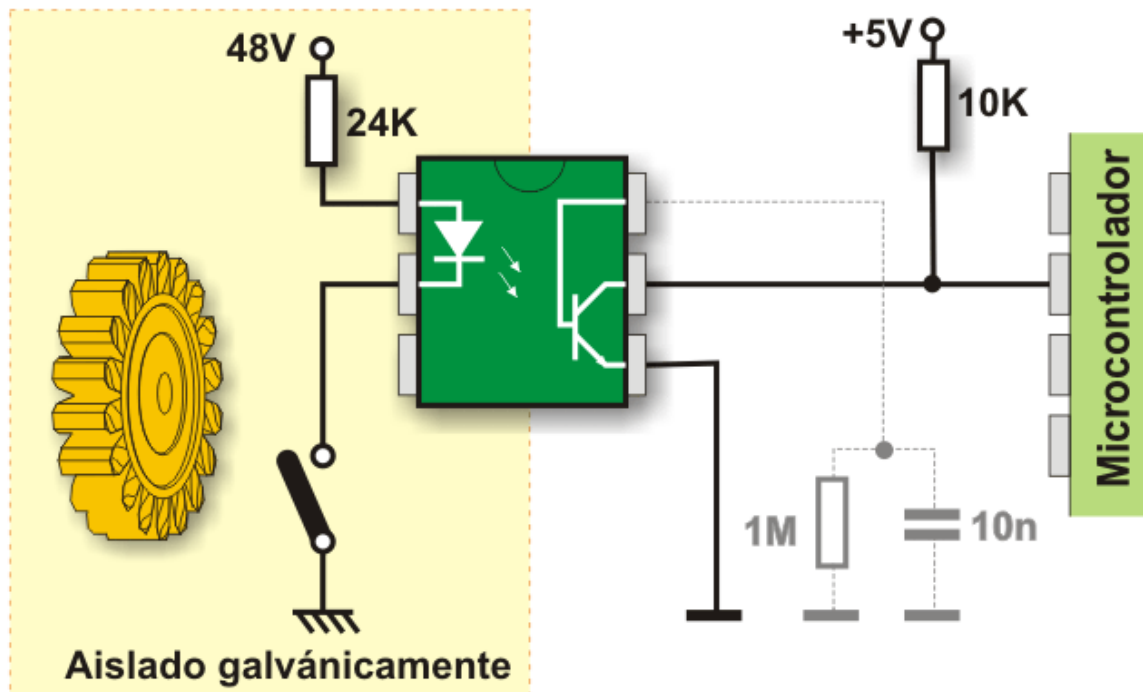
0	0	1	1	1	1	1
1	0	0	1	1	0	0
2	0	1	1	0	1	1
3	0	1	1	1	1	0
4	0	0	1	1	0	0
5	0	1	0	1	1	0
6	0	1	0	1	1	1
7	0	1	1	1	0	0
8	0	1	1	1	1	1
9	0	1	1	1	1	0

Además de los dígitos de 0 a 9, hay algunas letras -A, C, E, J, F, U, H, L, b, c, d, o, r, t - que se pueden visualizar al enmascarar. En caso de que se utilicen los visualizadores de ánodo común, todos los unos contenidos en la tabla anterior se deben sustituir por ceros y viceversa. Además, los transistores PNP se deben utilizar como controladores.

OPTOACOPLOADORES

Un optoacoplador es un dispositivo frecuentemente utilizado para aislar galvánicamente el microcontrolador de corriente o voltaje potencialmente peligroso de su entorno. Los optoacopladores normalmente disponen de una, dos o cuatro fuentes de luz (diodos LED) en su entrada mientras que en su salida, frente a los diodos, se encuentra el mismo número de los elementos sensibles a la luz (foto-transistores, foto-tiristores, foto-triacs). El punto es que un optoacoplador utiliza una

corta ruta de transmisión óptica para transmitir una señal entre los elementos de circuito, que están aislados eléctricamente. Este aislamiento tiene sentido sólo si los diodos y los elementos foto-sensitivos se alimentan por separado. Así, el microcontrolador y los componentes adicionales y caros están completamente protegidos de alto voltaje y ruidos que son la causa más frecuente de destrucción, daño y funcionamiento inestable de los dispositivos electrónicos en la práctica. Los optoacopladores utilizados con más frecuencia son aquéllos con foto-transistores en sus salidas. En los optoacopladores con la base conectada al pin 6 interno (también hay optoacopladores sin ella), la base puede quedarse desconectada.

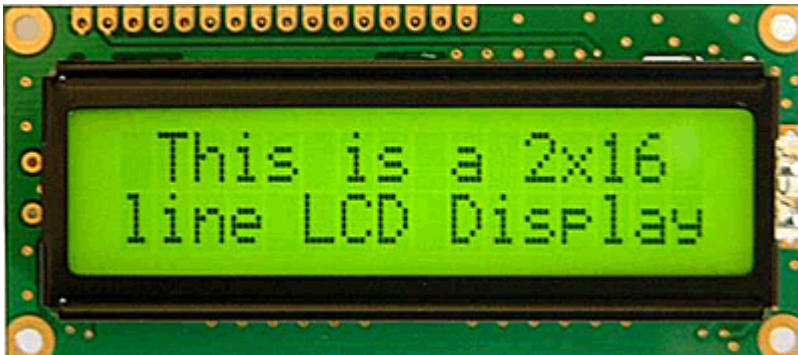


La red R/C representada por una línea quebrada en la figura anterior indica una conexión opcional de la base de transistores dentro del optoacoplador, que reduce los efectos de ruidos al eliminar los pulsos muy cortos.

VISUALIZADOR LCD

Este componente está específicamente fabricado para ser utilizado con los microcontroladores, lo que significa que no se puede activar por los circuitos integrados estándar. Se utiliza para visualizar los diferentes mensajes en un visualizador de cristal líquido miniatura. El modelo descrito aquí es el más utilizado en

la práctica por su bajo precio y grandes capacidades. Está basado en el microcontrolador HD44780 (Hitachi) integrado y puede visualizar mensajes en dos líneas con 16 caracteres cada una. Puede visualizar todas las letras de alfabeto, letras de alfabeto griego, signos de puntuación, símbolos matemáticos etc. También es posible visualizar símbolos creados por el usuario. Entre otras características útiles es el desplazamiento automático de mensajes (a la izquierda y a la derecha), aparición del cursor, retroiluminación LED etc.



Pines del visualizador LCD

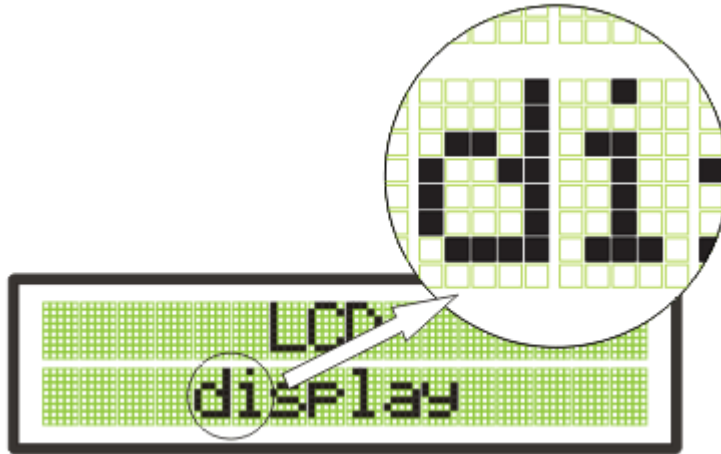
A lo largo de un lado de una placa impresa pequeña del visualizador LCD se encuentran los pines que le permiten estar conectado al microcontrolador. Hay 14 pines en total marcados con números (16 si hay retroiluminación). Su función se muestra en la tabla que sigue:

FUNCIÓN	NÚMERO	NOMBRE	ESTADO LÓGICO	DESCRIPCIÓN
Tierra	1	Vss	-	0V
Alimentación	2	Vdd	-	+5V
Contraste	3	Vee	-	0 - Vdd
Control de funcionamiento	4	RS	0 1	D0 – D7 considerados como comandos D0 – D7 considerados como datos

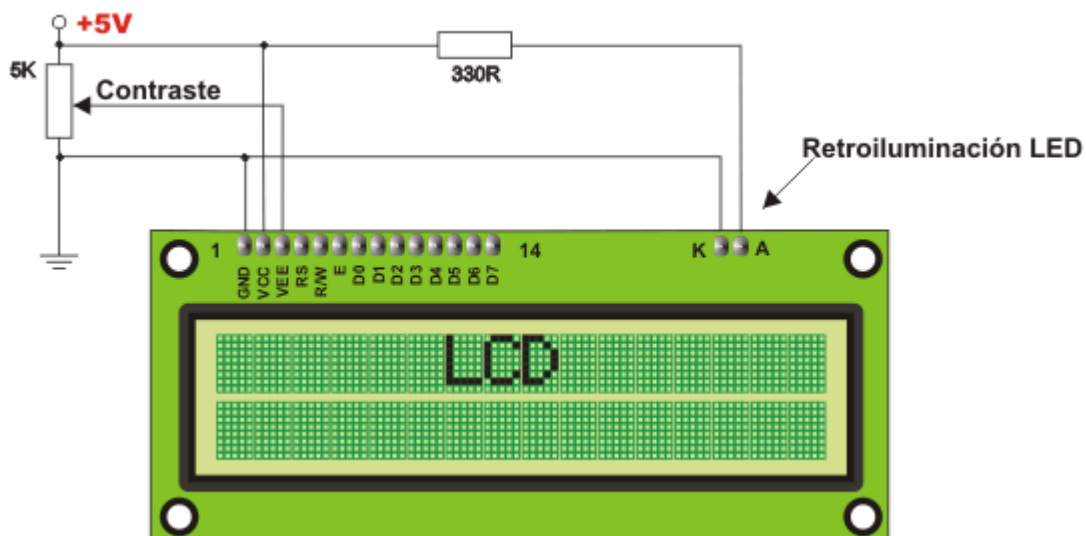
	5	R/W	0 1	Escribir los datos (del microcontrolador al LCD) Leer los datos (del LCD al microcontrolador)
	6	E	0 1 Transición de 1 a 0	Acceso al visualizador LCD deshabilitado Funcionamiento normal Datos/comandos se están transmitiendo al LCD
Datos / comandos	7	D0	0/1	Bit 0 LSB
	8	D1	0/1	Bit 1
	9	D2	0/1	Bit 2
	10	D3	0/1	Bit 3
	11	D4	0/1	Bit 4
	12	D5	0/1	Bit 5
	13	D6	0/1	Bit 6
	14	D7	0/1	Bit 7 MSB

Pantalla LCD

Una pantalla LCD puede visualizar dos líneas con 16 caracteres cada una. Cada carácter consiste en 5x8 o 5x11 píxeles. Este libro cubre un visualizador de 5x8 píxeles que es utilizado con más frecuencia.



El contraste del visualizador depende del voltaje de alimentación y de si los mensajes se visualizan en una o dos líneas. Por esta razón, el voltaje variable 0-V_{dd} se aplica al pin marcado como V_{ee}. Un potenciómetro trimer se utiliza con frecuencia para este propósito. Algunos de los visualizadores LCD tienen retroiluminación incorporada (diodos LED azules o verdes). Al utilizarlo durante el funcionamiento, se debe de conectar una resistencia en serie a uno de los pines para limitar la corriente (similar a diodos LED).



Si no hay caracteres visualizados o si todos los caracteres están oscurecidos al encender el visualizador, lo primero que se debe hacer es comprobar el potenciómetro para ajustar el contraste. ¿Está ajustado apropiadamente? Lo mismo se aplica si el modo de funcionamiento ha sido cambiado (escribir en una o en dos líneas).

Memoria LCD

El visualizador LCD dispone de tres bloques de memoria:

- DDRAM Display Data RAM (RAM de datos de visualización);
- CGRAM Character Generator RAM (generador de caracteres RAM); y
- CGROM Character Generator ROM (generador de caracteres ROM)

Memoria DDRAM

La memoria DDRAM se utiliza para almacenar los caracteres a visualizar. Tiene una capacidad de almacenar 80 caracteres. Algunas localidades de memoria están directamente conectadas a los caracteres en el visualizador. Todo funciona muy simple: basta con configurar el visualizador para incrementar direcciones automáticamente (desplazamiento a la derecha) y establecer la dirección inicial para el mensaje que se va a visualizar (por ejemplo 00 hex). Luego, todos los caracteres enviados por las líneas D0-D7 se van a visualizar en el formato de mensaje al que nos hemos acostumbrado - de la izquierda a la derecha. En este caso, la visualización empieza por el primer campo de la primera línea ya que la dirección inicial es 00hex. Si se envía más de 16 caracteres, todos se memorizarán, pero sólo los primeros 16 serán visibles. Para visualizar los demás, se debe utilizar el comando shift. Virtualmente, parece como si el visualizador LCD fuera una ventana, desplazándose de la izquierda a la derecha sobre las localidades de memoria con diferentes caracteres. En realidad, así es cómo se creó el efecto de desplazar los mensajes sobre la pantalla.

Memoria DDRAM



Si se habilita ver el cursor, aparecerá en la localidad actualmente direccionada. En otras palabras, si un carácter aparece en la posición del cursor, se va a mover automáticamente a la siguiente localidad direccionada. Esto es un tipo de memoria RAM así que los datos se pueden escribir en ella y leer de ella, pero su contenido se pierde irrecuperablemente al apagar la fuente de alimentación.

Memoria CGROM

La memoria CGROM contiene un mapa estándar de todos los caracteres que se pueden visualizar en la pantalla. A cada carácter se le asigna una localidad de memoria:

		4 bits más altos de la dirección															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
4 bits más bajos de la dirección	XXXX 0000	CG RAM (1)			0	Q	P	`	P					—	夕	Ξ	αp
	XXXX 0001	CG RAM (2)			!	1	A	Q	α	9				。	7	チ	4äq
	XXXX 0010	CG RAM (3)			"	2	B	R	b	r				「	イ	ツ	×βθ
	XXXX 0011	CG RAM (4)			#	3	C	S	c	s				」	ウ	テ	εω
	XXXX 0100	CG RAM (5)			\$	4	D	T	d	t				、	I	ト	トμΩ
	XXXX 0101	CG RAM (6)			%	5	E	U	e	u				・	オ	ナ	1εÜ
	XXXX 0110	CG RAM (7)			&	6	F	V	f	v				ヲ	カ	ニ	ヨρΣ
	XXXX 0111	CG RAM (8)			'	7	G	W	g	w				ア	キ	ヌ	うgπ
	XXXX 1000	CG RAM (1)			(8	H	X	h	x				イ	ウ	ホ	リJ又
	XXXX 1001	CG RAM (2))	9	I	Y	i	y				オ	ウ	ル	リ'y
	XXXX 1010	CG RAM (3)			*	:	J	Z	j	z				エ	コ	ン	ルjチ
	XXXX 1011	CG RAM (4)			+	:	K	L	k	l				オ	サ	ヒ	ロ*五
	XXXX 1100	CG RAM (5)			,	<	L	¥	1	l				カ	シ	ワ	ワΦ円
	XXXX 1101	CG RAM (6)			—	=	M	J	m	}				ユ	ヌ	ン	ト÷
	XXXX 1110	CG RAM (7)			。	>	N	^	n	≠				ヨ	セ	ホ	°ん
	XXXX 1111	CG RAM (8)			/	?	O	_	o	€				ッ	ウ	マ	°ö■

Las direcciones de las localidades de memoria CGROM corresponden a los caracteres ASCII. Si el programa que se está actualmente ejecutando llega al comando 'enviar el carácter P al puerto', el valor binario 0101 0000 aparecerá en el puerto. Este valor es el equivalente ASCII del carácter P. Al escribir este valor en un LCD, se visualizará el símbolo de la localidad 0101 0000 de la CGROM. En otras palabras, se visualizará el carácter P. Esto se aplica a todas las letras del alfabeto (minúsculas y mayúsculas), pero no se aplica a los números. Como se muestra en el mapa anterior, las direcciones de todos los dígitos se desplazan por 48 en relación con sus valores (dirección del dígito 0 es 48, dirección del dígito 1 es 49, dirección del dígito 2 es 50 etc.). Por consiguiente, para visualizar los dígitos correctamente es necesario añadir el número decimal 48 a cada uno antes de enviarlos a un LCD.

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
0 0 NUL	16 10 DLE	32 20 (space)	48 30 0
1 1 SOH	17 11 DC1	33 21 !	49 31 1
2 2 STX	18 12 DC2	34 22 "	50 32 2
3 3 ETX	19 13 DC3	35 23 #	51 33 3
4 4 EOT	20 14 DC4	36 24 \$	52 34 4
5 5 ENQ	21 15 NAK	37 25 %	53 35 5
6 6 ACK	22 16 SYN	38 26 &	54 36 6
7 7 BEL	23 17 ETB	39 27 '	55 37 7
8 8 BS	24 18 CAN	40 28 (56 38 8
9 9 TAB	25 19 EM	41 29)	57 39 9
10 A LF	26 1A SUB	42 2A *	58 3A :
11 B VT	27 1B ESC	43 2B +	59 3B ;
12 C FF	28 1C FS	44 2C ,	60 3C <
13 D CR	29 1D GS	45 2D -	61 3D =
14 E SO	30 1E RS	46 2E .	62 3E >
15 F SI	31 1F US	47 2F /	63 3F ?

ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol	ASCII Hex Symbol
64 40 @	80 50 P	96 60 ^	112 70 p
65 41 A	81 51 Q	97 61 a	113 71 q
66 42 B	82 52 R	98 62 b	114 72 r
67 43 C	83 53 S	99 63 c	115 73 s
68 44 D	84 54 T	100 64 d	116 74 t
69 45 E	85 55 U	101 65 e	117 75 u
70 46 F	86 56 V	102 66 f	118 76 v
71 47 G	87 57 W	103 67 g	119 77 w
72 48 H	88 58 X	104 68 h	120 78 x
73 49 I	89 59 Y	105 69 i	121 79 y
74 4A J	90 5A Z	106 6A j	122 7A z
75 4B K	91 5B [107 6B k	123 7B {
76 4C L	92 5C \	108 6C l	124 7C
77 4D M	93 5D]	109 6D m	125 7D }
78 4E N	94 5E ^	110 6E n	126 7E ~
79 4F O	95 5F _	111 6F o	127 7F

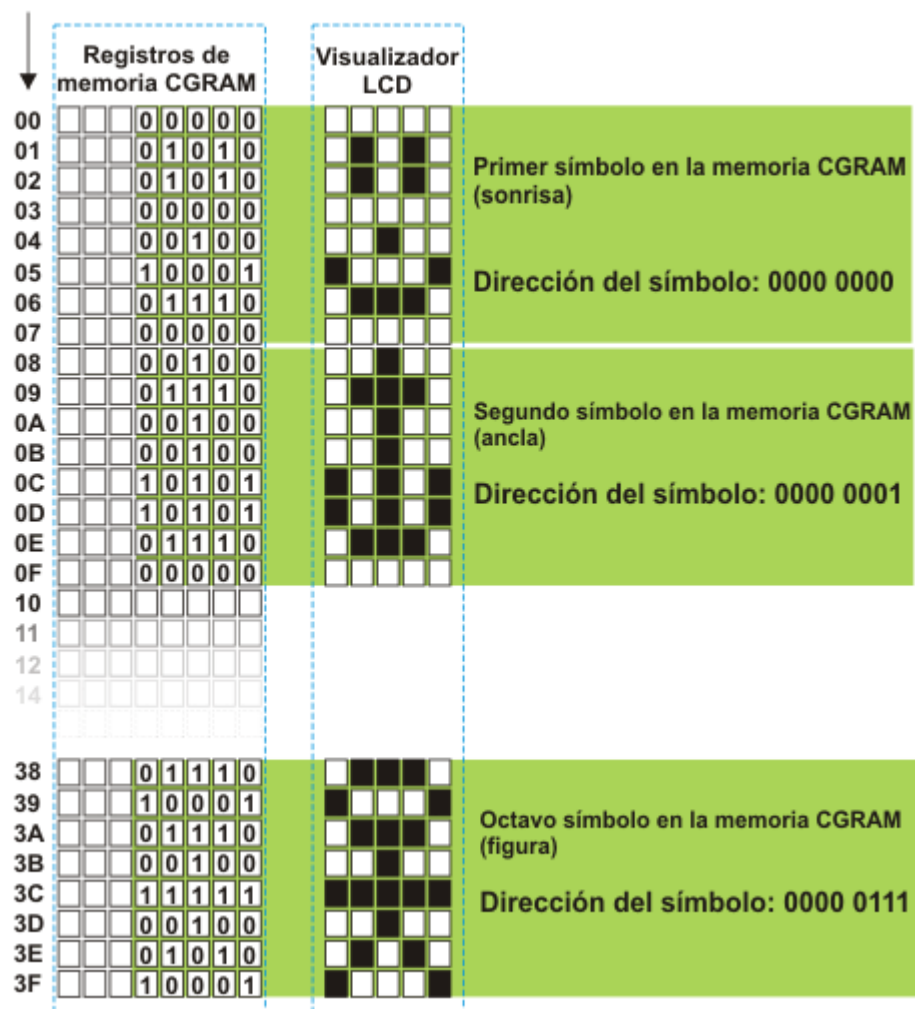
¿Qué es un código ASCII? Desde su aparición hasta hoy en día, las computadoras han sido capaces de reconocer solamente números, y no las letras. Esto significa que todos los datos que una computadora intercambia con un periférico, reconocidos

como letras por los humanos, en realidad están en el formato binario (el teclado es un buen ejemplo). En otras palabras, a cada carácter le corresponde la combinación única de ceros y unos. El código ASCII representa una codificación de caracteres basada en el alfabeto inglés. El ASCII especifica una correspondencia entre los símbolos de caracteres estándar y sus equivalentes numéricos.

Memoria CGRAM

Además de los caracteres estándar, el visualizador LCD puede visualizar símbolos definidos por el usuario. Esto puede ser cualquier símbolo de 5x8 píxeles. La memoria RAM denominada CGRAM de 64 bytes lo habilita. Los registros de memoria son de 8 bits de anchura, pero sólo se utilizan 5 bits más bajos. Un uno lógico (1) en cada registro representa un punto oscurecido, mientras que 8 localidades agrupados representan un carácter. Esto se muestra en la siguiente figura:

Direcciones hex. de los registros



Los símbolos están normalmente definidos al principio del programa por una simple escritura de ceros y unos de la memoria CGRAM así que crean las formas deseadas. Para visualizarlos basta con especificar su dirección. Preste atención a la primera columna en el mapa de caracteres CGROM. No contiene direcciones de la memoria RAM, sino los símbolos de los que se está hablando aquí. En este ejemplo 'visualizar 0' significa visualizar 'sonrisa', 'visualizar 1' significa - visualizar 'ancla' etc.

Comandos básicos del visualizador LCD

Todos los datos transmitidos a un visualizador LCD por las salidas D0-D7 serán interpretados como un comando o un dato, lo que depende del estado lógico en el pin RS:

- **RS = 1** - Los bits D0 - D7 son direcciones de los caracteres a visualizar. El procesador LCD direcciona un carácter del mapa de caracteres y lo visualiza. La dirección DDRAM especifica la localidad en la que se va a visualizar el carácter. Esta dirección se define antes de transmitir el carácter o la dirección del carácter anteriormente transmitido será aumentada automáticamente.
- **RS = 0** - Los bits D0 - D7 son los comandos para ajustar el modo del visualizador.

En la siguiente tabla se muestra una lista de comandos reconocidos por el LCD:

COMANDO	R S	R W	D 7	D 6	D 5	D 4	D3	D2	D1	D 0	TIEMPO DE EJECUCIÓ N
Borrar el visualizador	0	0	0	0	0	0	0	0	0	1	1.64mS
Poner el cursor al inicio	0	0	0	0	0	0	0	0	1	x	1.64mS
Modo de entrada	0	0	0	0	0	0	0	1	I/ D	S	40uS
Activar/desactivar el visualizador	0	0	0	0	0	0	1	D	U	B	40uS

Desplazar el cursor/visualizador	0	0	0	0	0	1	D/C	R/L	x	x	40uS
Modo de funcionamiento	0	0	0	0	1	D/L	N	F	x	x	40uS
Establecer la dirección CGRAM	0	0	0	1	Dirección CGRAM						40uS
Establecer la dirección DDRAM	0	0	1	Dirección CGRAM						40uS	
Leer la bandera "BUSY"(ocupado) (BF)	0	1	B F	Dirección CGRAM						-	
Escribir en la CGRAM o en la DDRAM	1	0	D 7	D 6	D 5	D 4	D3	D2	D1	D 0	40uS
Leer la CGRAM o la DDRAM	1	1	D 7	D 6	D 5	D 4	D3	D2	D1	D 0	40uS

I/D 1 = Incremento (por 1)
 0 = Decremento (por 1)
 a

R/L 1 = Desplazamiento a la derecha
 0 = Desplazamiento a la izquierda

S 1 = Desplazamiento del visualizador activado
 0 = Desplazamiento del visualizador desactivado

DL 1 = Bus de datos de 8 bits
 0 = Bus de datos de 4 bits

D 1 = Visualizador encendido
 0 = Visualizador apagado

N 1 = Visualizador de dos líneas
 0 = Visualizador en una línea

U 1 = Cursor activado
 0 = Cursor desactivado

F 1 = Carácter de 5x10 puntos
 0 = Carácter de 5x7 puntos

B 1 = Parpadeo del cursor encendido
0 = Parpadeo del cursor apagado

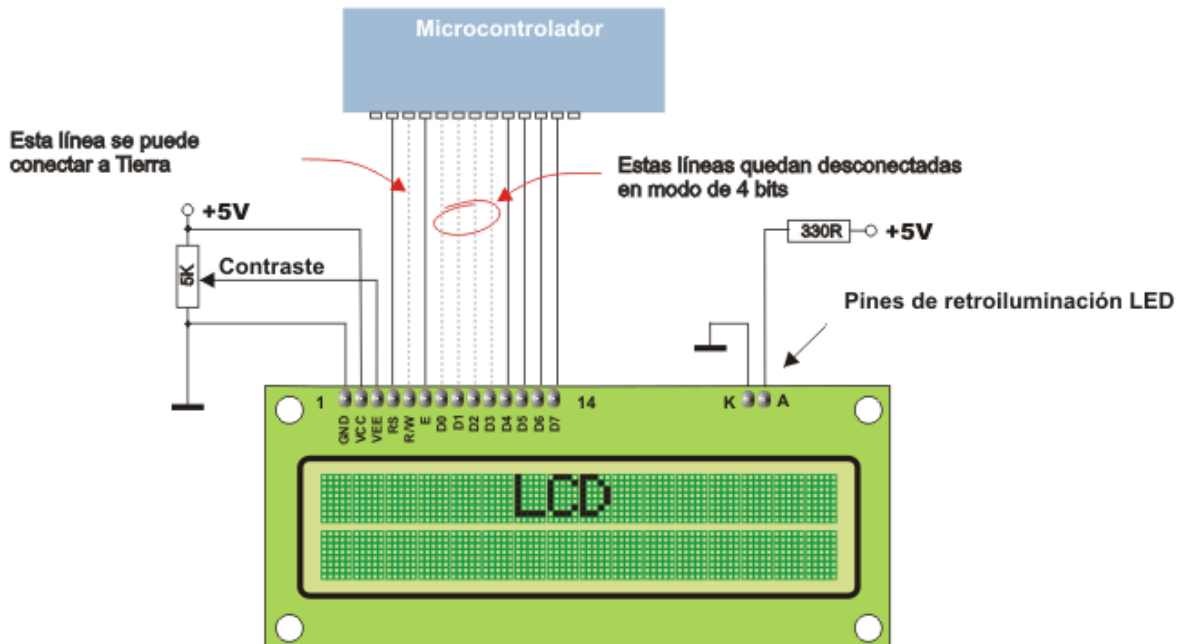
D/C 1 = Desplazamiento del visualiza
dor
0 = Desplazamiento del cursor

¿QUÉ ES UNA BANDERA DE OCUPADO (BUSY FLAG)?

En comparación al microcontrolador, el LCD es un componente extremadamente lento. Por esta razón, era necesario proporcionar una señal que, al ejecutar un comando, indicaría que el visualizador estaba listo para recibir el siguiente dato. Esta señal denominada bandera de ocupado (busy flag) se puede leer de la línea D7. El visualizador está listo para recibir un nuevo dato cuando el voltaje en esta línea es de 0V (BF=0).

Conectar al visualizador LCD

Dependiendo de cuántas líneas se utilizan para conectar un LCD al microcontrolador, hay dos modos de LCD, el de 8 bits y el de 4 bits. El modo apropiado se selecciona en el inicio del funcionamiento en el proceso denominado 'inicialización'. El modo de LCD de 8 bits utiliza los pines D0-D7 para transmitir los datos, como hemos explicado en la página anterior. El propósito principal del modo de LCD de 4 bits es de ahorrar los valiosos pines de E/S del microcontrolador. Sólo los 4 bits más altos (D4-D7) se utilizan para la comunicación, mientras que los demás pueden quedarse desconectados. Cada dato se envía al LCD en dos pasos - primero se envían 4 bits más altos (normalmente por las líneas D4- D7), y luego los 4 bits más bajos. La inicialización habilita que el LCD conecte e interprete los bits recibidos correctamente.



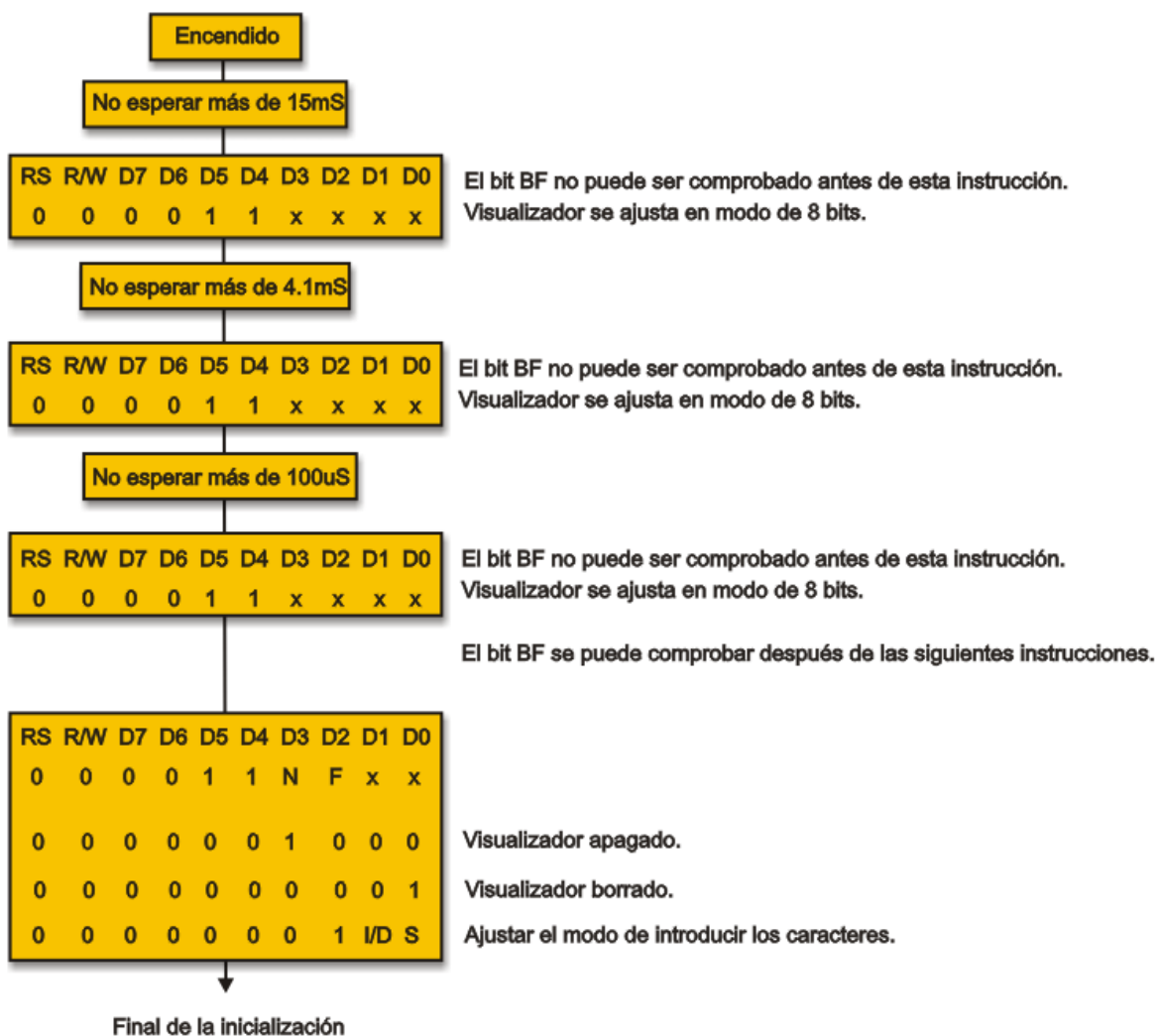
Pocas veces se leen los datos del LCD (por lo general se transmiten del microcontrolador al LCD) así que, con frecuencia, es posible guardar un pin de E/S de sobra. Es simple, basta con conectar el pin L/E a Tierra. Este "ahorro" del pin tiene su precio. Los mensajes se visualizarán normalmente, pero no será posible leer la bandera de ocupado ya que tampoco es posible leer los datos del visualizador. Afortunadamente, hay una solución simple. Después de enviar un carácter o un comando es importante dar al LCD suficiente tiempo para hacer su tarea. Debido al hecho de que la ejecución de un comando puede durar aproximadamente 1.64mS, el LCD tarda como máximo 2mS en realizar su tarea.

Inicializar al visualizador LCD

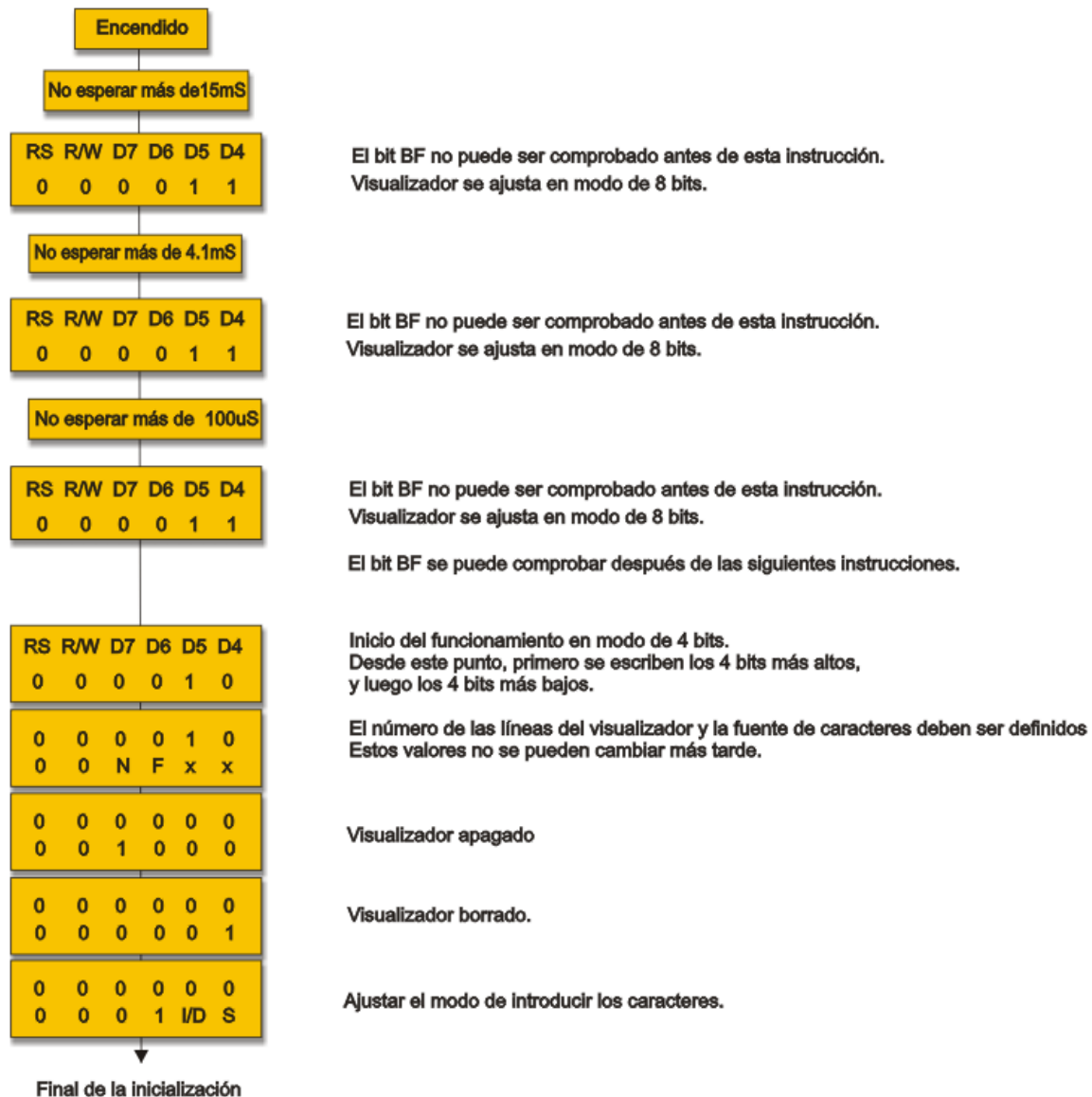
Al encender la fuente de alimentación, el LCD se reinicia automáticamente. Esto dura aproximadamente 15mS. Después de eso, el LCD está listo para funcionar. Asimismo, el modo de funcionamiento está configurado por defecto de la siguiente manera:

1. Visualizador está borrado.
2. Modo **DL** = 1 - Bus de datos de 8 bits **N** = 0 - LCD de una línea **F** = 0 - Carácter de 5 x 8 puntos
3. Visualizador/Cursor encendido/apagado **D** = 0 - Visualizador apagado **U** = 0 - Cursor apagado **B** = 0 - Parpadeo del cursor apagado
4. Introducción de caracteres **ID** = 1 Direcciones visualizadas se incrementan automáticamente en 1 **S** = Desplazamiento del visualizador desactivado

Por lo general, el reinicio automático se lleva a cabo sin problemas. ¡En la mayoría de los casos, pero no siempre! Si por cualquier razón, el voltaje de alimentación no llega a su máximo valor en 10mS, el visualizador se pone a funcionar de manera completamente imprevisible. Si la unidad de voltaje no es capaz de cumplir con las condiciones o si es necesario proporcionar un funcionamiento completamente seguro, se aplicará el proceso de inicialización. La inicialización, entre otras cosas, reinicia de nuevo al LCD, al habilitarle un funcionamiento normal. Hay dos algoritmos de inicialización. Cuál se utilizará depende de si la conexión al microcontrolador se realiza por el bus de datos de 4 o 8 bits. En ambos casos, después de inicialización sólo queda especificar los comandos básicos y, por supuesto, visualizar los mensajes. Refiérase a la Figura que sigue para el procedimiento de inicialización por el bus de datos de 8 bits:



¡Esto no es un error! En este algoritmo, el mismo valor se transmite tres veces en fila. El procedimiento de inicialización por el bus de datos de 4 bits:



Vamos a hacerlo en mikroC...

```
/* En mikroC for PIC, basta con escribir sólo una función para realizar todo el proceso
de la inicialización del LCD. Antes de llamar esta función es necesario declarar los
bits LCD_D4-LCD_D7, LCD_RS y LCD_EN. */
```

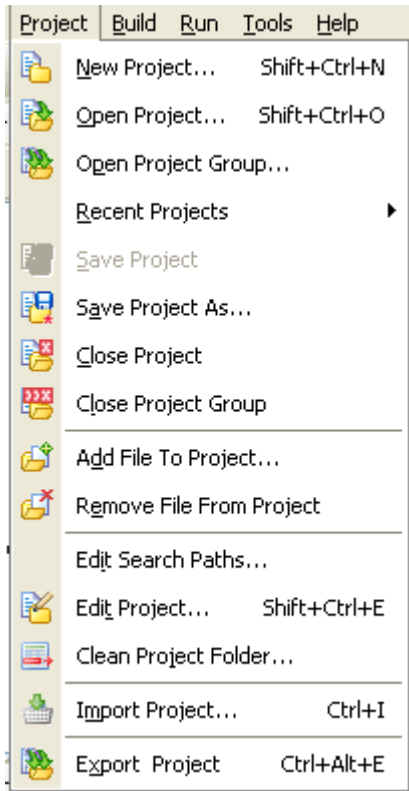
```
...
```

```
Lcd_Init(); // Inicializar el LCD
```

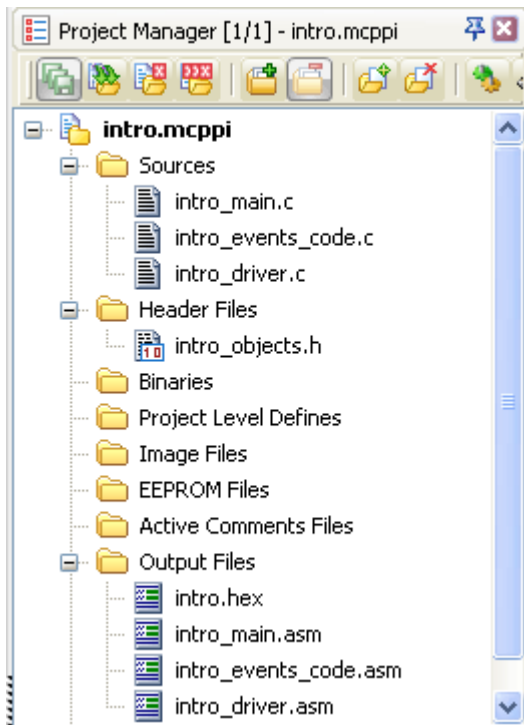
```
...
```

EJEMPLOS PRÁCTICOS

El proceso de crear un proyecto nuevo es muy simple. Seleccione la opción **New Project** del menú **Project** como se muestra en la Figura de la derecha.



Aparecerá una ventana que le guiará a través del proceso de creación de un proyecto nuevo. La ventana de entrada de este programa contiene una lista de acciones a realizar para crear un proyecto nuevo. Pulse el botón **Next**.



El proceso de creación de un proyecto nuevo consiste en cinco pasos:

1. Seleccione el tipo de microcontrolador a programar. En este caso se trata del PIC16F887.
2. Seleccione la frecuencia de reloj del microcontrolador. En este caso el valor seleccionado es 8 MHz.
3. Seleccione el nombre y la ruta del proyecto. En este caso, el nombre del proyecto es First_Project. Está guardado en la carpeta C:\My projects. Al nombre del proyecto se le asigna automáticamente la extensión .mcppi. Se creará en el proyecto el archivo fuente con el mismo nombre (First_Project.c.h).
4. Si el nuevo proyecto consiste de varios archivos fuente, se necesita especificarlos y pulse sobre el botón Add para incluirlos. En este ejemplo no hay archivos fuente adicionales.
5. Por último, se necesita confirmar todas las opciones seleccionadas. Pulse sobre Finish. Después de crear el proyecto, aparecerá una ventana blanca en la que debe escribir el programa. Vea la siguiente figura:

Introduzca el programa aquí

```
void Main () {  
  
}
```

Una vez creado el programa, es necesario compilarlo en un código .hex. Seleccione una de las opciones para compilar del menú **Project**:

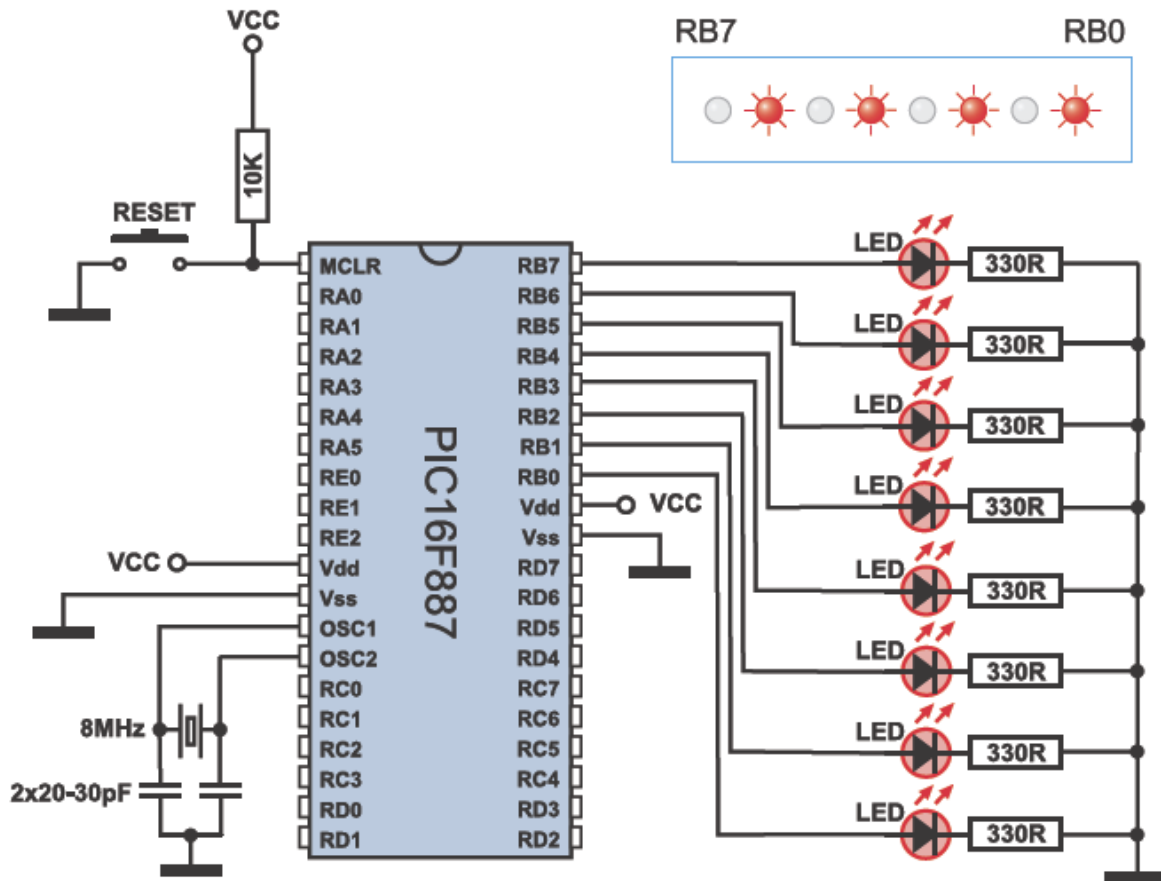
- Para crear un archivo .hex, seleccione la opción Build (Ctrl+F9) del menú Project o pulse sobre el icono Build de la barra de herramientas Project.
- Por medio de la opción Build All Projects (Shift+F9) se compilan todos los archivos del proyecto, librerías (si el código fuente contiene alguna de ellas) y los archivos def para el microcontrolador utilizado.
- La opción Build + Program (Ctrl+F11) es importante ya que permite al compilador mikroC PRO for PIC cargar automáticamente el programa en el microcontrolador después de la compilación. El proceso de la programación se realiza por medio del programador PICFlash.

Todos los errores encontrados durante la compilación aparecerán en la ventana Message. Si no hay errores en el programa, el compilador mikroC PRO for PIC generará los correspondientes archivos de salida.

4.3 Ejemplo 1

Escribir cabecera, configurar pines de E/S, utilizar la función Delay y el operador Switch

El único propósito de este programa es de encender varios diodos LED en el puerto B. Utilice este ejemplo para examinar cómo es un programa real. La siguiente figura muestra el esquema de conexión, mientras que el programa se encuentra en la siguiente página.



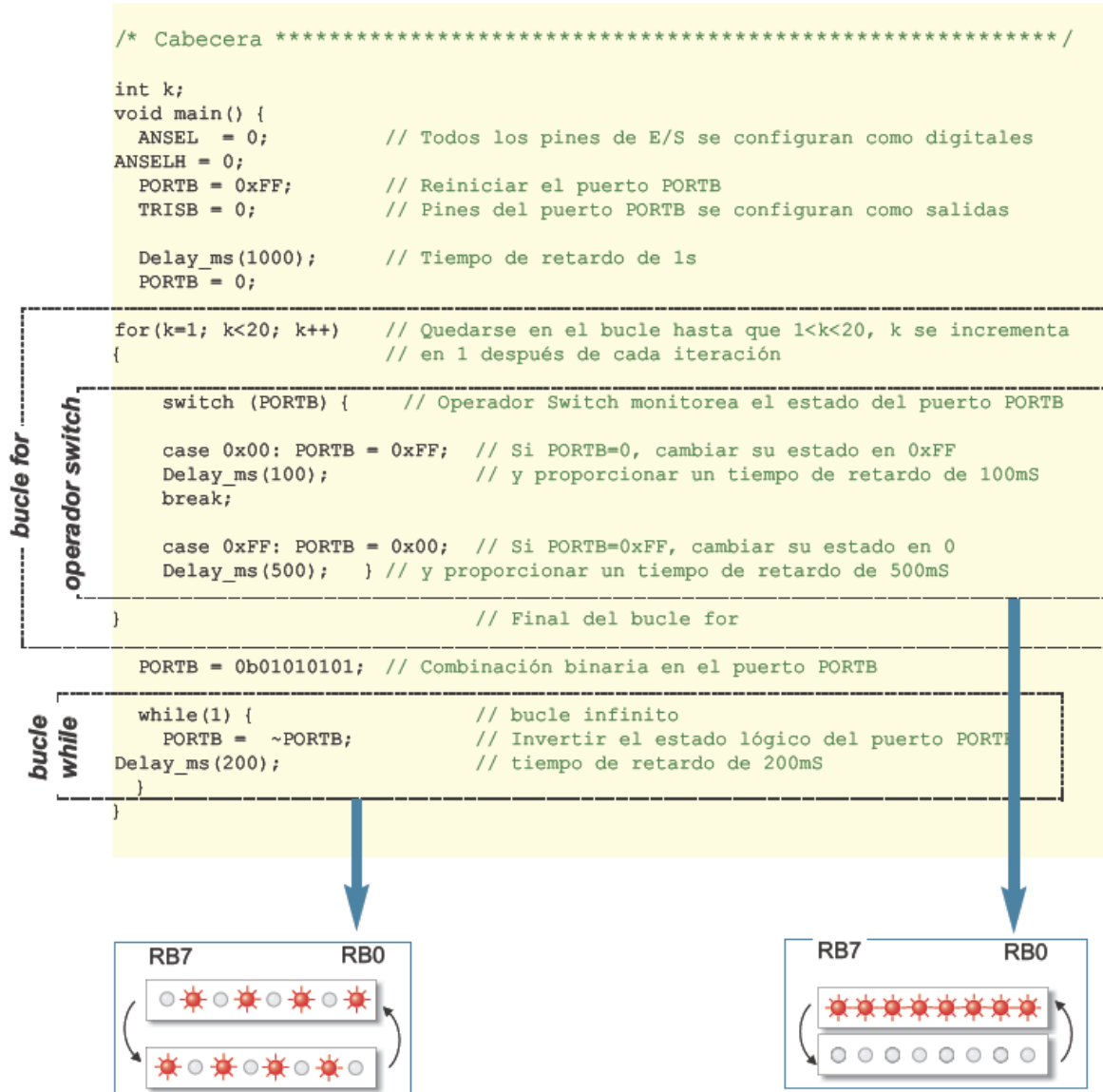
Al encender la fuente de alimentación, cada segundo, el diodo LED en el puerto B emite luz, lo que indica que el microcontrolador está conectado correctamente y que funciona normalmente. En este ejemplo se muestra cómo escribir una cabecera correctamente. Lo mismo se aplica a todos los programas descritos en este libro. Para no repetir, en los siguientes ejemplos no vamos a escribir la cabecera. Se considera estar en el principio de cada programa, marcada como "Cabecera".

Ejemplo 1

Cabecera	<pre>/* * Nombre de programa: * Ejemplo 1 * Derecho de autor: * (c) MikroElektronika, 2005-2010 * Descripción: * Esto es un simple programa utilizado para demostrar el funcionamiento del * microcontrolador. Cada segundo varios LED en el puerto PORTB estarán encendidos. * Configuración: * Microcontrolador: PIC16F887 * Dispositivo: EasyPIC6 * Oscilador: HS, 08.0000 MHz * SW: mikroC PRO v8.0 * Notas: - */</pre>	La cabecera se coloca en el principio del programa y proporciona informaciones básicas en forma de comentarios (nombre de programa, fecha de lanzamiento, etc.) No se haga ilusiones que en varios meses sabrá qué es lo que ha hecho el programa y por qué lo ha guardado. De eso se encargan los comentarios.
Ejecución de programa	<pre>void main() { ANSEL = 0; // Todos los pines de E/S se configuran como digitales ANSELH = 0; PORTB = 0b01010101; // Combinación binaria en el puerto PORTB TRISB = 0; // Pines del puerto PORTB se configuran como salidas }</pre>	

Para hacer este ejemplo más interesante, vamos a habilitar que los LEDs conectados al puerto PORTB parpadeen. Hay varios modos de hacerlo:

1. Tan pronto como se encienda el microcontrolador, todos los LEDs emitirán la luz por un segundo. La función Delay se encarga de eso en el programa. Sólo se necesita ajustar la duración del tiempo de retardo en milisegundos.
2. Después de un segundo, el programa entra en el bucle for, y se queda allí hasta que la variable k sea menor que 20. La variable se incrementa en 1 después de cada iteración. Dentro del bucle for, el operador switch monitorea el estado lógico en el puerto PORTB. Si PORTB=0xFF, su estado se invierte en 0x00 y viceversa. Cualquier cambio de estos estados lógicos hace todos los LEDs parpadear. El ciclo de trabajo es 5:1 (500mS:100mS).
3. Al salir del bucle for, el estado lógico del puerto POTRB cambia (0xb 01010101) y el programa entra en el bucle infinito while y se queda allí hasta que 1=1. El estado lógico del puerto PORTB se invierte cada 200mS.



4.4 Ejemplo 2

Utilizar instrucciones en ensamblador y el oscilador interno LFINTOSC...

En realidad, esto es una continuación del ejemplo anterior, pero se ocupa de un problema un poco más complicado... El propósito era hacer los LEDs en el puerto PORTB parpadear lentamente. Se puede realizar al introducir un valor suficiente grande para el parámetro del tiempo de retardo en la función Delay. No obstante, hay otra manera más eficiente para ejecutar el programa lentamente. Acuérdesse de que este microcontrolador tiene un oscilador incorporado LFINTOSC que funciona a una

frecuencia de 31kHz. Ahora llegó la hora de “darle una oportunidad”. El programa se inicia con el bucle do-while y se queda allí por 20 ciclos. Después de cada iteración, llega el tiempo de retardo de 100ms, indicado por un parpadeo relativamente rápido de los LEDs en el puerto PORTB. Cuando el programa salga de este bucle, el microcontrolador se inicia al utilizar el oscilador LFINTOSC como una fuente de señal de reloj. Los LEDs parpadean más lentamente aunque el programa ejecuta el mismo bucle do-while con un tiempo de retardo 10 veces más corto. Con el propósito de hacer evidentes algunas situaciones potencialmente peligrosas, se activan los bits de control por medio de las instrucciones en ensamblador. Dicho de manera sencilla, al entrar o salir una instrucción en ensamblador en el programa, el compilador no almacena los datos en un banco actualmente activo de la RAM, lo que significa que en esta sección de programa, la selección de bancos depende del registro SFR utilizado. Al volver a la sección de programa escrito en C, los bits de control RP0 y RP1 deben recuperar el estado que tenían antes de ‘la aventura en ensamblador’. En este programa, el problema se resuelve al utilizar la variable auxiliar saveBank, lo que guarda el estado de estos dos bits.

```
/* Cabecera *****/

int k = 0; // Variable k es de tipo int
char saveBank; // Variable saveBank es de tipo char

void main() {
    ANSEL = 0; // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTB = 0; // Todos los pines del puerto PORTB se ponen a 0
    TRISB = 0; // Pines del puerto PORTB se configuran como salidas

    do {
        PORTB = ~PORTB; // Invertir el estado lógico del puerto PORTB
        Delay_ms(100); // Tiempo de retardo de 100ms
        k++; // Incrementar k en 1
    }
    while(k<20); // Quedarse en bucle hasta que k<20

    k=0; // Reiniciar variable k
    saveBank = STATUS & 0b01100000; // Guardar el estado de los bits RP0 y RP1

    // (bits 5 y 6 del registro STATUS)
```

```

asm {                                     // Inicio de una secuencia en ensamblador
    bsf STATUS,RP0                       // Seleccionar el banco de memoria que contiene el
    bcf STATUS,RP1                       // registro OSCCON
    bcf OSCCON,6                         // Seleccionar el oscilador interno LFINTOSC
    bcf OSCCON,5                         // de frecuencia de 31KHz
    bcf OSCCON,4
    bsf OSCCON,0                         // Microcontrolador utiliza oscilador interno
}                                         // Final de la secuencia en ensamblador

STATUS &= 0b10011111;                  // Bits RP0 y RP1 recuperan el estado original
STATUS |= saveBank;

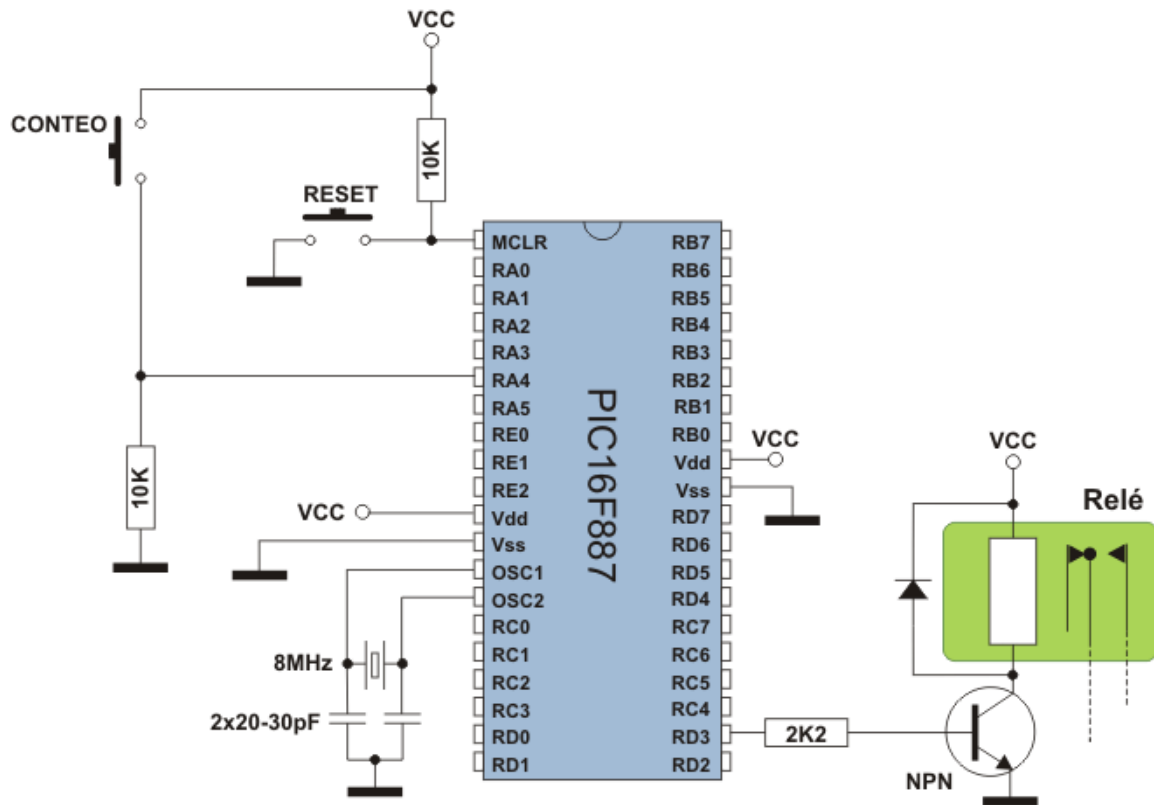
do {
    PORTB = ~PORTB;                     // Invertir el estado lógico del puerto PORTB
    Delay_ms(10);                       // Tiempo de retardo de 10 mS
    k++;                                 // Incrementar k en 1
}
while(k<20);                           // Quedarse en el bucle hasta que k<20

```

4.5 Ejemplo 3

Timer0 como un contador, declarar variables nuevas, constantes de enumeración, utilizar relés...

En cuanto a los ejemplos anteriores, el microcontrolador ha ejecutado el programa sin haber sido afectado de ninguna forma por su entorno. En la práctica, hay pocos dispositivos que funcionen de esta manera (por ejemplo, un simple controlador de luz de neón). Los pines de entrada se utilizan también en este ejemplo. En la siguiente figura se muestra un esquema, mientras que el programa está en la siguiente página. Todo sigue siendo muy simple. El temporizador Timer0 se utiliza como un contador. La entrada del contador está conectada a un botón de presión, así que cada vez que se presiona el botón, el temporizador Timer0 cuenta un pulso. Cuando el número de pulsos coincida con el número almacenado en el registro TEST, un uno lógico (5V) aparecerá en el pin PORTD.3. Este voltaje activa un relé electromecánico, y por eso este bit se le denomina 'RELÉ' en el programa.



/*Cabecera******/

```
void main() {  
    char TEST = 5;           // Constante TEST = 5  
    enum salidas {RELÉ = 3}; // Constante RELAY = 3  
  
    ANSEL = 0;               // Todos los pines de E/S se configuran como digitales  
    ANSELH = 0;  
    PORTA = 0;              // Reiniciar el puerto PORTA  
    TRISA = 0xFF;           // Todos los pines del puerto PORTA se configuran como entrada  
S  
    PORTD = 0;              // Reiniciar el puerto PORTD  
    TRISD = 0b11110111;     // Pin RD3 se configura como salida, mientras que los demás  
  
                               // se configuran como entradas  
    OPTION_REG.F5 = 1;      // Contador TMR0 recibe los pulsos por el pin RA4  
    OPTION_REG.F3 = 1;      // Valor del pre-escalador 1:1  
  
    TMR0 = 0;               // Reiniciar el temporizador/contador TMR0
```

```

do {
    if (TMR0 == TEST)      // ¿Coincide el número en el temporizador con la
                           // constante TEST?

        (PORTD.RELAY = 1); // Números coinciden. Poner el bit RD3 a uno (salida RELÉ)
    }
    while (1);              // Quedarse en el bucle infinito
}

```

Sólo una constante de enumeración RELÉ se utiliza en este ejemplo. Se le asigna un valor mediante la declaración.

```
enum salidas {RELÉ = 3}; // Constante RELÉ = 3
```

Si varios pines del puerto PORTD están conectados a los relés, la expresión anterior se puede escribir de la siguiente manera también:

```
enum salidas {RELÉ=3, CALENTADOR, MOTOR=6, SURTIDOR};
```

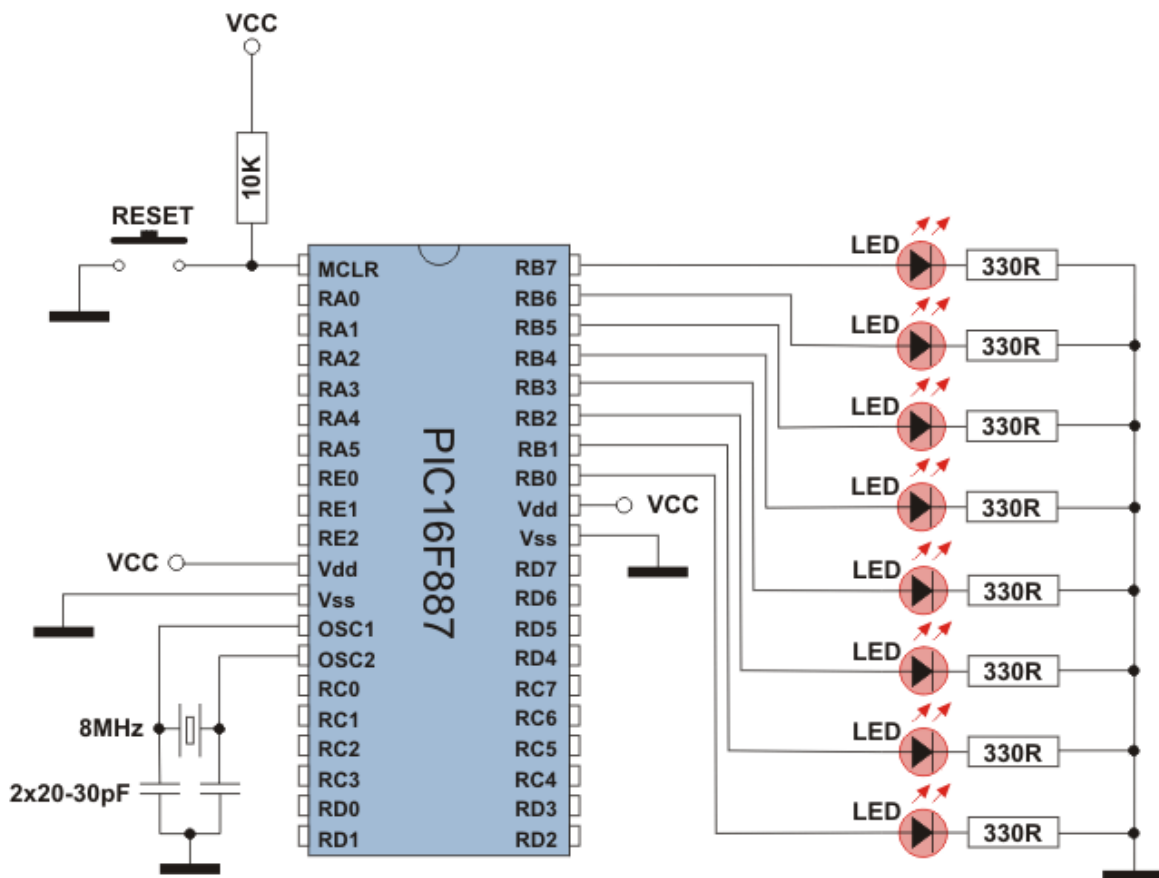
A todas las constantes, precedidas por las constantes con valores asignados (RELÉ=3 y MOTOR=6), se les asignan automáticamente los valores de las constantes precedentes, incrementados en 1. En este ejemplo, a las constantes CALENTADOR y SURTIDOR se les asignan los valores 4 y 7, es decir (CALENTADOR=4 y SURTIDOR=7), respectivamente.

4.6 Ejemplo 4

Utilizar los temporizadores Timer0, Timer1 y Timer2. Utilizar interrupciones, declarar nuevas funciones...

Al analizar los ejemplos anteriores, es probable que se haya fijado en la desventaja de proporcionar tiempo de retardo por medio de la función Delay. En estos casos, el microcontrolador se queda 'estático' y no hace nada. Simplemente espera que transcurra una cierta cantidad de tiempo. Tal pérdida de tiempo es un lujo inaceptable, por lo que se deberá aplicar otro método. ¿Se acuerda usted del capítulo de los temporizadores? ¿Se acuerda de lo de interrupciones? Este ejemplo los conecta de una manera práctica. El esquema se queda inalterada, y el desafío sigue siendo presente. Es necesario proporcionar un tiempo de retardo suficiente largo para darse

cuenta de los cambios en el puerto. Para este propósito se utiliza el temporizador Timer0 con el pre-escalador asignado. Siempre que se genere una interrupción con cada desbordamiento en el registro del temporizador, la variable cnt se aumenta automáticamente en 1 al ejecutarse cada rutina de interrupción. Cuando la variable llega al valor 400, el puerto PORTB se incrementa en 1. Todo el procedimiento se lleva a cabo “entre bastidores”, lo que habilita al microcontrolador hacer otra tarea.



```
/*Cabecera******/

unsigned cnt;          // Definir la variable cnt

void interrupt() {
    cnt++;              // Con una interrupción la cnt se incrementa en 1
    TMR0 = 96;          // El valor inicial se devuelve en el temporizador TMR0
    INTCON = 0x20;      // Bit T0IE se pone a 1, el bit T0IF se pone a 0
}

void main(){
```

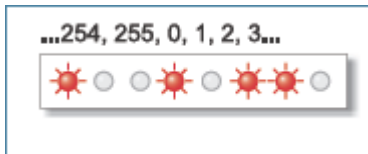
```

OPTION_REG = 0x84; // Pre-escalador se le asigna al temporizador TMR0
ANSEL = 0;         // Todos los pines de E/S se configuran como digitales
ANSELH = 0;
TRISB = 0;         // Todos los pines de puerto PORTB se configuran

                        // como salidas
PORTB = 0x0;       // Reiniciar el puerto PORTB
TMR0 = 96;          // Temporizador T0 cuenta de 96 a 255
INTCON = 0xA0;      // Habilitada interrupción TMR0
cnt = 0;            // A la variable cnt se le asigna un 0

do {                // Bucle infinito
    if (cnt == 400) { // Incrementar el puerto PORTB después 400 interrupciones
        PORTB = PORTB++; // Incrementar número en el puerto PORTB en 1
        cnt = 0;         // Reiniciar la variable cnt
    }
} while(1);
}

```



Siempre que se produzca un desbordamiento en el registro del temporizador TRM0, ocurre una interrupción.

```

/*Cabecera*****
*****

unsigned short cnt; // Definir la variable cnt

void interrupt() {
    cnt++ ;          // Con una interrupción la cnt se incrementa en 1
    PIR1.TMR1IF = 0; // Reiniciar el bit TMR1IF
    TMR1H = 0x80;    // El valor inicial se devuelve en los registros
    TMR1L = 0x00;    // del temporizador TMR1H y TMR1L
}

void main() {

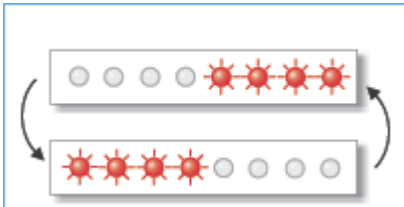
```

```

ANSEL = 0;           // Todos los pines de E/S se configuran como digitales
ANSELH = 0;
PORTB = 0xF0;        // Valor inicial de los bits del puerto PORTB
TRISB = 0;           // Pines del puerto PORTB se configuran como salidas
T1CON = 1;           // Configurar el temporizador TMR1
PIR1.TMR1IF = 0;      // Reiniciar el bit TMR1IF
TMR1H = 0x80;         // Ajustar el valor inicial del temporizador TMR1
TMR1L = 0x00;
PIE1.TMR1IE = 1;      // Habilitar la interrupción al producirse un desbordamiento
cnt = 0;              // Reiniciar la variable cnt
INTCON = 0xC0;        // Interrupción habilitada (bits GIE y PEIE)

do {                  // Bucle infinito
    if (cnt == 76) { // Cambiar el estado del puerto PORTB después de 76 interrupciones
        PORTB = ~PORTB; // Número en el puerto PORTB está invertido
        cnt = 0;        // Reiniciar la variable cnt
    }
} while (1);
}

```



En este caso, una interrupción se habilita después de que se produzca un desbordamiento en el registro del temporizador TMR1 (TMR1H, TMR1L). Además, la combinación de los bits que varía en el puerto POTRB difiere de la en el ejemplo anterior.

```

/*Cabecera*****
*****

unsigned short cnt; // Definir la variable cnt

void Reemplazar() {
    PORTB = ~PORTB; // Definir nueva función 'Reemplazar'
} // Función invierte el estado del puerto

```



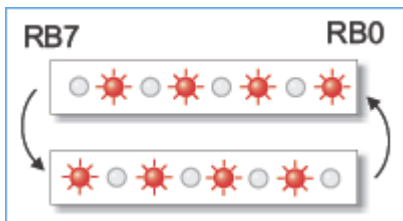
```

void interrupt() {
    if (PIR1.TMR2IF) { // Si el bit TMR2IF = 1,
        cnt++;          // Incrementar variable la cnt en 1
        PIR1.TMR2IF = 0; // Reiniciar el bit y
        TMR2 = 0;        // Reiniciar el registro TMR2
    }
}

// main
void main() {
    cnt = 0;              // Reiniciar la variable cnt
    ANSEL = 0;            // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTB = 0b10101010; // Estado lógico en los pines del puerto PORTB
    TRISB = 0;            // Todos los pines del puerto PORTB se configuran como salidas
    T2CON = 0xFF;          // Configurar el temporizador T2
    TMR2 = 0;              // Valor inicial del registro del temporizador TMR2
    PIE1.TMR2IE = 1;       // Interrupción habilitada
    INTCON = 0xC0;         // Bits GIE y PEIE se ponen a 1

    while (1) {            // Bucle infinito
        if (cnt > 30) {    // Cambiar el estado del puerto PORTB después de
                            // más de 30 interrupciones
            Reemplazar(); // Función Reemplazar invierte el estado del puerto PORTB
            cnt = 0;        // Reiniciar la variable cnt
        }
    }
}

```

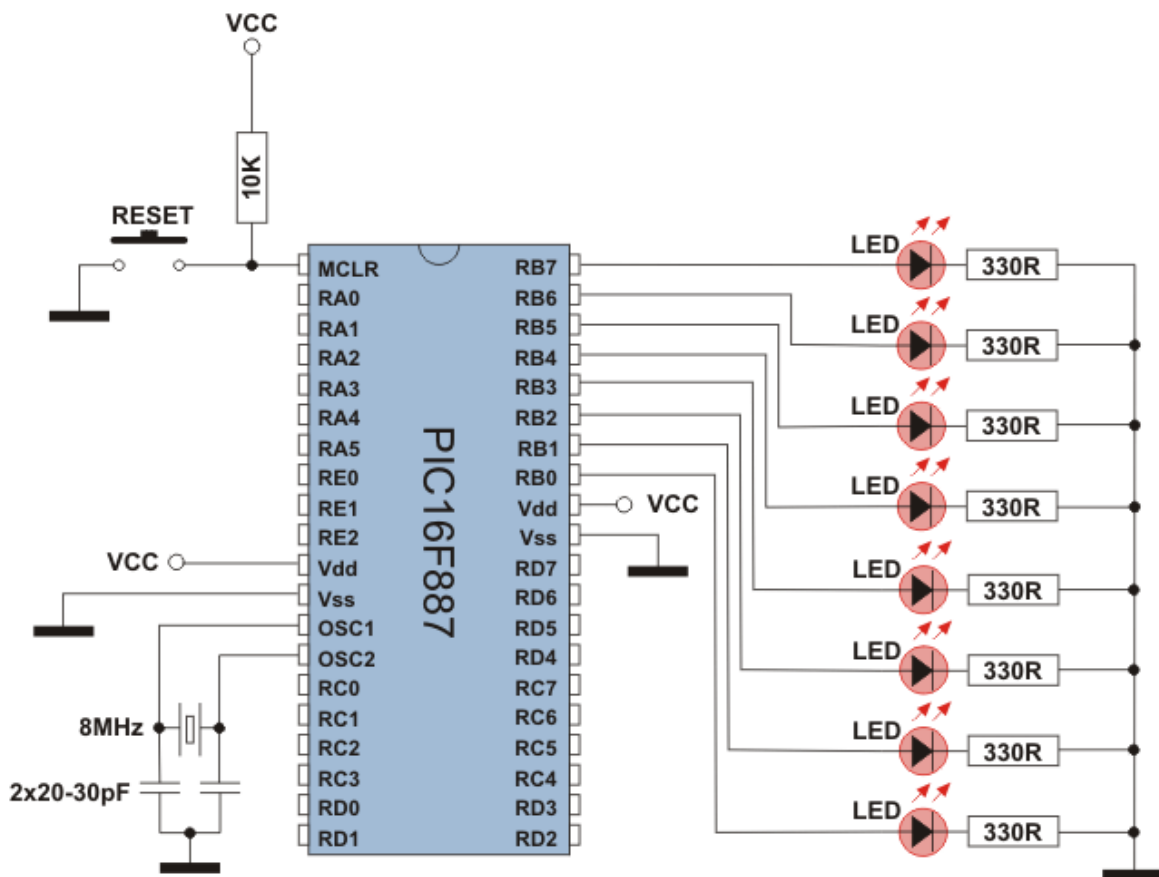


En este ejemplo, una interrupción ocurre después de que se produce un desbordamiento en el registro del temporizador TMR2. Para invertir el estado lógico de los pines del puerto se utiliza la función `Reemplazar`, que normalmente no pertenece al lenguaje C estándar.

4.7 Ejemplo 5

el temporizador perro - guardián

Este ejemplo muestra cómo NO se debe utilizar el temporizador perro-guardián. Un comando utilizado para reiniciar este temporizador se omite a propósito en el bucle del programa principal, lo que habilita al temporizador perro guardián “ganar la batalla del tiempo” y reiniciar al microcontrolador. Por consiguiente, el microcontrolador se va a reiniciar sin parar, lo que indicará el parpadeo de los LEDs del puerto PORTB.



```
/*Cabecera*****
```

```
void main() {
```

```
    OPTION_REG = 0x0E; // Pre-escalador se le asigna al temporizador WDT (1:64)
```

```
    asm CLRWDT;        // Comando en ensamblador para reiniciar el temporizador WDT
```

```
    PORTB = 0x0F;      // Valor inicial del registro PORTB
```

```
    TRISB = 0;         // Todos los pines del puerto PORTB se configuran como salidas
```

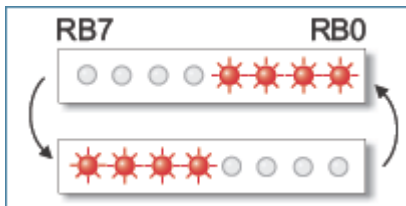
```

Delay_ms(300);    // Tiempo de retardo de 30mS
PORTB = 0xF0;     // Valor del puerto PORTB diferente del inicial

while (1);        // Bucle infinito. El programa se queda aquí hasta que el
                  // temporizador WDT reinicie al microcontrolador
}

```

Para que este ejemplo funcione apropiadamente, es necesario habilitar al temporizador perro-guardián al seleccionar la opción Tools/mE Programmer/Watchdog Timer - Enabled.



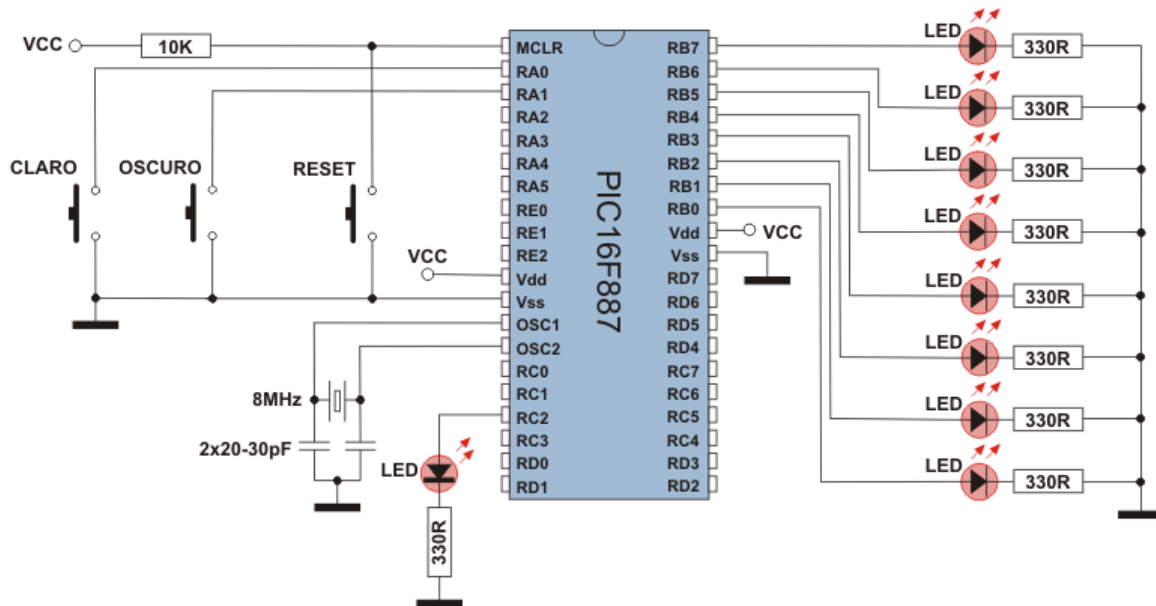
4.8 Ejemplo 6

Módulo CCP1 como generador de señal PWM

Este ejemplo muestra el uso del módulo CCP1 en modo PWM. Para hacer las cosas más interesantes, la duración de los pulsos en la salida P1A (PORTC,2) se puede cambiar por medio de los botones de presión simbólicamente denominados 'OSCURO' y 'CLARO'. La duración ajustada se visualiza como una combinación binaria en el puerto PORTB. El funcionamiento de este módulo está bajo el control de las funciones pertenecientes a la librería especializada PWM. Aquí se utilizan las tres de ellas:

1. **PWM1_init** tiene el prototipo: `void Pwm1_Init(long freq);` El parámetro `freq` ajusta la frecuencia de la señal PWM expresada en hercios. En este ejemplo equivale a 5kHz.
2. **PWM1_Start** tiene el prototipo: `void Pwm1_Start(void);`
3. **PWM1_Set_Duty** tiene el prototipo: `void Pwm1_Set_Duty(unsigned short duty_ratio);` El parámetro `duty_ratio` ajusta la duración de pulsos en una secuencia de pulsos.

La librería PWM también contiene la función `PWM_Stop` utilizada para deshabilitar este modo. Su prototipo es: `void Pwm1_Stop(void);`



```
/*Cabecera*****
```

```
// Definir las variables ciclo_de_trabajo_actual,
// ciclo_de trabajo_anterior
```

```
unsigned short ciclo_de_trabajo_actual;
unsigned short ciclo_de trabajo_anterior;
```

```
void initMain() {
```

```
    ANSEL = 0;      // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTA = 255;    // Estado inicial del puerto PORTA
    TRISA = 255;    // Todos los pines del puerto PORTA se configuran como entradas
    PORTB = 0;      // Estado inicial del puerto PORTB
    TRISB = 0;      // Todos los pines del puerto PORTB se configuran como salidas
    PORTC = 0;      // Estado inicial del puerto PORTC
    TRISC = 0;      // Todos los pines del puerto PORTC se configuran
```

```
    // como salidas
```

```
    PWM1_Init(5000); // Inicialización del módulo PWM (5KHz)
```

```
}
```

```
void main() {
```

```

initMain();
ciclo_de_trabajo_actual = 16;    // Valor inicial de la variable ciclo_de_trabajo_actua
1
ciclo_de_trabajo_anterior = 0;   // Reiniciar la variable ciclo_de_trabajo_anterior
PWM1_Start();                    // Iniciar el módulo PWM1

while (1) {                      // Bucle infinito
    if (Button(&PORTA, 0,1,1))    // Si se presiona el botón conectado al RA0
        ciclo_de_trabajo_actual++; // incrementar el valor de la variable current_duty
    if (Button(&PORTA, 1,1,1))    // Si se presiona el botón conectado al RA1
        ciclo_de_trabajo_actual--; // decrementar el valor de la variable current_duty

    if (old_duty != ciclo_de_trabajo_actual) { // Si ciclo_de_trabajo_actual
y
        // ciclo_de_trabajo_anterior no son iguales
        PWM1_Set_Duty(ciclo_de_trabajo_actual); // ajustar un nuevo valor a PW
M,
        ciclo_de_trabajo_anterior = ciclo_de_trabajo_actual; // Guardar el nuevo valor
        PORTB = ciclo_de_trabajo_anterior; // y visualizarlo en el puerto
PORTB
    }
    Delay_ms(200); // Tiempo de retardo de 200mS
}
}

```

Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

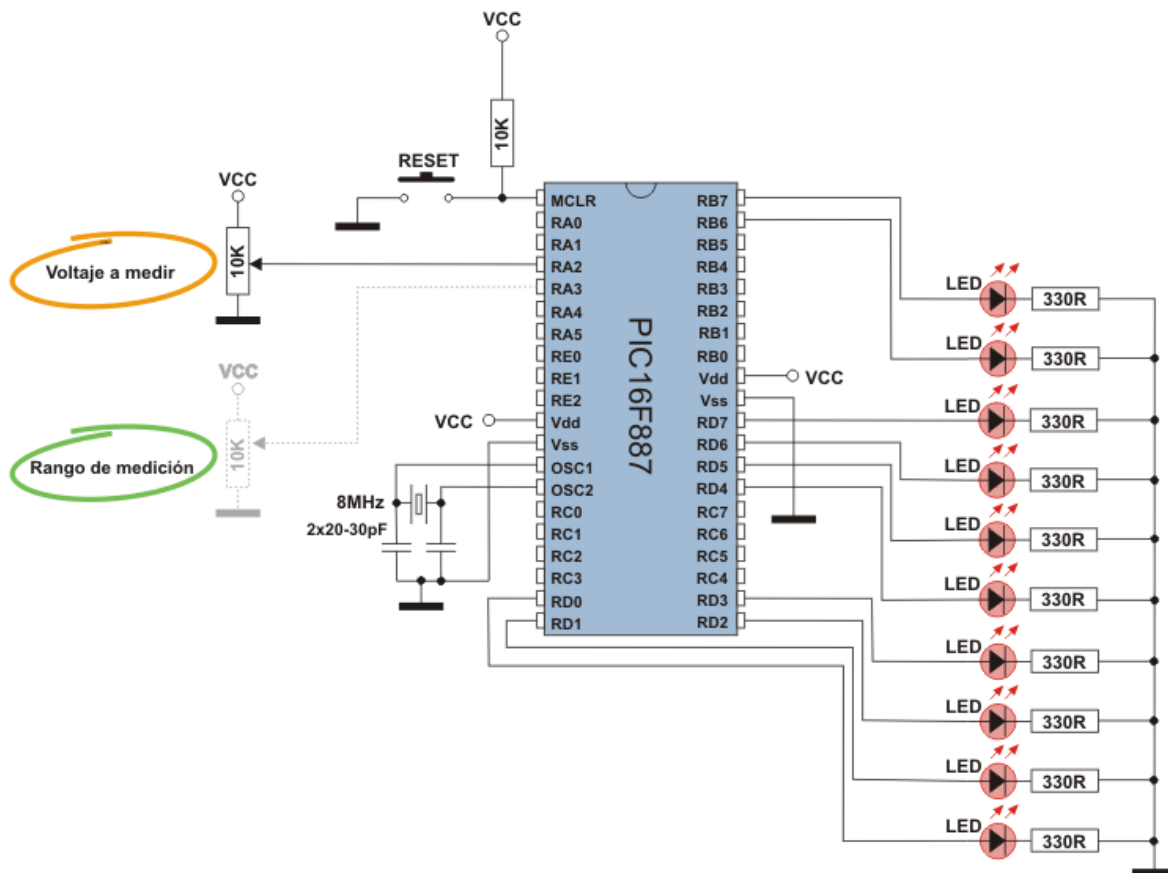
- PWM
- Button

4.9 Example 7

Utilizar el convertidor A/D

El convertidor A/D del microcontrolador PIC16F887 se utiliza en este ejemplo. ¿Hace falta decir que todo es pan comido? Una señal analógica variable se aplica al pin AN2, mientras que el resultado de la conversión de 10 bits se muestra en los puertos POTRB y PORTD (8 bits menos significativos en el puerto PORTD y 2 bits más significativos en el puerto PORTB). La Tierra (GND) se utiliza como voltaje de

referencia bajo Vref-, mientras que el voltaje de referencia alto se aplica al pin AN3. Esto habilita que la escala de medición se estire y encoja. En otras palabras, el convertidor A/D siempre genera un resultado binario de 10 bits, lo que significa que reconoce 1024 niveles de voltaje en total ($2^{10}=1024$). La diferencia entre dos niveles de voltaje no es siempre la misma. Cuánto menor sea la diferencia entre Vref+ y Vref-, tanto menor será la diferencia entre dos de 1024 niveles. Como hemos visto, el convertidor A/D es capaz de detectar pequeños cambios de voltaje.



```
/*Cabecera*****
/

unsigned int temp_res;

void main() {
    ANSEL = 0x0C;           // Pines AN2 y AN3 se configuran como analógicos
    TRISA = 0xFF;           // Todos los pines del puerto PORTA se configuran

    // como entradas

    ANSELH = 0;             // Los demás pines se configuran como digitales
}
```

```

TRISB = 0x3F;           // Pines del puerto PORTB, RB7 y RB6 se configuran

// como salidas
TRISD = 0;              // Todos los pines del PORTD se configuran como salidas
ADCON1.F4 = 1 ;        // Voltaje de referencia es llevado al pin RA3.

do {
    temp_res = ADC_Read(2); // Resultado de la conversión A/D es copiado a temp_res
    PORTD = temp_res;       // 8 bits menos significativos se mueven al puerto PORTD
    PORTB = temp_res >> 2; // 2 bits más significativos se mueven a los bits RB6 y RB7
} while(1);              // Bucle infinito
}

```

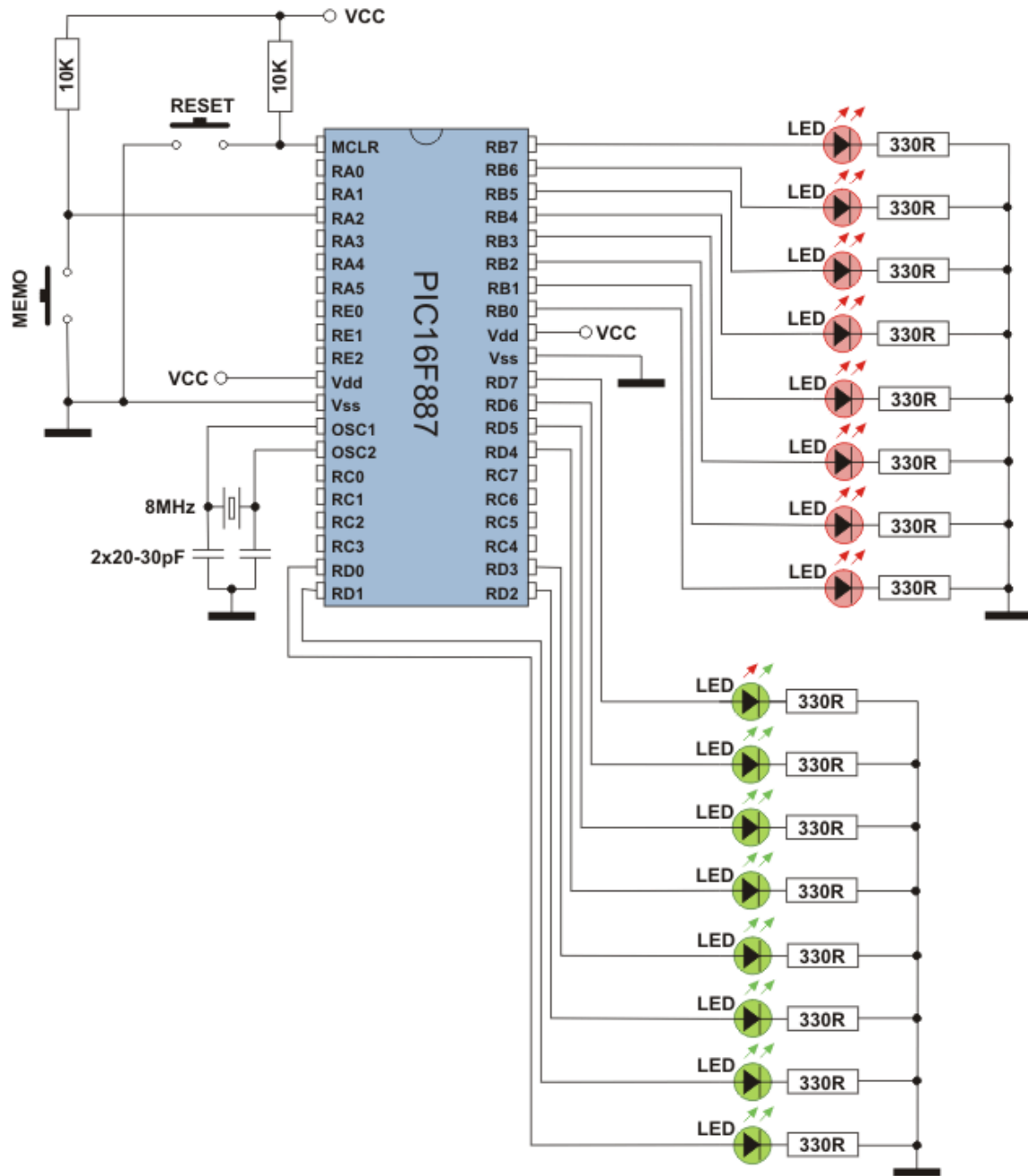
Para que este ejemplo funcione apropiadamente, es necesario marcar la librería ADC en la ventana Library Manager antes de compilar el programa:

- ADC

4.10 Ejemplo 8

Utilizar memoria EEPROM

Este ejemplo muestra cómo escribir y leer la memoria EEPROM incorporada. El programa funciona de la siguiente manera. El bucle principal lee constantemente el contenido de localidad de la memoria EEPROM en la dirección 5 (decimal). Luego el programa entra en el bucle infinito en el que el puerto PORTB se incrementa y se comprueba el estado de entradas del puerto PORTA.2. En el momento de presionar el botón denominado MEMO, un número almacenado en el puerto PORTB será guardado en la memoria EEPROM, directamente leído y visualizado en el puerto PORTD en forma binaria.



```

/*Cabecera*****
/

void main() {{
    ANSEL = 0;           // Todos los pines de E/S se configuran como digitales
    ANSELH = 0;
    PORTB = 0;           // Valor inicial del puerto PORTB
    TRISB = 0;           // Todos los pines del puerto PORTB se configuran

```



```

// como salidas
PORTD = 0;           // Valor inicial del puerto PORTB
TRISD = 0;           // Todos los pines del puerto PORTD se configuran

// como salidas
TRISA = 0xFF;        // Todos los pines del puerto PORTA se configuran

// como entradas
PORTD = EEPROM_Read(5); // Leer la memoria EEPROM en la dirección 5

do {
    PORTB = PORTB++;    // Incrementar el puerto PORTB en 1
    Delay_ms(100);      // Tiempo de retardo de 100mS

    if (PORTA.F2)
        EEPROM_Write(5,PORTB); // Si se pulsa el botón MEMO, guardar el puerto PORTB

    PORTD = EEPROM_Read(5); // Leer el dato escrito

    do {
        while (PORTA.F2);    // Quedarse en este bucle hasta que el botón esté pulsado
    }
}
while(1);              // Bucle infinito
}

```

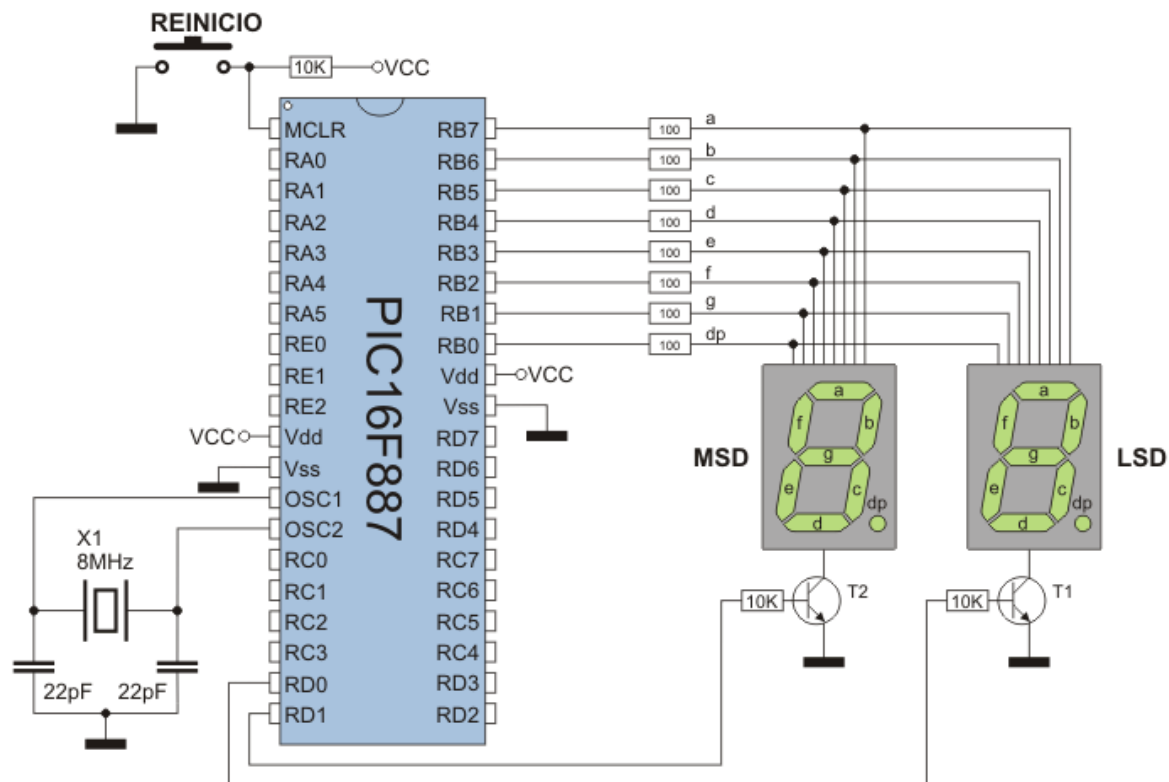
Para comprobar el funcionamiento de este circuito, basta con pulsar el botón MEMO y apagar el dispositivo. Después de reiniciar el dispositivo, el programa visualizará el valor guardado en el puerto PORTD. Acuérdesse de que en el momento de escribir, el valor fue visualizado en el puerto PORTB. Para que este ejemplo funcione apropiadamente, es necesario marcar la librería EEPROM en la ventana Library Manager antes de compilar el programa:

- EEPROM

4.11 Ejemplo 9

Contador de dos dígitos LED, multiplexión

En este ejemplo el microcontrolador funciona como un contador de dos dígitos. La variable *i* se incrementa (suficiente lentamente para ser visible) y su valor se visualiza en un visualizador de dos dígitos LED (99-0). El punto es habilitar una conversión de un número binario en un decimal y partirlo en dos dígitos (en decenas y unidades). Como los segmentos del visualizador LED se conectan en paralelo, es necesario asegurar que alternen rápidamente para tener una impresión de que emiten la luz simultáneamente (multiplexión por división en tiempo). En este ejemplo, el temporizador TMR0 está encargado de la multiplexión por división en tiempo, mientras que la función mask convierte un número binario a formato decimal.



```
/*Cabecera*****
```

```
unsigned short mask(unsigned short num);  
unsigned short digit_no, digit10, digit1, digit, i;
```

```

void interrupt() {
    if (digit_no == 0) {
        PORTA = 0;          // Apagar ambos visualizadores
        PORTD = digit1;     // Colocar máscara para visualizar unidades en el

        // puerto PORTD
        PORTA = 1;          // Encender el visualizador para las unidades (LSD)
        digit_no = 1;
    }else{
        PORTA = 0;          // Apagar ambos visualizadores
        PORTD = digit10;    // Colocar máscara para visualizar decenas en el

        // puerto PORTD
        PORTA = 2;          // Encender el visualizador para las decenas (MSD)
        digit_no = 0;
    }

    TMR0 = 0;               // Reiniciar el contador TMR0
    INTCON = 0x20;          // Bit T0IF=0, T0IE=1
}

void main() {
    OPTION_REG = 0x80;     // Ajustar el temporizador TMR0
    TMR0 = 0;
    INTCON = 0xA0;         // Deshabilitar las interrupciones PEIE, INTE, RBIE, T0IE
    PORTA = 0;             // Apagar ambos visualizadores
    TRISA = 0;             // Todos los pines del puerto PORTA se configuran

    // como salidas
    PORTD = 0;             // Apagar todos los segmentos del visualizador
    TRISD = 0;             // Todos los pines del puerto PORTD se configuran

    // como salidas
    do {
        for (i = 0; i<=99; i++) { // Contar de 0 a 99
            digit = i % 10u;
            digit1 = mask(digit); // Preparar la máscara para visualizar unidades

```

```

        digit = (char)(i / 10u) % 10u;
        digit10 = mask(digit); // Preparar la máscara para visualizar decenas
        Delay_ms(1000);
    }
} while (1);           // Bucle infinito
}

```

mask.c

```

/*Cabecera*****
unsigned short mask(unsigned short num) {
    switch (num) {
        case 0 : return 0x3F;
        case 1 : return 0x06;
        case 2 : return 0x5B;
        case 3 : return 0x4F;
        case 4 : return 0x66;
        case 5 : return 0x6D;
        case 6 : return 0x7D;
        case 7 : return 0x07;
        case 8 : return 0x7F;
        case 9 : return 0x6F;
    }
}

```

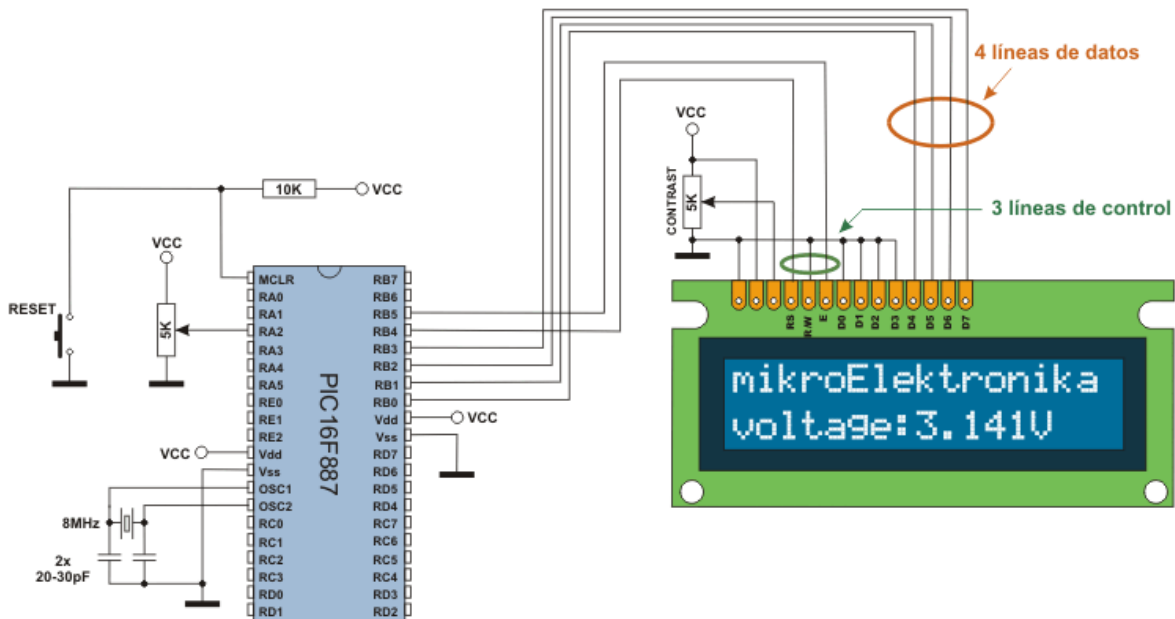
Para que este ejemplo funcione apropiadamente, es necesario incluir el archivo mask.c en el proyecto (aparte del archivo example9.c) en la ventana Project Manager antes de compilar el programa: *Example9.mcppi - Sources - Add File To Project*

- mask.c
- example9.c

4.12 Ejemplo 10

Utilizar el visualizador LCD

Este ejemplo muestra cómo utilizar un visualizador LCD alfanumérico. Las librerías de funciones simplifican este programa, lo que significa que al final el esfuerzo para crear el software vale la pena. Un mensaje escrito en dos líneas aparece en el visualizador: **mikroElektronika LCD example** Dos segundos más tarde, el mensaje en la segunda línea cambia, y se visualiza el voltaje presente en la entrada del convertidor A/D (el pin RA2). Por ejemplo: **mikroElektronika voltage:3.141V** En un dispositivo real se puede visualizar temperatura actual o algún otro valor medido en vez de voltaje.



Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- ADC
- LCD

```
/*Cabecera*****
```

```
// Conexiones del módulo LCD
```

```
sbit LCD_RS at RB4_bit;
```

```
sbit LCD_EN at RB5_bit;
```

```
sbit LCD_D4 at RB0_bit;
```

```

sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Final de las conexiones del módulo LCD

// Declarar variables
unsigned char ch;
unsigned int adc_rd;
char *text;
long tlong;

void main() {
    INTCON = 0;           // Todas las interrupciones deshabilitadas
    ANSEL = 0x04;         // Pin RA2 se configura como una entrada analógica
    TRISA = 0x04;
    ANSELH = 0;           // Los demás pines se configuran como digitales

    Lcd_Init();           // Inicialización del visualizador LCD
    Lcd_Cmd(_LCD_CURSOR_OFF); // Comando LCD (apagar el cursor)
    Lcd_Cmd(_LCD_CLEAR);  // Comando LCD (borrar el LCD)

    text = "mikroElektronika"; // Definir el primer mensaje
    Lcd_Out(1,1,text);        // Escribir el primer mensaje en la primera línea

    text = "LCD example";    // Definir el segundo mensaje
    Lcd_Out(2,1,text);       // Definir el primer mensaje

    ADCON1 = 0x82;          // Voltaje de referencia para la conversión A/D es VCC
    TRISA = 0xFF;           // Todos los pines del puerto PORTA se configuran como e
ntradas
    Delay_ms(2000);

```

```

text = "voltage:";          // Definir el tercer mensaje

while (1) {
    adc_rd = ADC_Read(2);    // Conversión A/D. Pin RA2 es una entrada.
    Lcd_Out(2,1,text);       // Escribir el resultado en la segunda línea
    tlong = (long)adc_rd * 5000; // Convertir el resultado en milivoltios
    tlong = tlong / 1023;     // 0..1023 -> 0-5000mV
    ch = tlong / 1000;        // Extraer voltios (miles de milivoltios)

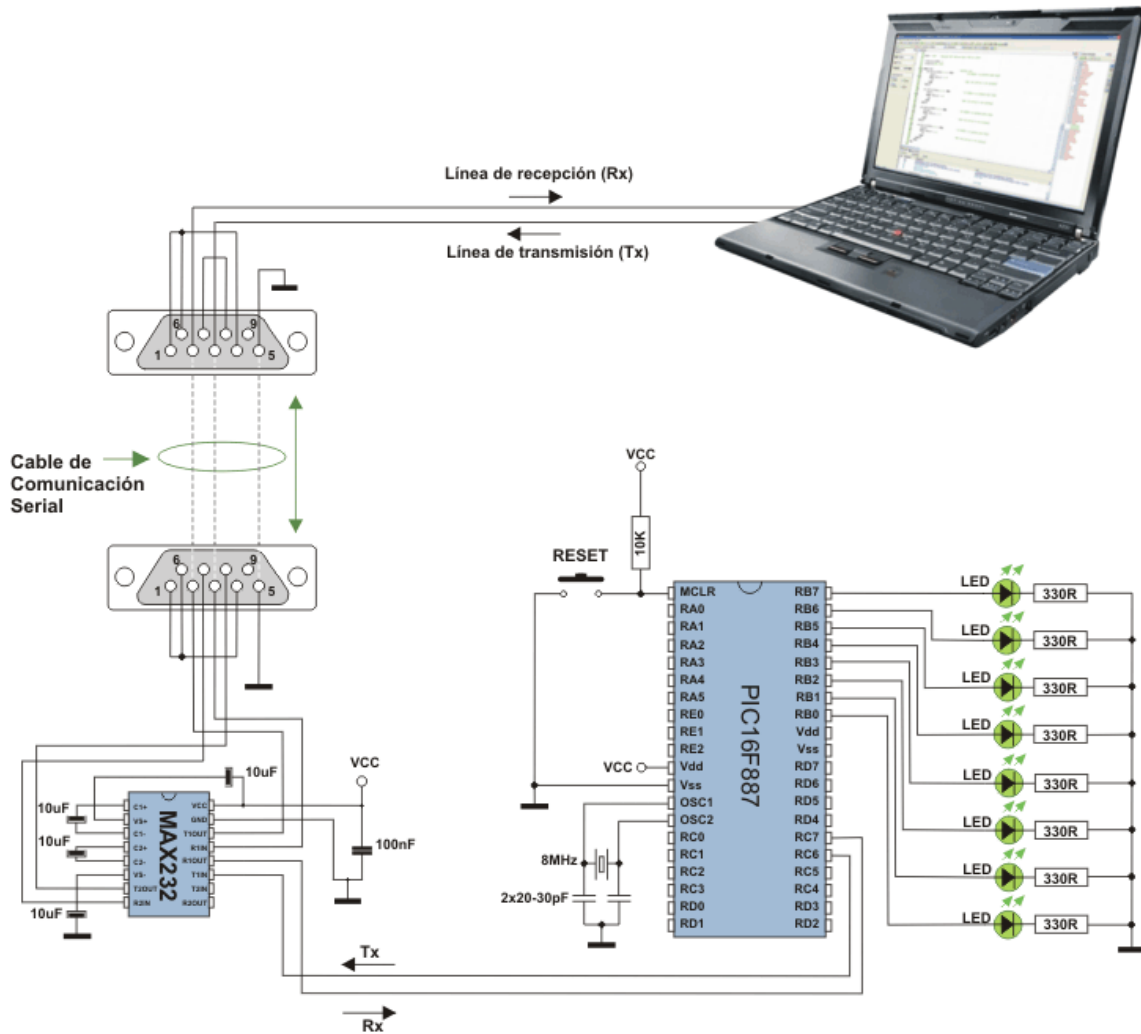
    // del resultado
    Lcd_Chr(2,9,48+ch);      // Escribir resultado en formato ASCII
    Lcd_Chr_CP('.');
    ch = (tlong / 100) % 10;  // Extraer centenas de milivoltios
    Lcd_Chr_CP(48+ch);       // Escribir resultado en formato ASCII
    ch = (tlong / 10) % 10;   // Extraer decenas de milivoltios
    Lcd_Chr_CP(48+ch);       // Escribir resultado en formato ASCII
    ch = tlong % 10;          // Extraer unidades de milivoltios
    Lcd_Chr_CP(48+ch);       // Escribir resultado en formato ASCII
    Lcd_Chr_CP('V');
    Delay_ms(1);
}
}

```

4.13 Ejemplo 11

Comunicación serial RS-232

Este ejemplo muestra cómo utilizar el módulo EUSART del microcontrolador. La conexión a una PC se habilita por medio del estándar de comunicación RS-232. El programa funciona de la siguiente manera. Cada byte recibido por medio de la comunicación serial se visualiza al utilizar los LEDs conectados al puerto PORTB y después se devuelve automáticamente al transmisor. Si ocurre un error en recepción, se lo indicará al encender el diodo LED. La manera más fácil es comprobar el funcionamiento del dispositivo en la práctica al utilizar un programa estándar de Windows denominado *Hyper Terminal*.



```
/*Cabecera*****
```

```
unsigned short i;
```

```
void main() {
```

```
    UART1_Init(19200); // Inicializar el módulo USART
```

```
    // (8 bits, tasa de baudios 19200, no hay bit
```

```
    // de paridad...)
```

```
    while (1) {
```

```
        if (UART1_Data_Ready()) { // si se ha recibido un dato
```

```
            i = UART1_Read(); // leerlo
```

```
            UART1_Write(i); // enviarlo atrás
```

```
        }
```



```
}  
}
```

Para que este ejemplo funcione apropiadamente, es necesario marcar la librería UART en la ventana Library Manager antes de compilar el programa:

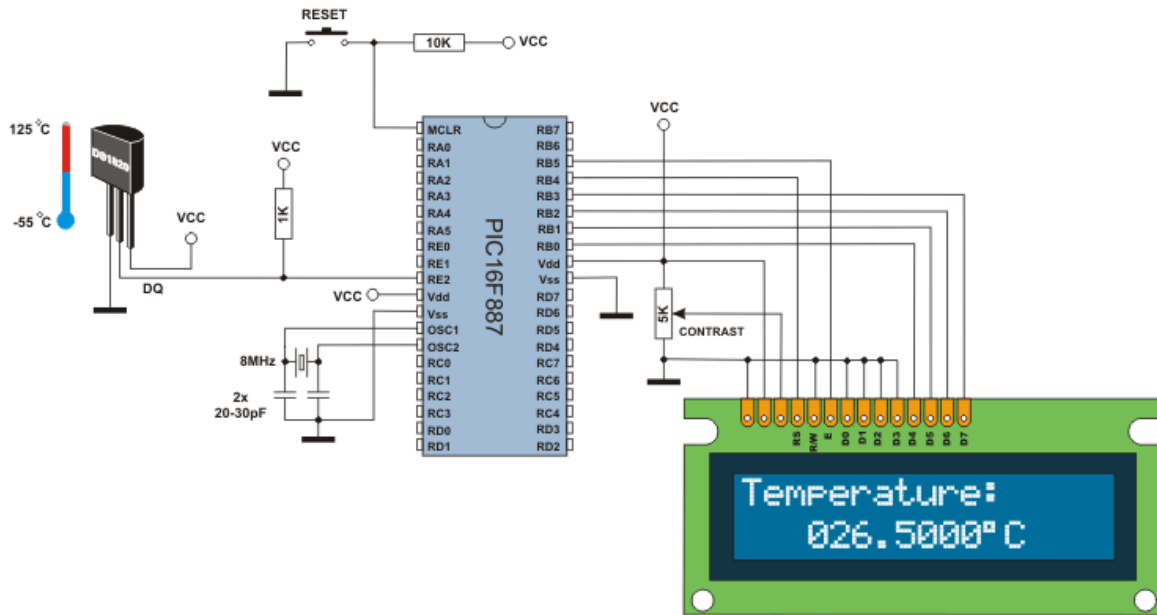
- UART

4.14 Ejemplo 12

Medición de temperatura por medio del sensor DS1820. Uso del protocolo '1-wire'...

La medición de temperatura es una de las tareas más frecuentes realizadas por el microcontrolador. En este ejemplo, se utiliza un sensor DS1820 para medir. Es capaz de medir en el rango de 55 °C a 125 °C con exactitud de 0.5 °C. Para transmitir los datos al microcontrolador se utiliza un tipo especial de la comunicación serial denominado 1-wire. Debido al hecho de que estos sensores son simples de utilizar y de grandes capacidades, los comandos utilizados para hacerlos funcionar y controlarlos tienen la forma de funciones almacenadas en la librería One_Wire. En total son las siguientes tres funciones:

- **Ow_Reset** se utiliza para reiniciar el sensor;
- **Ow_Read** se utiliza para recibir los datos del sensor; y
- **Ow_Write** se utiliza para enviar los comandos al sensor



Este ejemplo muestra la ventaja de utilizar librerías con las funciones listas para ser utilizadas. Concretamente, no tiene que examinar la documentación proporcionada por el fabricante para utilizar el sensor. Basta con copiar alguna de estas funciones en el programa. Si le interesa saber cómo se declaran, basta con pulsar sobre alguna de ellas y seleccione la opción Help.

```
/*Cabecera*****
```

```
// Conexiones del módulo LCD
```

```
sbit LCD_RS at RB4_bit;
```

```
sbit LCD_EN at RB5_bit;
```

```
sbit LCD_D4 at RB0_bit;
```

```
sbit LCD_D5 at RB1_bit;
```

```
sbit LCD_D6 at RB2_bit;
```

```
sbit LCD_D7 at RB3_bit;
```

```
sbit LCD_RS_Direction at TRISB4_bit;
```

```
sbit LCD_EN_Direction at TRISB5_bit;
```

```
sbit LCD_D4_Direction at TRISB0_bit;
```

```
sbit LCD_D5_Direction at TRISB1_bit;
```

```
sbit LCD_D6_Direction at TRISB2_bit;
```

```
sbit LCD_D7_Direction at TRISB3_bit;
```

```
// Final de conexiones del módulo LCD
```

```
const unsigned short TEMP_RESOLUTION = 9;
```

```

char *text = "000.0000";
unsigned temp;

void Display_Temperature(unsigned int temp2write) {
    const unsigned short RES_SHIFT = TEMP_RESOLUTION - 8;
    char temp_whole;
    unsigned int temp_fraction;

    // comprobar si la temperatura es negativa
    if (temp2write & 0x8000) {
        text[0] = '-';
        temp2write = ~temp2write + 1;
    }

    // extraer temp_whole
    temp_whole = temp2write >> RES_SHIFT ;

    // convertir temp_whole en caracteres
    if (temp_whole/100)
        text[0] = temp_whole/100 + 48;
    else
        text[0] = '0';

    text[1] = (temp_whole/10)%10 + 48;    // Extraer dígito de decenas
    text[2] = temp_whole%10 + 48;        // Extraer dígito de unidades

    // extraer temp_fraction y convertirlo en unsigned int
    temp_fraction = temp2write << (4-RES_SHIFT);
    temp_fraction &= 0x000F;
    temp_fraction *= 625;

    // convertir temp_fraction en caracteres
    text[4] = temp_fraction/1000 + 48;    // Extraer dígito de miles
    text[5] = (temp_fraction/100)%10 + 48; // Extraer dígito de centenas
    text[6] = (temp_fraction/10)%10 + 48; // Extraer dígito de decenas
    text[7] = temp_fraction%10 + 48;      // Extraer dígito de unidades

    // Visualizar temperatura en el LCD

```

```

    Lcd_Out(2, 5, text);
}

void main() {
    ANSEL = 0;           // Configurar los pines AN como digitales
    ANSELH = 0;
    C1ON_bit = 0;        // Deshabilitar los comparadores
    C2ON_bit = 0;

    Lcd_Init();          // Inicializar el LCD
    Lcd_Cmd(_LCD_CLEAR); // Borrar el LCD
    Lcd_Cmd(_LCD_CURSOR_OFF); // Apagar el cursor
    Lcd_Out(1, 1, " Temperatura: ");

    // Visualizar el carácter de grado, 'C' para centígrados
    Lcd_Chr(2,13,223); // distintos visualizadores LCD tienen diferentes códigos

    // de caracteres para el grado
    // si ve la letra griega Alfa, intente introducir 178 en vez de 223
    Lcd_Chr(2,14,'C');

    //--- bucle principal
    do {
        //--- realizar lectura de temperatura
        Ow_Reset(&PORTE, 2); // Señal de reinicio de Onewire
        Ow_Write(&PORTE, 2, 0xCC); // Ejecutar el comando SKIP_ROM
        Ow_Write(&PORTE, 2, 0x44); // Ejecutar el comando CONVERT_T
        Delay_us(120);
        Ow_Reset(&PORTE, 2);
        Ow_Write(&PORTE, 2, 0xCC); // Ejecutar el comando SKIP_ROM
        Ow_Write(&PORTE, 2, 0xBE); // Ejecutar el comando READ_SCRATCHPAD
        temp = Ow_Read(&PORTE, 2);
        temp = (Ow_Read(&PORTE, 2) << 8) + temp;

        //--- Formatear y visualizar el resultado en el LCD
        Display_Temperature(temp);
        Delay_ms(500);
    } while (1);
}

```

}

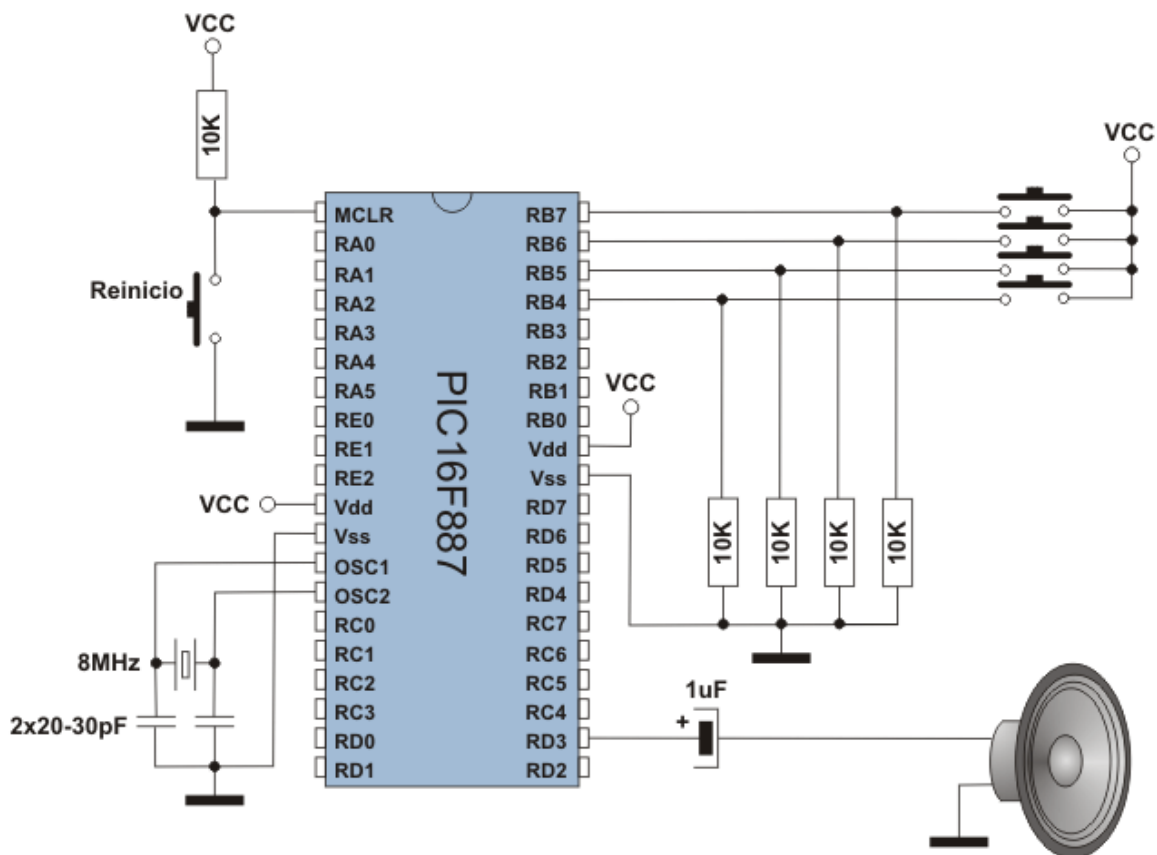
Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- One_Wire
- LCD

4.15 Ejemplo 13

Generación de sonido, librería de sonido...

Las señales de audio se utilizan con frecuencia cuando se necesita llamar la atención de usuario, confirmar que alguno de los botones se ha pulsado, avisar que se ha llegado hasta los valores mínimos o máximos etc. Pueden ser una simple señal de pitido así como melodías de una duración más larga o más corta. En este ejemplo se muestra la generación de sonido por medio de funciones que pertenecen a la librería Sound.



Además de estas funciones, la función Button que pertenece a la misma librería se utiliza para probar los botones de presión.

```
/*Cabecera*****  
  
void Tone1() {  
    Sound_Play(659, 250); // Frecuencia = 659Hz, duración = 250ms  
}  
  
void Tone2() {  
    Sound_Play(698, 250); // Frecuencia = 698Hz, duración = 250ms  
}  
  
void Tone3() {  
    Sound_Play(784, 250); // Frecuencia = 784Hz, duración = 250ms  
}  
  
void Melody1() { // Componer una melodía divertida 1  
    Tone1(); Tone2(); Tone3(); Tone3();  
    Tone1(); Tone2(); Tone3(); Tone3();  
    Tone1(); Tone2(); Tone3();  
    Tone1(); Tone2(); Tone3(); Tone3();  
    Tone1(); Tone2(); Tone3();  
    Tone3(); Tone3(); Tone2(); Tone2(); Tone1();  
}  
  
void ToneA() { // Tono A  
    Sound_Play( 880, 50);  
}  
  
void ToneC() { // Tono C  
    Sound_Play(1046, 50);  
}  
  
void ToneE() { // Tono E  
    Sound_Play(1318, 50);  
}
```

```

void Melody2() { // Componer una melodía divertida 2
    unsigned short i;

    for (i = 9; i > 0; i--) {
        ToneA(); ToneC(); ToneE();
    }
}

void main() {
    ANSEL = 0;                // Todos los pines de E/S son digitales
    ANSELH = 0;
    TRISB = 0xF0;            // Pines RB7-RB4 se configuran como entradas

    // RB3 se configura como salida
    Sound_Init(&PORTB, 3);
    Sound_Play(1000, 500);

    while (1) {
        if (Button(&PORTB,7,1,1)) // RB7 genera Tono1
            Tone1();
        while (PORTB & 0x80) ;    // Esperar que se suelte el botón
        if (Button(&PORTB,6,1,1)) // RB6 genera Tono2
            Tone2();
        while (PORTB & 0x40) ;    // Esperar que se suelte el botón
        if (Button(&PORTB,5,1,1)) // RB5 genera melodía 2
            Melody2();
        while (PORTB & 0x20) ;    // Esperar que se suelte el botón
        if (Button(&PORTB,4,1,1)) // RB4 genera melodía 1
            Melody1();
        while (PORTB & 0x10) ;    // Esperar que se suelte el botón
    }
}

```

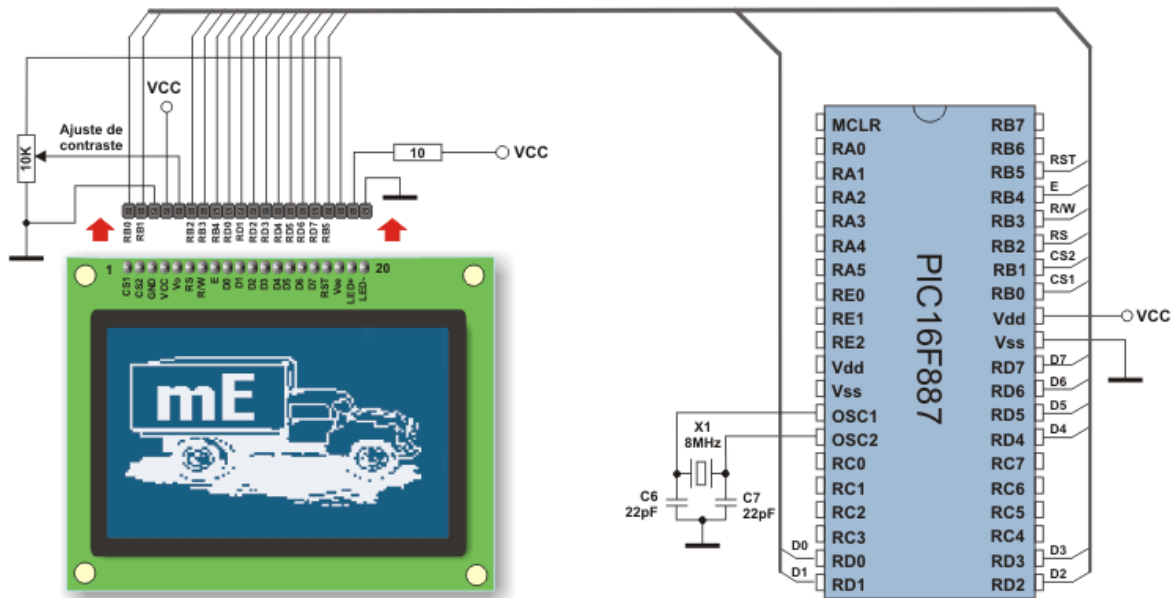
Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana Library Manager antes de compilar el programa:

- Button
- Sound

4.16 Ejemplo 14

Utilizar el visualizador LCD gráfico

Un LCD gráfico (GLCD) proporciona un método avanzado para visualizar mensajes. Mientras que un LCD de caracteres puede visualizar sólo caracteres alfanuméricos, el LCD gráfico puede visualizar los mensajes en forma de dibujos y mapas de bits. El LCD gráfico utilizado con más frecuencia tiene una resolución de pantalla de 128x64 píxeles. El contraste de un GLCD se puede ajustar por medio del potenciómetro P1.



En este ejemplo el GLCD visualiza una secuencia de imágenes, de las que un mapa de bits representa un camión almacenado en otro archivo denominado truck_bmp.c. Para habilitar que este programa funcione apropiadamente, usted debe añadir este archivo como archivo fuente a su proyecto. Las directivas del preprocesador incluidas en este ejemplo le permiten elegir si quiere visualizar toda la secuencia de imágenes o sólo una secuencia corta. Por defecto se visualizará toda la secuencia de imágenes. Sin embargo, al añadir un comentario delante de la directiva `#define COMPLETE_EXAMPLE` se deshabilita visualizar algunas imágenes de la secuencia. Si se comenta (o si se borra) esta directiva, las sentencias entre las directivas `#ifdef COMPLETE_EXAMPLE` y `#endif` no serán compiladas, así que no se ejecutarán.

```
/*Cabecera*****  
  
//Declaraciones-----  
const code char truck_bmp[1024]; // Declarar la constante definida en truck_bmp.c
```



```

// para utilizarla en este archivo
//-----final-de-declaraciones

// Conexiones del módulo Glcd
char GLCD_DataPort at PORTD;
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;
sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// Final de conexiones del módulo Glcd

void delay2S(){ // Función de tiempo de retardo de 2 segundos
    Delay_ms(2000);
}

void main() {
    unsigned short ii;
    char *someText;

    /* Esta directiva define un macro denominado COMPLETE_EXAMPLE. Más tarde en el
    programa, la directiva ifdef prueba si este macro está definido. Si se borra esta
    línea o si se transforma en un comentario, las secciones del código entre las
    directivas ifdef y endif no serán compiladas. */

    #define COMPLETE_EXAMPLE // Poner esta línea como un comentario si quiere
    // visualizar la versión corta de la secuencia

    ANSEL = 0; // Configurar los pines AN como digitales
    ANSELH = 0;
    C1ON_bit = 0; // Deshabilitar comparadores

```

```

C2ON_bit = 0;
Glcd_Init();      // Inicializar el GLCD
Glcd_Fill(0x00); // Borrar el GLCD

while(1) {          // bucle infinito, la secuencia se repite

/* Dibujar la primera imagen */
#ifdef COMPLETE_EXAMPLE
Glcd_Image(truck_bmp);      // Dibujar la imagen
delay2S(); delay2S();
#endif
Glcd_Fill(0x00);           // Borrar el GLCD

/* Dibujar la segunda imagen */
Glcd_Box(62,40,124,56,1);   // Dibujar la caja
Glcd_Rectangle(5,5,84,35,1); // Dibujar el rectángulo
Glcd_Line(0, 0, 127, 63, 1); // Dibujar la línea

delay2S();

for(ii = 5; ii < 60; ii+=5 ){ // Dibujar líneas horizontales y verticales
    Delay_ms(250);
    Glcd_V_Line(2, 54, ii, 1);
    Glcd_H_Line(2, 120, ii, 1);
}

delay2S();
Glcd_Fill(0x00); // Borrar el GLCD

/* Dibujar la tercera imagen */

#ifdef COMPLETE_EXAMPLE
    Glcd_Set_Font(Character8x7, 8, 7, 32); // Seleccionar la fuente, ver
    // __Lib_GLCDFonts.c en la carpeta Uses
#endif

Glcd_Write_Text("mikroE", 1, 7, 2);      // Escribir la cadena
for (ii = 1; ii <= 10; ii++)              // Dibujar los círculos

```

```

    Glcd_Circle(63,32, 3*ii, 1);
    delay2S();

    Glcd_Box(12,20, 70,57, 2);           // Dibujar la caja
    delay2S();

    /* Dibujar la cuarta imagen */
    #ifdef COMPLETE_EXAMPLE
    Glcd_Fill(0xFF); // Llenar el GLCD
    Glcd_Set_Font(Character8x7, 8, 7, 32); // Cambiar de la fuente
    someText = "8x7 Font";
    Glcd_Write_Text(someText, 5, 0, 2);    // Escribir la cadena
    delay2S();
    Glcd_Set_Font(System3x5, 3, 5, 32);    // Cambiar de la fuente
    someText = "3X5 CAPITALS ONLY";
    Glcd_Write_Text(someText, 60, 2, 2);    // Escribir la cadena
    delay2S();
    Glcd_Set_Font(font5x7, 5, 7, 32);      // Cambiar de la fuente
    someText = "5x7 Font";
    Glcd_Write_Text(someText, 5, 4, 2);    // Escribir la cadena
    delay2S();
    Glcd_Set_Font(FontSystem5x7_v2, 5, 7, 32); // Cambiar de la fuente
    someText = "5x7 Font (v2)";
    Glcd_Write_Text(someText, 5, 6, 2);    // Escribir la cadena
    delay2S();
    #endif
    }
}

```

truck_bmp.c:

```

/*Cabecera******/

unsigned char const truck_bmp[1024] = {
    0, 0, 0, 0, 0,248, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12,
    12, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9, 9,
    9, 9, 9, 9, 9, 9, 9, 9, 9, 9,137,137,137,137,137,137,
    137,137,137,137,137,137,137, 9, 9, 9, 9, 9, 9, 9, 9,

```

9, 9, 13,253, 13,195, 6,252, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
240,240,240,240,240,224,224,240,240,240,240,224,192,192,224,
240,240,240,240,240,224,192, 0, 0, 0,255,255,255,255,195,
195,195,195,195,195,195, 3, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,255,240, 79,224,255, 96, 96, 96, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 64, 64, 64, 64,128, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
255,255,255,255,255, 0, 0, 0, 0,255,255,255,255,255, 0, 0,
0, 0,255,255,255,255,255, 0, 0, 0,255,255,255,255,129,
129,129,129,129,129,129,128, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,255, 1,248, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8, 16,224, 24, 36,196, 70,130,130,133,217,102,112,
160,192, 96, 96, 32, 32,160,160,224,224,192, 64, 64,128,128,192,
64,128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 63, 96, 96, 96,224, 96, 96, 96, 96, 96, 96,
99, 99, 99, 99, 99, 96, 96, 96, 96, 99, 99, 99, 99, 99, 96, 96,
96, 96, 99, 99, 99, 99, 99, 96, 96, 96, 99, 99, 99, 99, 99,
99, 99, 99, 99, 99, 99, 99, 96, 96, 96, 96, 96, 96, 64, 64,
64,224,224,255,246, 1, 14, 6, 6, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2,130, 67,114, 62, 35, 16, 16, 0, 7, 3, 3, 2,
4, 4, 4, 4, 4, 4, 28, 16, 16, 16, 17, 17, 9, 9, 41,
112, 32, 67, 5,240,126,174,128, 56, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1,127,127,127,127,255,255,247,251,123,191, 95, 93,125,189,
189, 63, 93, 89,177,115,243,229,207, 27, 63,119,255,207,191,255,
255,255,255,255,255,255,127,127,127,127,127,127,127,255,
255,255,127,127,125,120,120,120,120,120,248,120,120,120,120,
120,248,248,232,143, 0, 0, 0, 0, 0, 0, 0, 0,128,240,248,
120,188,220, 92,252, 28, 28, 60, 92, 92, 60,120,248,248, 96,192,
143,168,216,136, 49, 68, 72, 50,160, 96, 0, 0, 0, 0, 0,
0, 0, 0,128,192,248,248,248,248,252,254,254,254,254,254,
254,254,254,254,254,255,255,255,255,255,246,239,208,246,174,173,

```

169,128,209,208,224,247,249,255,255,252,220,240,127,255,223,255,
255,255,255,255,255,254,254,255,255,255,255,255,255,254,255,
255,255,255,255,255,255,254,254,254,254,254,254,254,254,254,
254,254,254,254,255,255,255,255,255,255,254,255,190,255,255,253,
240,239,221,223,254,168,136,170,196,208,228,230,248,127,126,156,
223,226,242,242,242,242,242,177, 32,  0,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  1,  1,  1,  1,  3,  3,  3,  7,  7,  7,  7,  7, 15,
15, 15,  7, 15, 15, 15,  7,  7, 15, 14, 15, 13, 15, 47, 43, 43,
43, 43, 43, 47,111,239,255,253,253,255,254,255,255,255,255,
191,191,239,239,239,191,255,191,255,255,255,255,255,255,255,
255,255,255,255,255,255,255,255,255,255,255,255,255,255,
255,255,255,255,127,127,127,127,255,255,191,191,191,191,255,254,
255,253,255,255,255,251,255,255,255,127,125, 63, 31, 31, 31, 31,
31, 31, 63, 15, 15,  7,  7,  3,  3,  3,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,
  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
  1,  1,  1,  1,  3,  3,  3, 11, 11, 11, 11,  7,  3, 14,  6,  6,
  6,  2, 18, 19, 19,  3, 23, 21, 21, 17,  1, 19, 19,  3,  6,  6,
14, 15, 15,  7, 15, 15, 15, 11,  2,  0,  0,  0,  0,  0,  0,  0,
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
};

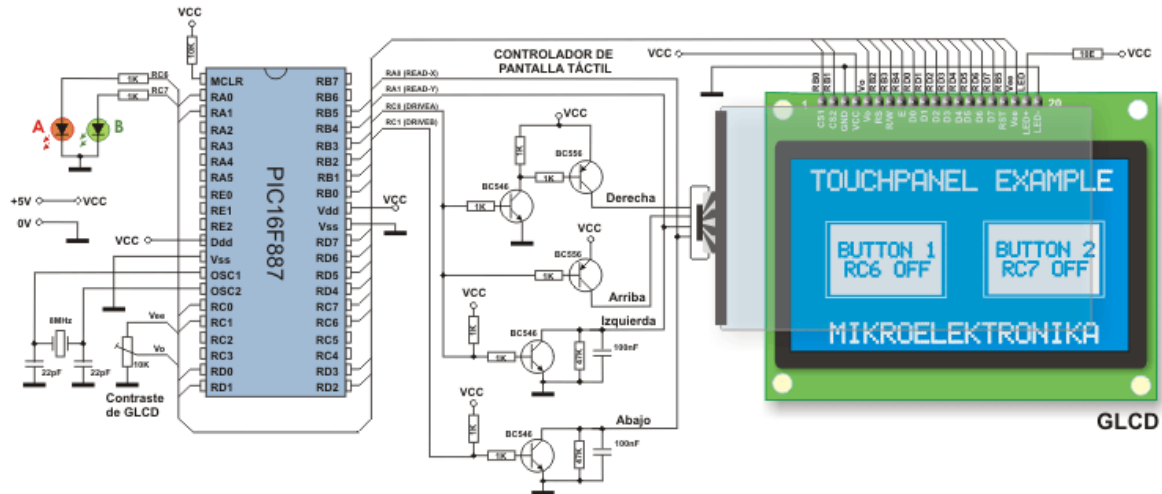
```

Para que este ejemplo funcione apropiadamente, es necesario marcar la librería GLCD en la ventana *Library Manager* antes de compilar el programa. Asimismo, es necesario incluir el documento *truck_bmp.c* en el proyecto.

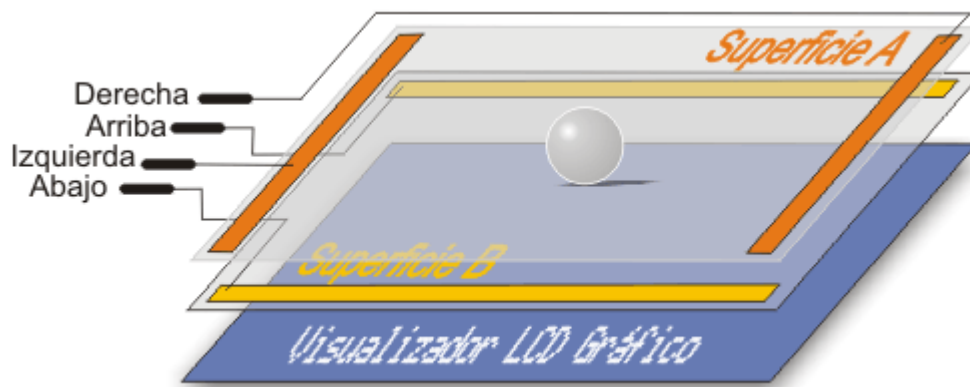
4.17 Ejemplo 15

Utilizar el panel táctil...

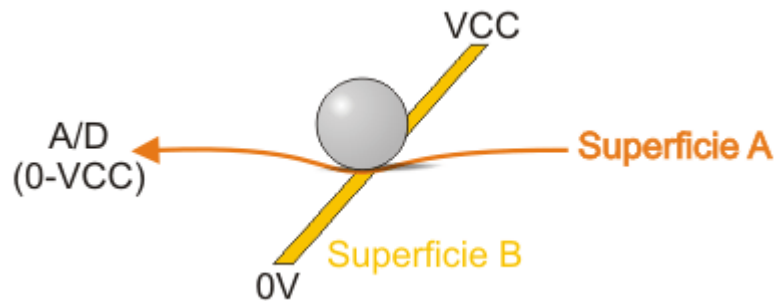
Un panel táctil es un panel fino, autoadhesivo y transparente, colocado sobre la pantalla de un LCD gráfico. Es muy sensible a la presión así que un suave toque provoca algunos cambios en la señal de salida. Hay diferentes tipos de paneles táctiles. El más sencillo es el panel táctil resistivo del que se hablará aquí.



Un panel táctil está compuesto por dos láminas rígidas, formando una estructura de 'sándwich' que tiene capas resistivas en sus caras internas. La resistencia de estas capas no excede normalmente de 1Kohm. Los lados opuestos de las láminas disponen de los contactos para acceder a un cable plano.



El procedimiento para determinar las coordenadas de la posición del panel que ha sido presionado se puede dividir en dos pasos. El primero es determinación de la coordenada X, y el segundo es de determinar la coordenada Y de la posición. Para determinar la coordenada X, es necesario conectar el contacto izquierdo en la superficie A a la masa (tierra) y el contacto derecho a la fuente de alimentación. Esto permite obtener un divisor de voltaje al presionar el panel táctil. El valor de voltaje obtenido en el divisor se puede leer en el contacto inferior de la superficie B. El voltaje variará en el rango de 0V al voltaje suministrado por la fuente de alimentación y depende de la coordenada X. Si el punto está próximo al contacto izquierdo de la superficie A, el voltaje estará próximo a 0V.



Para la determinación de la coordenada Y, es necesario conectar el contacto inferior de la superficie B a masa (tierra), mientras que el contacto superior se conectará a la fuente de alimentación. En este caso, el voltaje se puede leer en el contacto izquierdo de la superficie A. Para conectar un panel táctil al microcontrolador es necesario crear un circuito para el control del panel táctil. Por medio de este circuito, el microcontrolador conecta los contactos adecuados del panel táctil a masa y a la voltaje de alimentación (como describimos anteriormente) para determinar las coordenadas X y Y. El contacto inferior de la superficie B y el contacto izquierdo de la superficie A están conectados al convertidor A/D del microcontrolador. Las coordenadas X e Y se determinan midiendo el voltaje en los respectivos contactos. El software consiste en mostrar un menú en una pantalla LCD gráfica, conmutar de encendido a apagado del panel táctil (control del panel táctil) y leer los valores del convertidor A/D que representan realmente las coordenadas X e Y de la posición. Una vez determinadas las coordenadas, es posible decidir qué es lo que deseamos que haga el microcontrolador. En este ejemplo se explica cómo conmutar entre encendido y apagado dos pines digitales del microcontrolador, conectados a los LEDs A y B. En este ejemplo se utilizan las funciones que pertenecen a las librerías Glcd y ADC. *Teniendo en cuenta que la superficie del panel táctil es ligeramente mayor que la del LCD gráfico, en caso de requerir una mayor precisión en la determinación de las coordenadas, es necesario incluir el software de calibración del panel táctil.*

```
/* Cabecera *****/

// Conexiones del módulo Glcd
char GLCD_DataPort at PORTD;
sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;
```

```

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction at TRISB2_bit;
sbit GLCD_RW_Direction at TRISB3_bit;
sbit GLCD_EN_Direction at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// Final de conexiones del módulo Glcd

// Declaración de la cadena a visualizar en el GLCD
char msg1[] = "TOUCHPANEL EXAMPLE";
char msg2[] = "MIKROELEKTRONIKA";
char msg3[] = "BUTTON1";
char msg4[] = "BUTTON2";
char msg5[] = "RC6 OFF";
char msg6[] = "RC7 OFF";
char msg7[] = "RC6 ON ";
char msg8[] = "RC7 ON ";

// Declaración de variables globales
long x_coord, y_coord, x_coord128, y_coord64; // almacenar la posición de las
// coordenadas x e y

// Leer la coordenada X
unsigned int GetX() {
    //reading X
    PORTC.F0 = 1;      // DRIVEA = 1 (electrodo izquierdo (LEFT) conectado, electrodo
                        // derecho (RIGHT) conectado, electrodo superior (TOP)desconectado)
    PORTC.F1 = 0;      // DRIVEB = 0 (electrodo inferior (BOTTOM) desconectado)
    Delay_ms(5);
    return ADC_Read(0); // leer el valor de X de RA0(BOTTOM)
}

// Leer la coordenada Y
unsigned int GetY() {
    //Leer la Y
    PORTC.F0 = 0;      // DRIVEA = 0 (electrodo izquierdo (LEFT) desconectado, electrodo
                        // derecho (RIGHT) desconectado, electrodo superior (TOP) conectado
    )

```



```

PORTC.F1 = 1;          // DRIVEB = 1 (electrodo inferior (BOTTOM) conectado)
Delay_ms(5);
return ADC_Read(1); // leer el valor de Y de RA1 (del electrodo izquierdo LEFT)
}

void main() {
    PORTA = 0x00;
    TRISA = 0x03; // RA0 y RA1 son entradas analógicas
    ANSEL = 0x03;
    ANSELH = 0;   // Configurar otros pines AN como digitales de E/S
    PORTC = 0 ;   // Todos los pines del puerto PORTC están a 0 (incluyendo los
                  // pines RC6 y RC7)

    TRISC = 0 ;   // PORTC es una salida

    // Inicialización del GLCD
    Glcd_Init();          // Glcd_Init_EP5
    Glcd_Set_Font(FontSystem5x7_v2, 5, 7, 32); // Seleccionar el tamaño de fuente 5x7
    Glcd_Fill(0);          // Borrar GLCD
    Glcd_Write_Text(msg1,10,0,1);
    Glcd_Write_Text(msg2,17,7,1);

    // Visualizar botones en el GLCD:
    Glcd_Rectangle(8,16,60,48,1);
    Glcd_Rectangle(68,16,120,48,1);
    Glcd_Box(10,18,58,46,1);
    Glcd_Box(70,18,118,46,1);

    // Visualizar los mensajes en los botones
    Glcd_Write_Text(msg3,14,3,0);
    Glcd_Write_Text(msg5,14,4,0);
    Glcd_Write_Text(msg4,74,3,0);
    Glcd_Write_Text(msg6,74,4,0);

    while (1) {
        // leer X-Y y convertirlo en la resolución de 128x64 píxeles
        x_coord = GetX();
        y_coord = GetY();
    }
}

```

```

x_coord128 = (x_coord * 128) / 1024;
y_coord64 = 64 - ((y_coord * 64) / 1024);

//Si BUTTON1 ha sido presionado
if ((x_coord128 >= 10) && (x_coord128 <= 58) && (y_coord64 >= 18) &&
    (y_coord64 <= 46)) {
    if(PORTC.F6 == 0) { // Si RC6 = 0
        PORTC.F6 = 1; // Invertir el estado lógico del pin RC6
        Glcd_Write_Text(msg7,14,4,0); // Visualizar un nuevo mensaje: RC6 ON
    }
    else { // Si RC6 = 1
        PORTC.F6 = 0; // Invertir el estado lógico del pin RC6
        Glcd_Write_Text(msg5,14,4,0); // Visualizar un nuevo mensaje: RC6 OFF
    }
}

// Si BUTTON2 ha sido presionado
if ((x_coord128 >= 70) && (x_coord128 <= 118) && (y_coord64 >= 18) &&
    (y_coord64 <= 46)) {
    if(PORTC.F7 == 0) { // Si RC7 = 0
        PORTC.F7 = 1; // Invertir el estado lógico del pin RC7
        Glcd_Write_Text(msg8,74,4,0); // Visualizar un nuevo mensaje: RC7 ON
    }
    else { // Si RC7 = 1
        PORTC.F7 = 0; // Invertir el estado lógico del pin RC7
        Glcd_Write_Text(msg6,74,4,0); // Visualizar un nuevo mensaje: RC7 OFF
    }
}
Delay_ms(100);
}
}

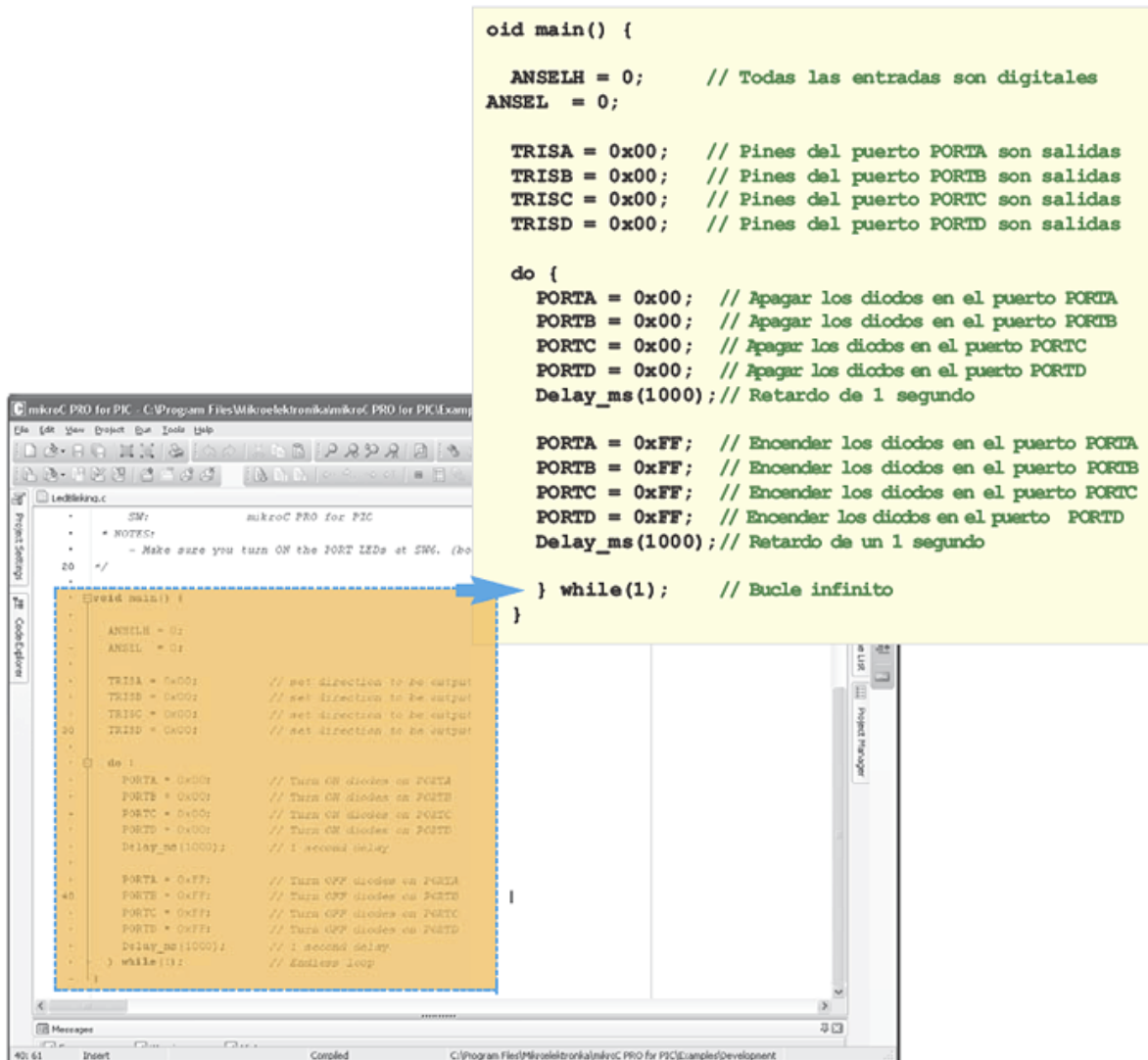
```

Para que este ejemplo funcione apropiadamente, es necesario marcar las siguientes librerías en la ventana *Library Manager* antes de compilar el programa.

- GLCD
- ADC
- C Stdlib

A1. VAMOS A EMPEZAR...

Los programas especiales en el entorno de Windows se utilizan para escribir un programa para el microcontrolador. Este libro describe el programa denominado mikroC PRO for PIC. La ventaja principal de este programa son las herramientas adicionales instaladas para facilitar el proceso de desarrollo. Si tiene experiencia en escribir programas, entonces sabe que se trata de escribir todas las instrucciones en el orden en el que se deben ejecutar por el microcontrolador y observar las reglas del lenguaje C. En otras palabras, sólo tiene que seguir su idea al escribir el programa. ¡Esto es todo!



```
oid main() {  
  
    ANSELH = 0;    // Todas las entradas son digitales  
    ANSEL = 0;  
  
    TRISA = 0x00;  // Pines del puerto PORTA son salidas  
    TRISB = 0x00;  // Pines del puerto PORTB son salidas  
    TRISC = 0x00;  // Pines del puerto PORTC son salidas  
    TRISD = 0x00;  // Pines del puerto PORTD son salidas  
  
    do {  
        PORTA = 0x00; // Apagar los diodos en el puerto PORTA  
        PORTB = 0x00; // Apagar los diodos en el puerto PORTB  
        PORTC = 0x00; // Apagar los diodos en el puerto PORTC  
        PORTD = 0x00; // Apagar los diodos en el puerto PORTD  
        Delay_ms(1000); // Retardo de 1 segundo  
  
        PORTA = 0xFF; // Encender los diodos en el puerto PORTA  
        PORTB = 0xFF; // Encender los diodos en el puerto PORTB  
        PORTC = 0xFF; // Encender los diodos en el puerto PORTC  
        PORTD = 0xFF; // Encender los diodos en el puerto PORTD  
        Delay_ms(1000); // Retardo de un 1 segundo  
  
    } while(1);    // Bucle infinito  
}
```

A.2 COMPILACIÓN DE PROGRAMA

El microcontrolador no entiende los lenguajes de alto nivel de programación, de ahí que sea necesario compilar el programa en lenguaje máquina. Basta con pulsar sólo una vez sobre el icono apropiado dentro del compilador para crear un documento nuevo con extensión .hex. En realidad, es el mismo programa, pero compilado en lenguaje máquina que el microcontrolador entiende perfectamente. Este programa se le denomina con frecuencia un código hex y forma una secuencia de números hexadecimales aparentemente sin significado.

```
:03000000020100FA1001000075813F
7590FFB29012010D80F97A1479D40
90110003278589EAF3698E8EB25B
A585FEA2569AD96E6D8FED9FAD
AF6DD00000001FF255AFED589EA
F3698E8EB25BA585FEA2569AD96
DAC59700D00000278E6D8FED9FA
DAF6DD00000001FF255AFED8FED
9FADAF6DD000F7590FFB29013278
E6D8FED9FADAF6DD00000001FF2
55AFED589EAF3698E8EB25BA585
FEA2569AD96DAC59D9FADAF6D
D00000001FF255AFED8FED9FADA
F6DD000F7590FFB29013278E6D82
78E6D8FED9FA589EAF3698E8EB2
5BA585FEA2569AD96DAF6DD000
00001FF2DAF6DD00000001FF255A
ADAF6DD00000001FF255AFED8FE
D9FA
```

Una vez compilado, el programa se debe cargar en el chip. Usted necesita un hardware apropiado para hacerlo posible - un programador.

PROGRAMAR EL MICROCONTROLADOR

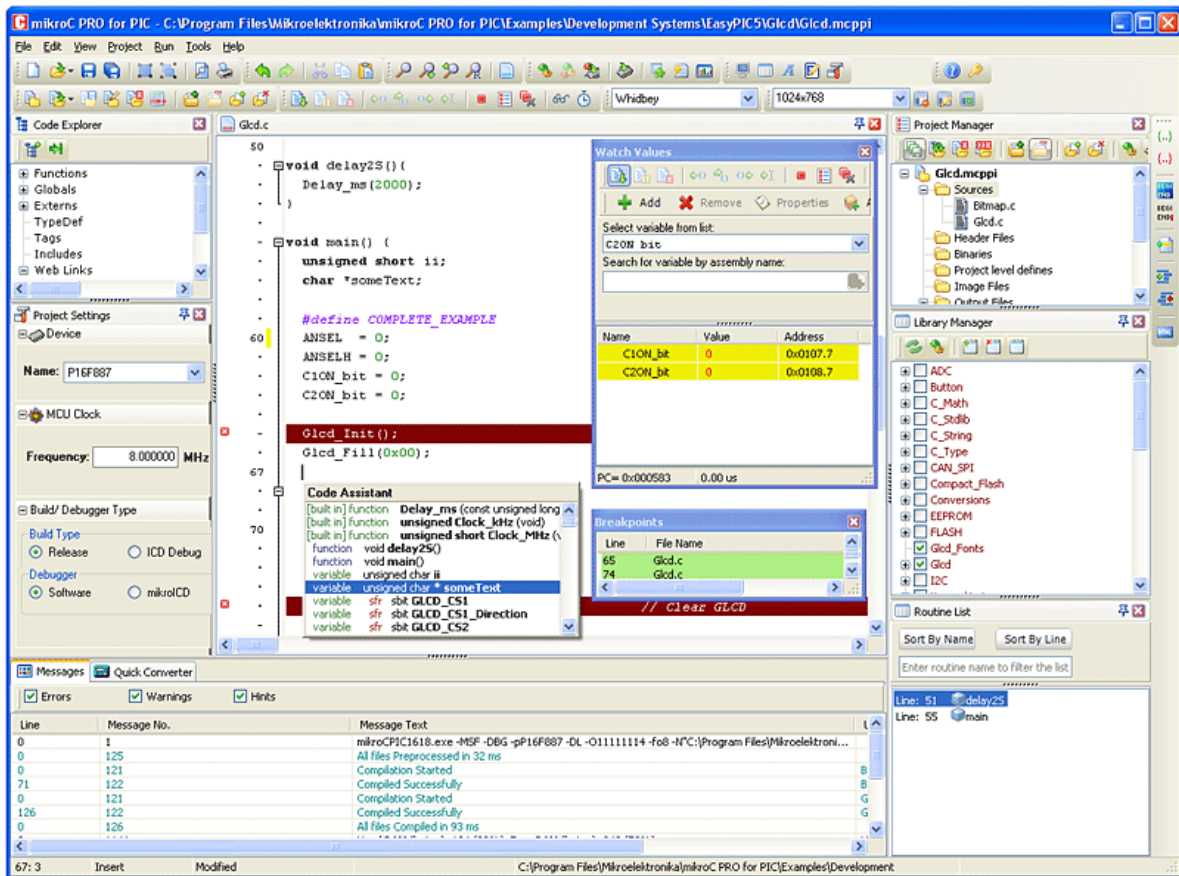
Como hemos mencionado, para habilitar cargar un código hex en el microcontrolador es necesario proporcionar un dispositivo especial, denominado el programador, con software apropiado. Un gran número de programas y circuitos electrónicos utilizados con este propósito se pueden encontrar en Internet. El procedimiento es básicamente el mismo para todos ellos y se parece a lo siguiente:

1. Coloque el microcontrolador en el zócalo apropiado del programador;
2. Utilice un cable adecuado para conectar el programador a una PC;
3. Abra el programa en código hex dentro de software del programador, ajuste varios parámetros, y pulse sobre el icono para transmitir el código. Pocos segundos después, una secuencia de ceros y unos se va a programar en el microcontrolador.

Sólo ha quedado instalar el chip programado en el dispositivo destino. Si es necesario hacer algunos cambios en el programa, el procedimiento anterior se puede repetir un número ilimitado de veces.

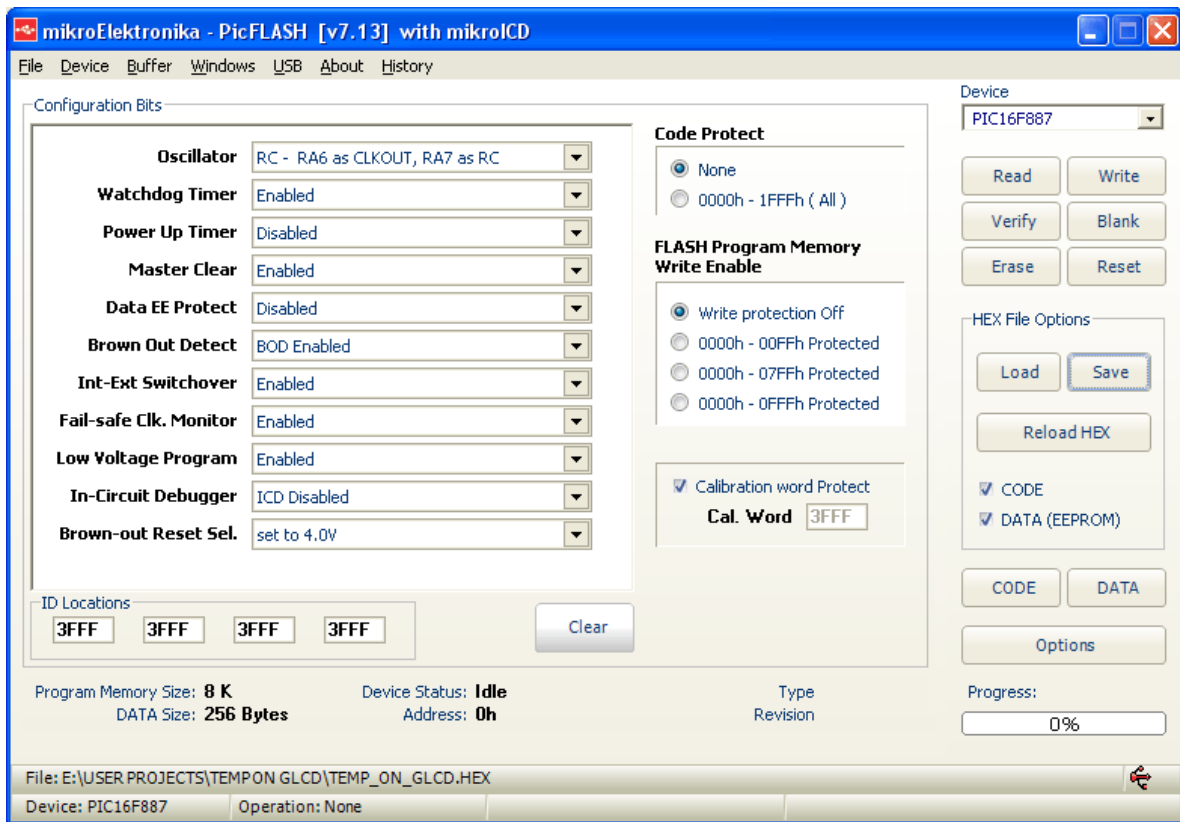
A.3 ¿SERÁ UN FINAL FELIZ?

Esta sección describe en breve el uso del programa (compilador) *mikroC PRO for PIC* y del software de programación (programador) *PICflash*. Todo es muy simple... Usted ya tiene instalado el *mikroC PRO for PIC*, ¿verdad? Al iniciarlo, abra un proyecto nuevo y un documento nuevo con extensión *.c* dentro del mismo. Escriba su programa...



De acuerdo. El programa ha sido escrito y probado con el simulador. ¿No ha informado de ningún error durante el proceso de compilación en el código hex? Parece que todo funciona perfecto... El programa ha sido compilado con éxito. Sólo queda cargarlo en el microcontrolador. Ahora necesita un *programador* que está compuesto por software y hardware. Inicie el programa *PICFlash*. La configuración es simple y no hacen falta explicaciones adicionales (tipo de microcontrolador, frecuencia y reloj del oscilador etc.).

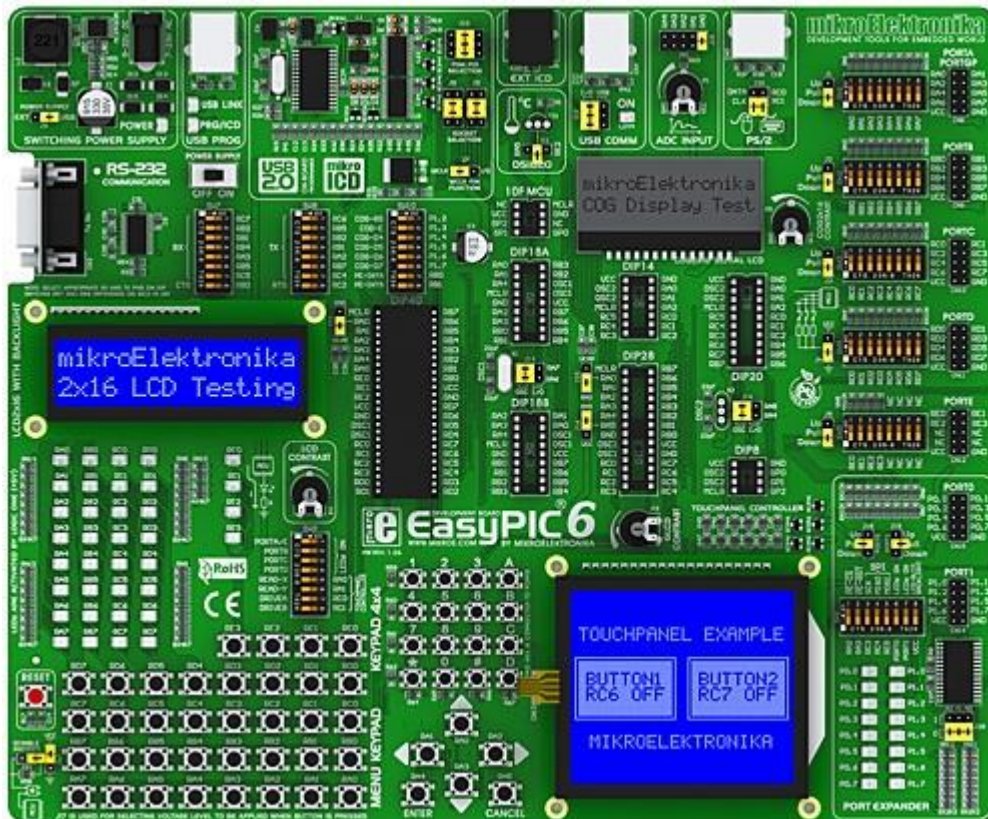
- Conecte la PC con el hardware del programador por un cable USB;
- Cargue el código hex utilizando el comando: File a Load HEX; y
- Pulse sobre el botón Write y espere...



¡Esto es todo! El microcontrolador está programado y todo está listo para su funcionamiento. Si no está satisfecho, haga algunos cambios en el programa y repita el procedimiento. ¿Hasta cuándo? Hasta quedar satisfecho...

A.4 SISTEMAS DE DESARROLLO

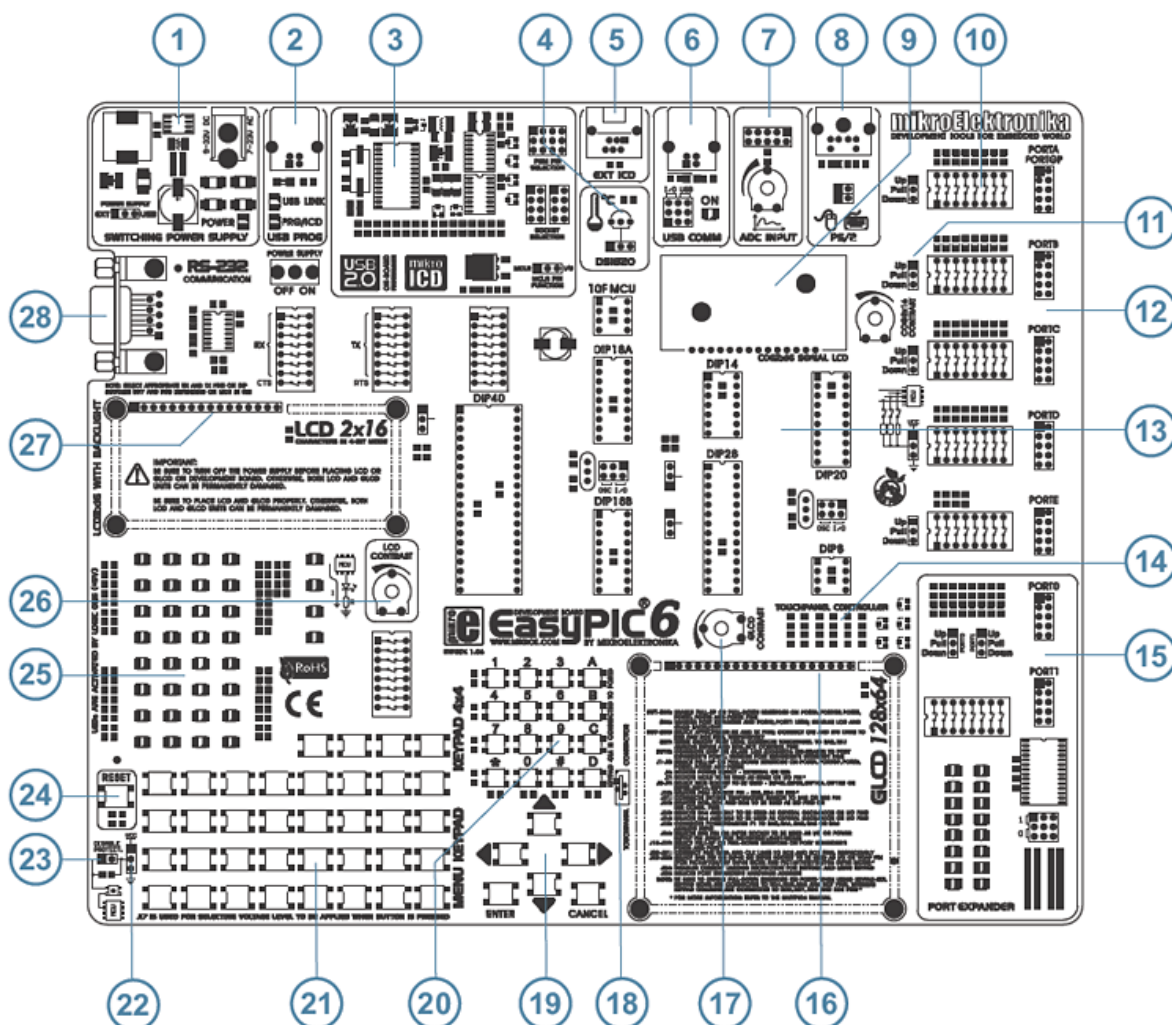
Un dispositivo que puede simular cualquier dispositivo en la fase de prueba, es denominado un sistema de desarrollo. Aparte del programador, unidad de alimentación, zócalo del microcontrolador, el sistema de desarrollo dispone de los componentes para activar los pines de entrada y monitorear los pines de salida. La versión más simple tiene cada pin conectado a su respectivo botón de presión y un



LED. Una

versión de calidad alta tiene los pines conectados a los visualizadores LED, visualizadores LCD, sensores de temperatura u otros componentes por los que puede estar compuesto un dispositivo destino. Si es necesario, todos estos periféricos pueden estar conectados al microcontrolador por medio de los puentes. Esto permite probar el programa entero en la práctica aún durante el proceso de desarrollo, porque el microcontrolador no “sabe o no le interesa” si su entrada está activada por un botón de presión o un sensor incorporado en un dispositivo real. Si dispone de un sistema de desarrollo, el proceso de programar y probar un programa es aún más sencillo. Teniendo en cuenta que el compilador mikroC PRO for PIC (en su PC) y el hardware del programador PICflash (en su sistema de desarrollo) colaboran perfectamente, el proceso de compilar un programa y programar el microcontrolador se lleva a cabo en un simple paso - al pulsar sobre el icono Build and Program dentro del compilador. Desde este momento, cualquier cambio en el programa afectará inmediatamente al funcionamiento de alguno de los componentes del sistema de desarrollo. ¿Está de acuerdo con nosotros que es hora de divertirse?

Características principales del sistema de desarrollo EasyPIC6



1. Regulador de voltaje de alimentación
2. Conector USB para el programador en la placa
3. Programador USB 2.0 con soporte de mikroICD
4. Zócalo para el sensor de temperatura DS1820
5. Conector para el depurador externo (ICD2 o ICD3) de Microchip
6. Conector para la comunicación USB
7. Entradas de prueba del convertidor A/D
8. Conector PS/2
9. LCD 2x16 en la placa
10. Interruptores DIP permiten el funcionamiento de las resistencias pull-up/pull-down
11. Puente para seleccionar las resistencias pull-up/pull-down
12. Conectores de los puertos E/S

13. Zócalo para colocar el microcontrolador PIC
14. Controlador del panel tácti
15. Extensor de puertos
16. Conector del LCD gráfico128x64
17. Potenciómetro de contraste del LCD gráfico
18. Conector de panel táctil
19. Teclado Menu
20. Teclado 4x4
21. Botones de presión para simular las entradas digitales
22. Puente para seleccionar el estado lógico de los botones de presión
23. Puente para poner en cortocircuito la resistencia de protección
24. Botón para reiniciar el microcontrolador
25. 36 diodos LED indican el estado lógico de los pines
26. Ajuste de contraste del LCD alfanumérico
27. Conector del LCD alfanumérico
28. Conector para la comunicación RS-232