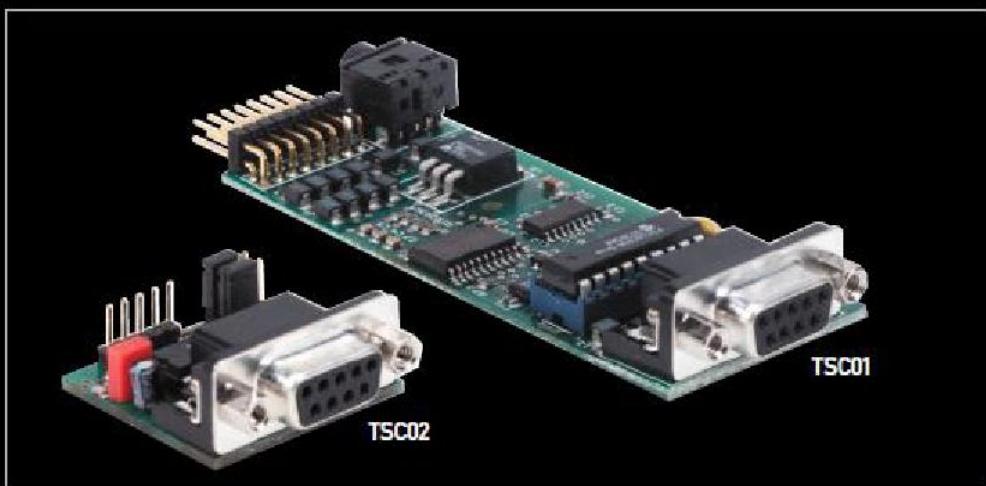
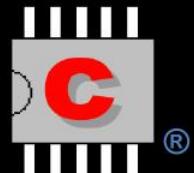


# **CÓMO PROGRAMAR EN LENGUAJE C LOS MICROCONTROLADORES PIC16F88, 16F628A Y 16F877A**

**SEGUNDA EDICIÓN 2010**



**JUAN RICARDO PENAGOS PLAZAS**



# ***Cómo programar en lenguaje C los microcontroladores PIC16F88, 16F628A y 16F877A***



***2da Edición***

***microC***

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

**©Juan Ricardo Penagos Plazas**

*Ingeniero en Electrónica y Telecomunicaciones*

*Profesor universitario de Programación, Circuitos Eléctricos, Electrónica y Sistemas Digitales*

[www.programarpicenc.com](http://www.programarpicenc.com)

**2010**

Cómo programar en lenguaje C los microcontroladores PIC16F88, 16F628A y 16F877A, 2da edición.  
©Juan Ricardo Penagos Plazas

MARCAS COMERCIALES: El autor ha intentado a lo largo de este libro distinguir las marcas registradas de los términos descriptivos, siguiendo el estilo de mayúsculas que utiliza el fabricante, sin intención de infringir la marca y sólo en beneficio del propietario de la misma:

**PIC16F88, PIC16F628A y PIC16F877A** son propiedad de Microchip.

**PICkit2 y PICkit 2 v2.61** son propiedad de Microchip.

**IC-Prog** es propiedad de Bonny Gijzen.

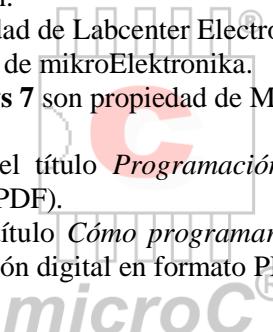
**PROTEUS, ISIS y ARES** son propiedad de Labcenter Electronics.

**mikroC™ PRO for PIC** es propiedad de mikroElektronika.

**Windows XP Professional y Windows 7** son propiedad de Microsoft.

Primera edición: Octubre 2009 con el título *Programación en lenguaje C del microcontrolador PIC16F88* (edición digital en formato PDF).

Segunda edición: Junio 2010 con el título *Cómo programar en lenguaje C los microcontroladores PIC16F88, 16F628A y 16F877A* (edición digital en formato PDF y edición impresa).



#### Derechos reservados.

No. de registro 1era edición: 032377 IEPI [programarpicenc.com](http://www.programarpicenc.com)

No. de registro 2da edición: 034985 IEPI Ricardo Penagos Plazas

Esta obra es propiedad intelectual de su autor, así como los derechos para su publicación. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

❖ **Este libro se escribió para ti, protégelo de la copia.**

#### NOTA IMPORTANTE

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. El autor no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, y en los archivos adjuntos, ni por la utilización indebida que pudiera dársele.

#### Ventas de este libro:

**microC®**

[www.programarpicenc.com](http://www.programarpicenc.com)

Teléfono celular: 087438877 (Movistar)

Quito-Ecuador

*A mi Dios por las incontables bendiciones que me ha brindado a lo largo de mi vida.*

*A mi gran Maestro, Jesucristo, por todos sus consejos de sabiduría divina  
y sus palabras llenas de esperanza.*

*A mi madre Carmen María; mi gran apoyo  
durante mis largas horas de estudio e investigación.*

*A mi padre Gustavo; mi hermano Jorge Enrique; mi hermana Lina Marcela por los ejemplos  
de lucha y valentía.*

*A todos mis alumnos universitarios, ejemplos de constancia y dedicación, por su inmenso  
respeto, paciencia y consideración durante estos diez años de docencia.*



www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

**Juan Ricardo**

## TABLA DE CONTENIDO

<b>CAPÍTULO I: PRIMEROS PASOS CON LOS PICS .....</b>	<b>1</b>
<b>1.1 MICROCONTROLADOR VS. MICROPROCESADOR.....</b>	<b>1</b>
<b>1.2 APLICACIONES DE LOS MICROCONTROLADORES .....</b>	<b>1</b>
<b>1.3 PIC16F84A VS. PIC16F88, PIC16F628A Y PIC16F877A.....</b>	<b>1</b>
<b>1.4 RESUMEN DE CARACTERÍSTICAS DE LOS PICS.....</b>	<b>3</b>
<b>1.5 CONFIGURACIÓN INICIAL BÁSICA DE LOS PICS.....</b>	<b>6</b>
1.5.1 Configuración inicial básica en mikroC™ .....	7
<b>1.6 MIKROC™.....</b>	<b>12</b>
1.6.1 Características de mikroC™ .....	12
1.6.2 Los menús de mikroC™.....	13
1.6.3 Proceso de simulación básica en mikroC™ .....	13
<b>1.7 CONCEPTOS BÁSICOS DE LENGUAJE C.....</b>	<b>15</b>
1.7.1 Estructura básica de un programa en lenguaje C (sin funciones) .....	15
1.7.2 Los siete elementos básicos de la programación .....	16
1.7.3 Funciones.....	19
1.7.4 Para tener en cuenta.....	21
<b>CAPÍTULO II: PUERTOS DIGITALES.....</b>	<b>22</b>
<b>2.1 PRINCIPALES CARACTERÍSTICAS.....</b>	<b>22</b>
<b>2.2 PUERTO A .....</b>	<b>23</b>
2.2.1 Características de las tecnologías de entrada/salida .....	24
<b>2.3 PUERTO B.....</b>	<b>24</b>
<b>2.4 PUERTOS A, B, C, D Y E DEL PIC16F877A.....</b>	<b>25</b>
<b>2.5 VALORES MÁXIMOS ABSOLUTOS .....</b>	<b>25</b>
<b>2.6 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>26</b>
<b>CAPÍTULO III: LCD .....</b>	<b>40</b>
<b>3.1 CONEXIÓN DEL LCD AL PIC .....</b>	<b>40</b>
<b>3.2 FUNCIONES DE MIKROC™ PARA LCD .....</b>	<b>40</b>
<b>3.3 EJEMPLOS DE MENSAJES DE TEXTO FIJOS Y EN MOVIMIENTO CON VALORES NUMÉRICOS .....</b>	<b>41</b>
<b>3.4 CREACIÓN DE CARACTERES ESPECIALES.....</b>	<b>45</b>
3.4.1 RAM del Generador de Caracteres (CGRAM).....	46
3.4.2 Herramienta LCD Custom Character de mikroC™ .....	46
<b>CAPÍTULO IV: EEPROM DE DATOS.....</b>	<b>48</b>
<b>4.1 FUNCIONES DE MIKROC™ PARA LA EEPROM .....</b>	<b>48</b>
<b>4.2 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>48</b>

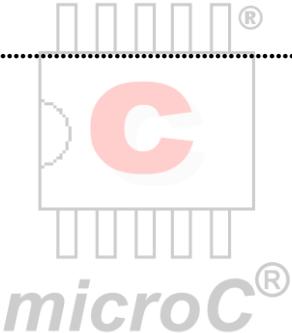
<b>CAPÍTULO V: TEMPORIZADORES Y CONTADORES.....</b>	<b>50</b>
<b>5.1 TIMER0 (16F88, 16F628A Y 16F877A) .....</b>	<b>50</b>
5.1.1 Prescaler .....	50
5.1.2 Ejemplos de programación del Timer0.....	51
<b>CAPÍTULO VI: CARACTERÍSTICAS ESPECIALES Y OTROS RECURSOS.....</b>	<b>54</b>
<b>PIC16F88.....</b>	<b>54</b>
<b>6.1 RESET .....</b>	<b>54</b>
<b>6.2 RESET MAESTRO #MCLR.....</b>	<b>56</b>
<b>6.3 RESET AL ENCENDIDO (POWER-ON RESET POR).....</b>	<b>56</b>
<b>6.4 TEMPORIZADOR DE ENCENDIDO (POWER-UP TIMER PWRT) .....</b>	<b>56</b>
<b>6.5 TEMPORIZADOR DE ENCENDIDO DEL OSCILADOR (OSCILATOR START-UP TIMER OST) .....</b>	<b>56</b>
<b>6.6 RESET POR DESVANECIMIENTO (BROWN-OUT RESET BOR) .....</b>	<b>56</b>
<b>6.7 SECUENCIA GENERAL DE ENCENDIDO .....</b>	<b>56</b>
<b>6.8 TEMPORIZADOR DE VIGILANCIA (WATCHDOG TIMER WDT) .....</b>	<b>57</b>
6.8.1 Oscilador WDT .....	57
6.8.2 Control del WDT .....	58
<b>6.9 MODO DE ENCENDIDO DE DOBLE VELOCIDAD.....</b>	<b>58</b>
6.9.1 Secuencia de encendido en doble velocidad.....	59
<b>6.10 OSCILADOR DE SEGURIDAD (FAIL-SAFE CLOCK MONITOR FSCM) .....</b>	<b>59</b>
<b>6.11 MODO DE BAJO CONSUMO (SLEEP).....</b>	<b>59</b>
6.11.1 Despertar (Wake-up from Sleep).....	59
6.11.2 Despertar usando interrupciones.....	60
<b>PIC16F628A Y 16F877A.....</b>	<b>60</b>
<b>6.12 RESET .....</b>	<b>60</b>
<b>6.13 RESET AL ENCENDIDO (POWER-ON RESET POR).....</b>	<b>63</b>
<b>6.14 TEMPORIZADOR DE ENCENDIDO (POWER-UP TIMER PWRT) .....</b>	<b>63</b>
<b>6.15 TEMPORIZADOR DE ENCENDIDO DEL OSCILADOR (OSCILATOR START-UP TIMER OST).....</b>	<b>63</b>
<b>6.16 RESET POR DESVANECIMIENTO (BROWN-OUT RESET BOR) .....</b>	<b>64</b>
<b>6.17 SECUENCIA GENERAL DE ENCENDIDO .....</b>	<b>64</b>
<b>6.18 TEMPORIZADOR DE VIGILANCIA (WATCHDOG TIMER WDT) .....</b>	<b>65</b>
<b>6.19 MODO DE BAJO CONSUMO (SLEEP) .....</b>	<b>66</b>
6.19.1 Despertar (Wake-up from Sleep).....	66
<b>6.20 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>66</b>
<b>CAPÍTULO VII: INTERRUPCIONES.....</b>	<b>69</b>
<b>7.1 INTERRUPCIÓN INT .....</b>	<b>70</b>
<b>7.2 INTERRUPCIÓN DEL TIMER0 .....</b>	<b>70</b>
<b>7.3 INTERRUPCIÓN RB .....</b>	<b>70</b>
<b>7.4 MANEJO DE INTERRUPCIONES EN MIKROC™.....</b>	<b>71</b>

<b>7.5 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>71</b>
<b>CAPÍTULO VIII: TECLADO MATRICIAL.....</b>	<b>88</b>
<b>8.1 FUNCIONES DE MIKROC™ PARA TECLADO .....</b>	<b>88</b>
<b>8.2 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>89</b>
<b>CAPÍTULO IX: PERIFÉRICOS .....</b>	<b>93</b>
<b>9.1 MODULACIÓN DE ANCHO DE PULSO (PWM).....</b>	<b>93</b>
9.1.1 Período PWM .....	94
9.1.2 Ciclo de trabajo PWM .....	94
9.1.3 Funciones de mikroC™ para PWM .....	94
9.1.4 Ejemplos de programación PWM.....	95
<b>9.2 CONVERTIDOR A/D DE 7 CANALES (PIC16F88 Y 16F877A) .....</b>	<b>97</b>
9.2.1 Selección del reloj de conversión .....	99
9.2.2 Registros de resultado.....	99
9.2.3 Funciones de mikroC™ para conversión A/D.....	100
9.2.4 Ejemplos de programación de conversión A/D.....	100
<b>CAPÍTULO X: COMUNICACIÓN CON EL ORDENADOR .....</b>	<b>102</b>
<b>10.1 ESTÁNDAR RS232 .....</b>	<b>102</b>
<b>10.2 CIRCUITO INTEGRADO MAX232.....</b>	<b>102</b>
<b>10.3 CONEXIÓN DEL PIC AL ORDENADOR.....</b>	<b>103</b>
<b>10.4 FUNCIONES DE MIKROC™ PARA EL MÓDULO USART.....</b>	<b>105</b>
<b>10.5 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>106</b>
<b>CAPÍTULO XI: BUS I<sup>2</sup>C.....</b>	<b>112</b>
<b>11.1 CARACTERÍSTICAS DEL BUS I<sup>2</sup>C .....</b>	<b>112</b>
<b>11.2 FUNCIONES DE MIKROC™ PARA EL BUS I<sup>2</sup>C .....</b>	<b>113</b>
<b>11.3 MEMORIA EEPROM EN BUS I<sup>2</sup>C .....</b>	<b>114</b>
11.3.1 Direccionamiento como esclavo.....	114
11.3.2 Operaciones de escritura.....	115
11.3.3 Operaciones de lectura .....	116
<b>11.4 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>117</b>
<b>CAPÍTULO XII: MOTORES DC Y PASO A PASO (PAP) .....</b>	<b>121</b>
<b>12.1 CONTROLADOR L293B.....</b>	<b>121</b>
<b>12.2 CONTROLADOR L293D.....</b>	<b>123</b>
<b>12.3 CONEXIÓN DEL DRIVER L293D AL PIC .....</b>	<b>123</b>
<b>12.4 MOTORES PASO A PASO-PAP (STEPPER / STEPPING MOTOR).....</b>	<b>127</b>
12.4.1 Motores PAP bipolares.....	127

12.4.2 Motores PAP unipolares .....	127
12.4.3 Configuración de las bobinas .....	128
12.4.4 Conexión de un motor bipolar al PIC .....	129
12.4.5 Conexión de un motor unipolar al PIC .....	131
<b>CAPÍTULO XIII: LCD GRÁFICO (<i>GRAPHIC LCD GLCD</i>).....</b>	<b>134</b>
<b>13.1 TIPOS DE LCD .....</b>	<b>134</b>
13.1.1 LCD de segmentos o alfanumérico.....	134
13.1.2 LCD de matriz de puntos (o LCD de caracteres).....	134
13.1.3 LCD para gráficos ( <i>graphic LCD GLCD</i> ).....	134
<b>13.2 TECNOLOGÍA LCD .....</b>	<b>135</b>
13.2.1 Efecto de campo del LCD .....	136
<b>13.3 TIPOS DE LÍQUIDOS PARA LCDS .....</b>	<b>136</b>
13.3.1 LCDs Twisted Nematic (TN) .....	136
13.3.2 LCDs High Twisted Nematic (HTN) .....	137
13.3.3 LCDs Super Twisted Nematic (STN).....	137
13.3.4 LCDs Film Compensated STN (FSTN) .....	137
13.3.5 LCDs Double Super Twisted Nematic (DSTN) .....	137
13.3.6 LCDs Color Super Twisted Nematic (CSTN) .....	137
<b>13.4 MODOS DE LCD .....</b>	<b>138</b>
13.4.1 Modo positivo.....	138
13.4.2 Modo negativo.....	138
<b>13.5 MODOS DE POLARIZADOR.....</b>	<b>138</b>
13.5.1 Polarizador reflectante .....	138
13.5.2 Polarizador transmisor .....	139
13.5.3 Polarizador transreflectivo .....	139
13.5.4 Polarizador de grado comercial .....	139
13.5.5 Polarizador de grado industrial .....	139
<b>13.6 ÁNGULO DE VISIÓN.....</b>	<b>140</b>
<b>13.7 RANGOS DE TEMPERATURA .....</b>	<b>140</b>
13.7.1 Rangos de temperatura TN .....	140
13.7.2 Rangos de temperatura STN/FSTN .....	141
<b>13.8 LUZ DE FONDO DEL LCD .....</b>	<b>141</b>
13.8.1 Alumbrado de fondo con LED ( <i>LED backlighting</i> ) .....	141
13.8.2 Panel electroluminiscente ( <i>Electroluminescence Panel ELP</i> ) .....	142
13.8.3 Lámpara fluorescente de cátodo frío ( <i>Cold Cathode Fluorescent Lamp CCFL</i> ).....	142
13.8.4 Malla de fibra óptica en zigzag ( <i>Woven Fiber Optic Mesh WFOM</i> ) .....	143
13.8.5 Alumbrado incandescente .....	143
<b>13.9 TIPOS DE MONTAJES DE MÓDULOS LCD .....</b>	<b>143</b>
13.9.1 Tecnología de montaje superficial ( <i>Surface Mount Technology SMT</i> ).....	143

13.9.2 Chip en tablero ( <i>Chip On Board COB</i> ) .....	144
13.9.3 Unión automatizada de cinta ( <i>Tape Automated Bonding TAB</i> ) .....	144
13.9.4 Chip en vidrio ( <i>Chip On Glass COG</i> ) .....	145
<b>13.10 CIRCUITERÍA DE CONTROL DEL LCD: ESTÁTICO.....</b>	<b>145</b>
<b>13.11 CIRCUITERÍA DE CONTROL DEL LCD: MULTIPLEXADO .....</b>	<b>146</b>
<b>13.12 CONTROL DE CONTRASTE DEL LCD.....</b>	<b>147</b>
13.12.1 Ajuste del contraste del LCD.....	147
13.12.2 Ajuste de contraste con compensación de temperatura .....	148
<b>13.13 FUNCIONES DE MIKROC™ PARA GLCD .....</b>	<b>149</b>
<b>13.14 CONTROLADOR T6963C.....</b>	<b>149</b>
13.14.1 Definiciones.....	150
13.14.2 Diagrama general de aplicación del T6963C.....	150
13.14.3 Conexión a una pantalla .....	151
13.14.4 Conexión a un microcontrolador y a la RAM de video .....	151
13.14.5 Configuración del T6963C .....	153
<b>13.15 LM4228: LCD GRÁFICO 128X64 (GLCD 128X64) .....</b>	<b>155</b>
13.15.1 Precauciones .....	155
13.15.2 Valores máximos absolutos .....	156
13.15.3 Características eléctricas.....	156
13.15.4 Especificaciones de la luz de fondo .....	156
13.15.5 Fuente de alimentación.....	157
13.15.6 Descripción de los pines .....	158
13.15.7 Diagrama de bloques .....	158
13.15.8 Dimensiones del módulo .....	159
13.15.9 Descripción del número de parte para las opciones disponibles.....	159
<b>13.16 EJEMPLOS DE PROGRAMACIÓN.....</b>	<b>160</b>
<b>APÉNDICE A: PROGRAMADOR AN589.....</b>	<b>168</b>
<b>ESQUEMA ELÉCTRICO .....</b>	<b>168</b>
<b>CIRCUITO IMPRESO PCB .....</b>	<b>168</b>
<b>SIMBOLOGÍA (SILK) .....</b>	<b>169</b>
<b>APÉNDICE B: SOFTWARE IC-PROG .....</b>	<b>170</b>
<b>INSTALACIÓN EN WINDOWS XP .....</b>	<b>170</b>
<b>PROGRAMACIÓN DE LOS PICS 16F88, 16F628A Y 16F877A.....</b>	<b>170</b>
<b>APÉNDICE C: SIMULACIÓN BÁSICA EN ISIS® DE PROTEUS®.....</b>	<b>172</b>
<b>CONSTRUCCIÓN DEL ESQUEMA ELÉCTRICO .....</b>	<b>172</b>
<b>SIMULACIÓN.....</b>	<b>176</b>
<b>SOBRECARGA DEL ORDENADOR.....</b>	<b>177</b>

SIMULACIÓN DEL MÓDULO USART .....	178
APÉNDICE D: PROGRAMADOR PICKIT2 CLONE PARA PUERTO USB .....	179
CIRCUITO IMPRESO VISTO DESDE LA CARA DE COMPONENTES.....	179
UBICACIÓN DE COMPONENTES .....	179
ESQUEMA ELÉCTRICO DEL PROGRAMADOR .....	180
PROCEDIMIENTO DE PROGRAMACIÓN .....	180
 APÉNDICE E: CONSIDERACIONES PRÁCTICAS .....	182
PINES NO UTILIZADOS .....	182
RESET INDESEADO #MCLR .....	182
RESET POR DESVANECIMIENTO (BOR) .....	183
FUNCIONAMIENTO ERRÁTICO DEL PIC .....	183
FUENTES DE ALIMENTACIÓN.....	183
LCD ALFANUMÉRICO (CON EL CONTROLADOR HD44780 O COMPATIBLE).....	184
CONEXIÓN DE UN RELÉ ELECTROMECÁNICO AL PIC.....	184
 ÍNDICE ALFABÉTICO .....	185



www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## PRÓLOGO

El uso de los microcontroladores en las aplicaciones de electrónica digital y analógica se ha popularizado a partir de la reducción de sus precios y la integración cada vez mayor de una gran variedad de periféricos, así como el incremento en la capacidad de memoria de datos y programa, en este sentido el PIC16F88, el PIC16F628A y el PIC16F877A pueden considerarse como tres de las mejores alternativas a la hora de seleccionar un microcontrolador, debido a sus altas prestaciones y precio reducido, si se los compara con el conocido PIC16F84A, que presenta grandes limitaciones en periféricos y memoria, con un precio relativamente elevado (de acuerdo a la información obtenida de la pagina del fabricante: [www.microchip.com](http://www.microchip.com)).

Por otra parte, hoy en día se continúa empleando ampliamente el lenguaje ensamblador para la programación de las aplicaciones con microcontroladores, a pesar de las grandes complicaciones y desventajas que implica, y que seguramente serán conocidas por los lectores: muchas líneas de código, aun para las aplicaciones más simples, falta de librerías de funciones que hagan más eficiente la programación, excesivo tiempo requerido para el desarrollo de una aplicación, conocimiento detallado del hardware y del set de instrucciones, etc. Una de las causas para que esto sea así es la escasez de bibliografía y ejemplos prácticos existentes acerca de cómo desarrollar aplicaciones con microcontroladores PIC en lenguajes de alto nivel, tales como el muy difundido y conocido lenguaje C. Otra es la falta de difusión de los ambientes de desarrollo integrado para lenguajes de alto nivel, a pesar de que existen varios de ellos desde hace algunos años, por ejemplo: CCS, PICC, DEM ICT, PCW, PICBASIC PRO, mikroC<sup>TM</sup> PRO for PIC, entre otros.

Existen muchos libros sobre microcontroladores, pero en la gran mayoría hay una vasta cantidad de conceptos y definiciones teóricas, en volúmenes de unas 400 páginas a enciclopedias de unas 800 páginas en varios tomos, que sinceramente no brindan solución a la principal necesidad del estudiante: la implementación práctica de los conceptos teóricos, por lo cual al final de largas horas de estudio el alumno se siente decepcionado, pues conoce mucha teoría y no sabe cómo llevarla a la práctica.

Con todos estos antecedentes en mente y pensando en esta necesidad clamorosa se ha desarrollado este texto, que sin dejar de lado los conceptos teóricos relevantes, se enfoca principalmente en la solución de ejemplos de problemas prácticos, de manera directa y eficiente, integrando de manera sistemática y ordenada todos los detalles de la programación en lenguaje C de los microcontroladores PIC16F88, PIC16F628A y PIC16F877A, desde los aspectos más básicos hasta los más avanzados, permitiendo de esta manera hacer uso eficiente de los recursos en el desarrollo de proyectos de control electrónico, obteniendo los mejores resultados con mínimo esfuerzo.

Una vez que se conocen los procedimientos de programación de un PIC específico se puede emprender la tarea de programar cualquier otro tipo de PIC, sin mayor dificultad. La innovación contenida en este libro consiste en la explicación de los métodos de aplicación de las nuevas herramientas de programación en la solución de problemas con microcontroladores PIC.

**Esta segunda edición de *Cómo programar en lenguaje C los microcontroladores PIC16F88, 16F628A y 16F877A* trae nuevos capítulos y mejoras relevantes con respecto a la primera edición:**

- **¡NUEVO! Se han añadido cuatro capítulos de fundamental importancia para la realización de proyectos en el mundo actual:** Capítulo X: Comunicación con el ordenador (empleando el estándar RS232); Capítulo XI: Bus I<sup>2</sup>C; Capítulo XII: Motores DC y paso a paso PAP; y, Capítulo XIII: LCD gráfico GLCD.
- **El Apéndice A ha sido revisado y se ha añadido la información necesaria, con imágenes incluidas, para que el estudiante pueda construir su propio programador de microcontroladores para el puerto paralelo.**
- **¡NUEVO! En el Apéndice C se explica en detalle cómo realizar simulaciones con microcontroladores PIC en el ambiente PROTEUS®.**
- **¡NUEVO! El Apéndice D contiene la información necesaria para la construcción del programador PICkit2 Clone para el puerto USB. También se explica el procedimiento de programación.**
- **¡NUEVO! El Apéndice E es un compendio de recomendaciones prácticas y consejos para construir circuitos en el mundo real, con el propósito de evitar posibles dificultades que no se observan durante la fase de simulación.**
- **Todos los ejemplos han sido revisados y se han hecho las correcciones necesarias para mejorar tanto la definición del problema, como los resultados obtenidos, con el fin de aclarar las ideas incorporadas en cada uno de ellos.**

- ¡NUEVO! Se ha añadido a cada proyecto del libro el diseño correspondiente en ISIS®, de tal manera que el lector pueda observar el resultado de las simulaciones desde el primer momento (los diseños en ISIS® están incluidos en la carpeta *Código Fuente* que se entrega con este libro).

Se asume que el lector domina los elementos básicos de la programación de computadoras en lenguaje C, tales como la entrada de datos, la declaración de tipos de variables, las operaciones lógicas y matemáticas, la ejecución condicional, los lazos y las funciones; también debe tener conocimientos de electrónica digital, tales como sistemas binario y hexadecimal y diseño con compuertas lógicas; una mínima experiencia en programación de microcontroladores PIC (por ejemplo el popular PIC16F84A); y lo más importante:

“UN FIRME DESEO DE APRENDER A REALIZAR SUS PROYECTOS DE ELECTRÓNICA DE FORMA EFICIENTE Y ACTUAL”.

Para el desarrollo de los ejemplos contenidos en este libro se han seleccionado las siguientes herramientas:

#### Software:

- mikroCTM PRO for PIC (mikroCTM) 3.2, ambiente de desarrollo integrado en lenguaje C, que se puede descargar de la página del fabricante. La versión gratuita de mikroCTM es completamente funcional, con todas las librerías, ejemplos y archivos de ayuda incluidos. La única limitación de la versión gratuita es que no se pueden generar archivos ejecutables (\*.hex) de más de 2k palabras de programa. Aún así, esto permite desarrollar todas las aplicaciones prácticas de variada complejidad de este libro. Si el lector lo requiere, debe comprar una licencia para poder retirar esta limitación.
- IC-Prog, software de grabación de microcontroladores PIC, probado con numerosos programadores. Se puede descargar de Internet. En el Apéndice B se explica en detalle cómo instalarlo y emplearlo.
- PICkit2 v2.61, software de grabación de microcontroladores PIC para el programador USB PICkit2 Clone. En el apéndice D se explica en detalle cómo utilizarlo.
- ISIS® de PROTEUS®, software de simulación de microcontroladores PIC, en el que se han probado con éxito todos los ejemplos resueltos en este libro. En el Apéndice C se explica cómo hacer simulaciones en este programa.

#### Hardware.-

- Programador AN589, la información para su construcción se puede encontrar en el Apéndice A. No obstante, se puede emplear cualquier programador compatible con el PIC16F84A, ya que es compatible con el 16F88 y el 16F628A (se debe hacer notar que para la programación se necesitan únicamente 5 pines, los cuales tienen la misma ubicación en los tres microcontroladores mencionados). El PIC16F877A también se puede programar si se realizan las conexiones adecuadas de los cinco pines de programación de acuerdo al estándar ICSP.
- Programador PICkit2 Clone para el puerto USB, la información para su construcción se puede encontrar en el Apéndice D.

## CAPÍTULO I: PRIMEROS PASOS CON LOS PICs

### 1.1 MICROCONTROLADOR VS. MICROPROCESADOR

Se suele pensar que un microcontrolador es lo mismo que un microprocesador, pero eso no es cierto. Se diferencian en varios aspectos. El primero y más importante a favor de un microcontrolador es su funcionalidad. Por otro lado, para que se pueda usar un microprocesador, se le deben añadir otros componentes, por ejemplo la memoria. Aunque un microprocesador es una poderosa máquina de cálculo, no está preparado para la comunicación con el entorno exterior; para que esto sea posible se le deben añadir circuitos especiales. Las primeras computadoras se hicieron al agregar periféricos externos tales como memoria, líneas de entrada/salida, temporizadores y otros al microprocesador. Con la mejora en la fabricación de circuitos integrados se logró incluir tanto el microprocesador como los periféricos en un solo chip. A este chip se le dio el nombre de microcontrolador (figura 1.1).

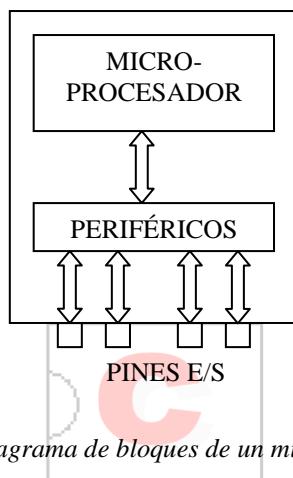


Figura 1.1 Diagrama de bloques de un microcontrolador

El microcontrolador está diseñado para agrupar todas las funciones en una sola unidad. No se necesitan componentes externos (periféricos) para su aplicación, porque todos los circuitos necesarios ya están incorporados. Esto significa ahorro de tiempo, dinero y espacio para un determinado diseño.

Existe una historia muy extensa de logros desde los primeros días del microcontrolador hasta los poderosos chips de la actualidad. ©Ing. Juan Ricardo Penagos Plazas

### 1.2 APPLICACIONES DE LOS MICROCONTROLADORES

Los microcontroladores se emplean en la actualidad en una inmensa variedad de aplicaciones de control electrónico: producción industrial, sector automotriz, aviación, construcción naval, electrodomésticos, exploración espacial, exploración marina, equipo médico, telecomunicaciones, robótica, etc. Su gran éxito se debe fundamentalmente a su enorme versatilidad, ya que el proceso de diseño se realiza casi exclusivamente en software (programación), mientras que en lo referente al hardware lo más importante es la selección del microcontrolador apropiado y nada más. Esto permite que el mantenimiento y la modificación de los diseños se haga de una forma muy eficiente. Las instrucciones del programa almacenado se encargan de decirle al microcontrolador lo que debe hacer bajo determinadas condiciones. Además, antes de la construcción del circuito de aplicación final, se pueden realizar todas las simulaciones y correcciones necesarias hasta lograr los resultados esperados. Si en alguna ocasión se desea cambiar, mejorar o potenciar la aplicación, simplemente se debe modificar el programa y listo.

### 1.3 PIC16F84A vs. PIC16F88, PIC16F628A y PIC16F877A

Para colocar frente a frente a estos microcontroladores y poder sacar una conclusión acerca de sus características, lo mejor es presentar la información que Microchip publica en su página de Internet, donde se tabulan todos los datos relativos a los productos disponibles actualmente. La información presentada es la siguiente:

Producto	Estado	Precio	Tipo de memoria	Memoria de programa kWords	Memoria de datos EEPROM
<b>PIC16F84A</b>	En producción	\$3,11	Flash	1	64
<b>PIC16F88</b>	En producción	\$2,20	Flash	4	256
<b>PIC16F628A</b>	En producción	\$1,47	Flash	2	128
<b>PIC16F877A</b>	En producción	\$4,26	Flash	8	256

Tabla 1.1 Características de los PICs 16F84A, 16F88, 16F628A y 16F877A

RAM	No. de pines E/S	No. total de pines	Máx. velocidad MHz	Oscilador interno	No. de canales A/D
68	13	18	20	NO	0
368	16	18	20	Hasta 8 MHz	7
224	16	18	20	4MHz	0
368	33	40	20	NO	8

Tabla 1.2 Características de los PICs 16F84A, 16F88, 16F628A y 16F877A (continuación)

Comunicación digital	Timers	Rango de temperatura	Rango de voltaje de operación	Encapsulado
NO	1 de 8 bit	-40 a 85°C	2 V a 6 V	18 PDIP
1 -A/E/USART 1 -SSP(SPI/I2C)	2 de 8 bit 1 de 16 bit	-40 a 125°C	2 V a 5,5V	18 PDIP
1 -A/E/USART	2 de 8 bit 1 de 16 bit	-40 a 125°C	2 V a 5,5V	18 PDIP
1 -A/E/USART 1 -MSSP(SPI/I2C)	2 de 8 bit 1 de 16 bit	-40 a 125°C	2 V a 5,5V	40 PDIP (0,6 in)

Tabla 1.3 Características de los PICs 16F84A, 16F88, 16F628A y 16F877A (continuación)

Los PICs 16F88 y 16F628A pertenecen a la gama media y pueden ser aplicados en circuitos electrónicos de propósito general. Se parecen al PIC16F84A en el número de pines y su distribución física. Como ventajas relevantes se pueden destacar las siguientes: menor precio, mayor capacidad de memoria de programa y datos, mayor disponibilidad de pines E/S, oscilador interno hasta 8 MHz (16F88) y 4MHz (16F628A), convertidor A/D (16F88), módulos de comunicación serie y tres temporizadores. Todas estas características los hacen completamente superiores al PIC16F84A, y por estas razones fueron seleccionados como modelos para la escritura del presente texto. Obsérvese que el PIC16F88 es el más completo y avanzado de los tres en cuestión, y su precio siendo un poco mayor que el del 16F628A, aún es menor que el del 16F84A.

El PIC16F877A incorpora funciones similares a los otros tres microcontroladores, se diferencia fundamentalmente en que es un microcontrolador de 40 pines y 0,6 pulgadas de ancho, tiene una memoria de programa de 8k que permite realizar aplicaciones muy extensas y gran cantidad de pines E/S (33 en total); estas características hacen que su precio sea un poco más elevado, aún así sigue siendo muy accesible. Una desventaja de este PIC es que no posee oscilador interno, lo que hace necesario la instalación de un cristal externo para su operación.

## 1.4 RESUMEN DE CARACTERÍSTICAS DE LOS PICS

### PIC16F88

El lector puede descargar la hoja de especificaciones *PIC16F87/88 Data Sheet (DS30487C)* de la página de Microchip, donde encontrará todos los detalles de este microcontrolador.

#### Características de bajo consumo:

- Modos de potencia controlada:
  - Oscilador primario: Oscilador RC, 76 uA, 1 MHz, 2 V
  - Modo RC\_RUN: 7 uA, 31,25 kHz, 2 V
  - Modo SEC\_RUN: 9 uA, 32 kHz, 2 V
  - Modo Sleep: 0,1 uA, 2 V
- Oscilador Timer1: 1,8 uA, 32 kHz, 2 V
- Temporizador de vigilancia (*Watchdog Timer WDT*): 2,2 uA, 2 V
- Encendido de doble velocidad

#### Osciladores:

- Tres modos de cristal:
  - LP, XT, HS: hasta 20 MHz
- Dos modos externos RC
- Un modo de reloj externo:
  - ECIO: hasta 20 MHz
- Bloque oscilador interno:
  - 8 frecuencias que pueden ser seleccionadas por el usuario: 31 kHz, 125 kHz, 250 kHz, 500 kHz, 1 MHz, 2 MHz, 4 MHz, 8 MHz

#### Periféricos:

- Módulo de Captura, Comparación y Modulación de Ancho de Pulso PWM (CCP):
  - Captura de 16 bits, resolución máxima 12,5 ns
  - Comparación de 16 bits, resolución máxima 200 ns
  - Resolución máxima PWM de 10 bits
- Convertidor A/D de 7 canales / 10 bits
- Puerto Serie Sincrónico (SSP) con SPI™ (Maestro/Escalvo) e I<sup>2</sup>C™ (Escalvo)
- Receptor Transmisor Sincrónico Asincrónico Universal Direccional (AUSART/SCI), con detección de direcciones de 9 bits:
  - Operación RS-232 empleando el oscilador interno (no se requiere cristal externo)
- Doble Módulo Comparador Analógico:
  - Referencia de voltaje programable en el chip
  - Entradas a los comparadores multiplexadas con los pines E/S del chip
  - Las salidas del comparador son accesibles externamente

#### Características especiales del microcontrolador:

El fabricante en su hoja de especificaciones afirma que la memoria de programa se puede grabar y borrar unas 100.000 veces, la memoria de datos EEPROM 1.000.000 de veces y sus datos permanecen almacenados por más de 40 años. Otras características son la Programación en Serie en el Circuito (ICSP™) que requiere un total de 5 pines, acceso del procesador a la memoria de programa, programación en bajo voltaje, depuración en el circuito por medio de 2 pines, temporizador de vigilancia extendido (período programable desde 1 ms hasta 268 s) y un rango amplio de voltaje de operación (desde 2 V a 5,5 V).

#### Diagrama de pines:

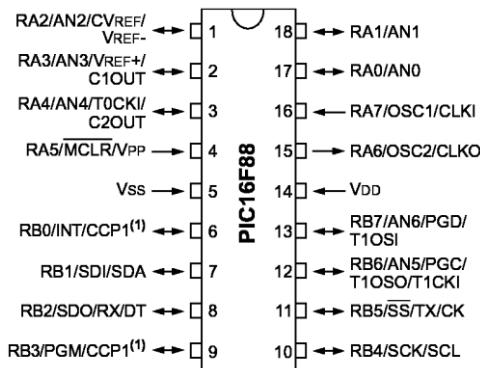


Figura 1.2.1 Distribución de terminales del PIC16F88

## **PIC16F628A**

El lector puede descargar la hoja de especificaciones *PIC16F627A/628A/648A Data Sheet (DS40044F)* de la página de Microchip, donde encontrará todos los detalles de este microcontrolador.

### **Microporcesador RISC de alto desempeño:**

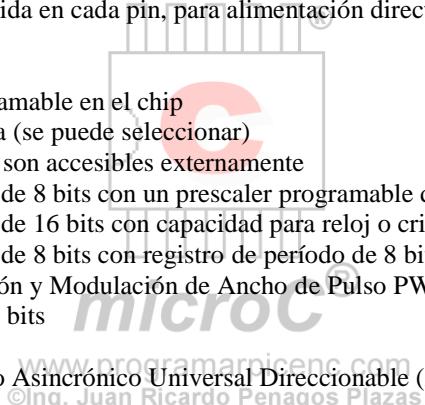
- Velocidad de operación desde DC hasta 20MHz
- Manejo de interrupciones
- Pila (*stack*) de 8 niveles
- 35 instrucciones de 1 palabra

### **Características de bajo consumo:**

- Corriente en reposo:
  - 100nA@2V
- Corriente de operación:
  - 12uA@32kHz, 2V
  - 120uA@1MHz, 2V
- Temporizador de vigilancia (*Watchdog Timer WDT*): 1uA@2V
- Corriente del oscilador del Timer1:
  - 1,2uA@32kHz, 2V
- Oscilador interno de doble velocidad:
  - Se puede seleccionar entre dos opciones: 4MHz y 48 kHz
  - Tiempo para despertar: 4us, 3V

### **Periféricos:**

- 16 pines E/S con control individual
- Elevada corriente de entrada/salida en cada pin, para alimentación directa de LEDs
- Comparador analógico:
  - Dos comparadores
  - Referencia de voltaje programable en el chip
  - Referencia interna o externa (se puede seleccionar)
  - Las salidas del comparador son accesibles externamente
- Timer0: contador/temporizador de 8 bits con un prescaler programable de 8 bits
- Timer1: contador/temporizador de 16 bits con capacidad para reloj o cristal externo
- Timer2: contador/temporizador de 8 bits con registro de período de 8 bits, prescaler y postscaler
- Módulo de Captura, Comparación y Modulación de Ancho de Pulso PWM:
  - Captura/comparación de 16 bits
  - PWM de 10 bits
- Receptor Transmisor Sincrónico Asincrónico Universal Direccional (AUSART/SCI)



### **Características especiales del microcontrolador:**

- Opciones de oscilador interno y externo:
  - Oscilador interno de precisión de 4MHz
  - Oscilador interno de 48kHz para modo de bajo consumo
  - Soporte de osciladores externos tipo cristal y resonador
- Modo de bajo consumo (*Sleep*)
- *Pull-ups* programables para el puertoB
- Pin multiplexado: Reset maestro/entrada
- WDT con oscilador independiente para mayor confiabilidad
- Programación en bajo voltaje
- Programación en serie en el circuito (ICSP)
- Protección de código programable
- Reset por desvanecimiento (BOR)
- Reset al encendido (POR)
- Amplio intervalo de voltajes de operación (2,0 a 5,5V)
- Memoria interna de gran duración:
  - 100.000 ciclos de escritura en la flash
  - 1.000.000 de ciclos de escritura en la EEPROM
  - Retención de datos durante 40 años

### **Diagrama de pines:**

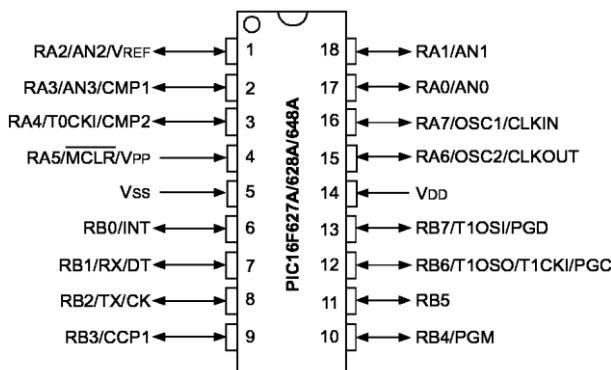


Figura 1.2.2 Distribución de terminales del PIC16F628A

## PIC16F877A

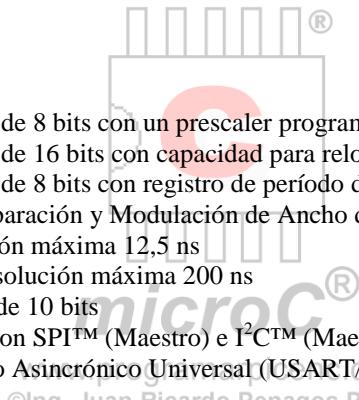
El lector puede descargar la hoja de especificaciones *PIC16F87XA Data Sheet (DS39582B)* de la página de Microchip, donde encontrará todos los detalles de este microcontrolador.

### **Microprocesador RISC de alto desempeño:**

- Instrucciones de un ciclo, excepto para los saltos (dos ciclos)
- Velocidad de operación desde DC hasta 20MHz
- 8k de memoria de programa
- 368 bytes de memoria RAM
- 256 bytes de EEPROM
- 35 instrucciones de 1 palabra

### **Periféricos:**

- Timer0: contador/temporizador de 8 bits con un prescaler programable de 8 bits
- Timer1: contador/temporizador de 16 bits con capacidad para reloj o cristal externo
- Timer2: contador/temporizador de 8 bits con registro de período de 8 bits, prescaler y postscaler
- Dos Módulos de Captura, Comparación y Modulación de Ancho de Pulso PWM:
  - Captura de 16 bits, resolución máxima 12,5 ns
  - Comparación de 16 bits, resolución máxima 200 ns
  - Resolución máxima PWM de 10 bits
- Puerto Serie Sincrónico (SSP) con SPI™ (Maestro) e I<sup>2</sup>C™ (Maestro/Esclavo)
- Receptor Transmisor Sincrónico Asincrónico Universal (USART/SCI), con detección de direcciones de 9 bits
- Puerto Paralelo Esclavo (PSP) de 8 bits con control externo



©Ing. Juan Ricardo Penagos Plazas

### **Características analógicas:**

- Convertidor A/D de 8 canales / 10 bits
- Reset por desvanecimiento (BOR)
- Comparador analógico:
  - Dos comparadores
  - Referencia de voltaje programable en el chip
  - Las salidas del comparador son accesibles externamente

### **Características especiales del microcontrolador:**

El fabricante en su hoja de especificaciones afirma que la memoria de programa se puede grabar y borrar unas 100.000 veces, la memoria de datos EEPROM 1.000.000 de veces y sus datos permanecen almacenados por más de 40 años. Otras características son la Programación en Serie en el Circuito (ICSP™) que requiere un total de 5 pines, WDT con oscilador independiente para mayor confiabilidad, protección de código programable, modo de bajo consumo (*Sleep*), múltiples opciones de oscilador externo, depuración en el circuito por medio de 2 pines, rango amplio de voltajes de operación (desde 2 V a 5,5 V) y bajo consumo.

### **Diagrama de pines:**

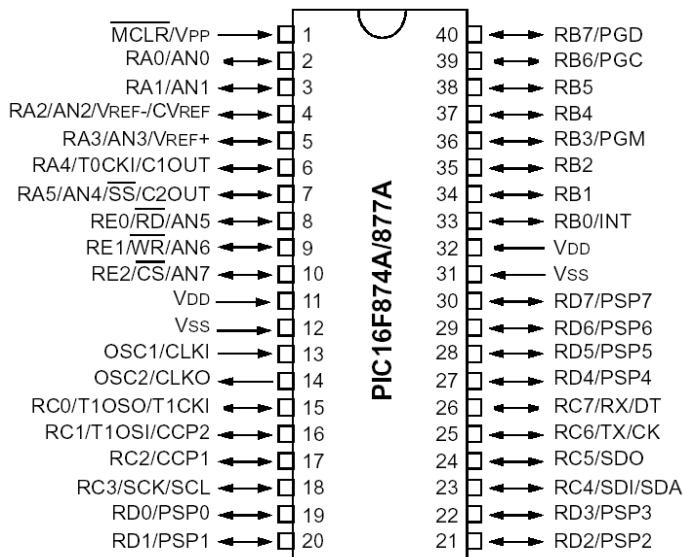
**40-Pin PDIP**

Figura 1.2.3 Distribución de terminales del PIC16F877A

**1.5 CONFIGURACIÓN INICIAL BÁSICA DE LOS PICS****PIC16F88**

Se emplea el oscilador interno (oscilador primario) integrado dentro del PIC16F88 para simplificar el diseño del hardware, así como disponer de dos pines más (RA6 y RA7) como E/S digital. Para seleccionar cualquiera de las 8 frecuencias disponibles se emplea el registro de control del oscilador OSCCON. Al seleccionar una frecuencia se debe esperar a que el bit IOFS de este registro sea igual a 1, indicando de esta forma que el oscilador se ha estabilizado. La frecuencia se selecciona por medio de los bits IRCF.

El temporizador de encendido PWRT se habilita para mantener al PIC en reset hasta que la fuente de alimentación se estabilice. En la configuración básica no se habilitan ni el encendido de doble velocidad, ni el oscilador de seguridad (palabra CONFIG2).

◆En caso de habilitar el reset por desvanecimiento (*Brown-out Reset BOR*) se debe conectar un capacitor de desacoplo de 100 nF (0.1 uF) lo más cerca posible de los pines de alimentación del PIC (VDD-VSS), para evitar que se produzca un reset indeseado cuando cualquiera de las salidas del microcontrolador cambia de estado.

La configuración inicial básica del PIC16F88 es la siguiente:

**CONFIG1 (Palabra de configuración 1):**

- Programación en bajo voltaje deshabilitada (LVP=0).
- Reset maestro deshabilitado (el pin RA5 es E/S digital, MCLRE=0).
- Temporizador de encendido habilitado (#PWRTEN=0).
- Temporizador de vigilancia deshabilitado (WDTEN=0).
- Oscilador interno con RA6 y RA7 como E/S digital (FOSC<2:0>=100).

**CONFIG2 (Palabra de configuración 2):**

- Encendido de doble velocidad deshabilitado (IESO=0).
- Oscilador de seguridad deshabilitado (FCMEN=0).

**OSCCON (Registro de control del oscilador):**

- Selección de frecuencia del oscilador interno de acuerdo a los bits IRCF<2:0>.
- Bit de indicación de estabilidad de la frecuencia del oscilador interno (IOFS=1: oscilador estable; IOFS=0: oscilador no estabilizado).
- Bits de selección de la fuente de reloj del sistema (SCS<1:0>=00: oscilador definido por los bits FOSC<2:0> de la palabra CONFIG1).

**PIC16F628A**

Se emplea el oscilador interno de 4MHz (modo INTOSC) integrado dentro del PIC16F628A para simplificar el diseño del hardware, así como disponer de dos pines más (RA6 y RA7) como E/S digital. Para seleccionar una

de las 2 frecuencias disponibles (4MHz por defecto, o 48kHz) se emplea el bit OSCF del registro de control de consumo PCON.

El temporizador de encendido PWRT se habilita para mantener al PIC en reset hasta que la fuente de alimentación se estabilice.

◆En caso de habilitar el reset por desvanecimiento (*Brown-out Reset BOR*) se debe conectar un capacitor de desacoplo de 100 nF (0.1 uF) lo más cerca posible de los pines de alimentación del PIC ( $V_{DD}$ - $V_{SS}$ ), para evitar que se produzca un reset indeseado cuando cualquiera de las salidas del microcontrolador cambia de estado.

La configuración inicial básica del PIC16F628A es la siguiente:

#### **CONFIG (Palabra de configuración):**

- Programación en bajo voltaje deshabilitada (LVP=0).
- Reset maestro deshabilitado (el pin RA5 es E/S digital, MCLRE=0).
- Temporizador de encendido habilitado (#PWRTE=0).
- Temporizador de vigilancia deshabilitado (WDTE=0).
- Oscilador interno con RA6 y RA7 como E/S digital (FOSC<2:0>=100).

## **PIC16F877A**

Se emplea el oscilador externo de 4MHz (modo HS). El temporizador de encendido PWRT se habilita para mantener al PIC en reset hasta que la fuente de alimentación se estabilice.

◆En caso de habilitar el reset por desvanecimiento (*Brown-out Reset BOR*) se debe conectar un capacitor de desacoplo de 100 nF (0.1 uF) lo más cerca posible de los pines de alimentación del PIC ( $V_{DD}$ - $V_{SS}$ ), para evitar que se produzca un reset indeseado cuando cualquiera de las salidas del microcontrolador cambia de estado.

La configuración inicial básica del PIC16F877A es la siguiente:

#### **CONFIG (Palabra de configuración):**

- Programación en bajo voltaje deshabilitada (LVP=0).
- Temporizador de encendido habilitado (#PWRTEN=0).
- Temporizador de vigilancia deshabilitado (WDTEN=0).
- Oscilador externo (HS).

### **1.5.1 Configuración inicial básica en mikroC™**

La configuración inicial es muy sencilla y consiste fundamentalmente en crear un nuevo proyecto a través del comando *Project>New Project* (figura 1.3), seleccionar el dispositivo (figura 1.4), la frecuencia de operación (figura 1.5) de mikroC™ (que debe coincidir con la frecuencia real de operación del PIC: 1MHz para el PIC16F88 y 4MHz para el PIC16F628A y el PIC16F877A), crear una nueva carpeta y definir un nombre de proyecto (figura 1.6) donde se almacenarán todos los archivos relacionados con el programa a desarrollar (código fuente en lenguaje C, código fuente en lenguaje ensamblador, código de máquina a grabarse en el PIC, entre otros), añadir archivos disponibles que hayan sido creados previamente (figura 1.7), en este caso no se ha añadido ninguno ya que el proyecto consta de un solo archivo de código fuente, y por último finalizar (figura 1.8).



Figura 1.3 Ventana de Bienvenida de Nuevo Proyecto

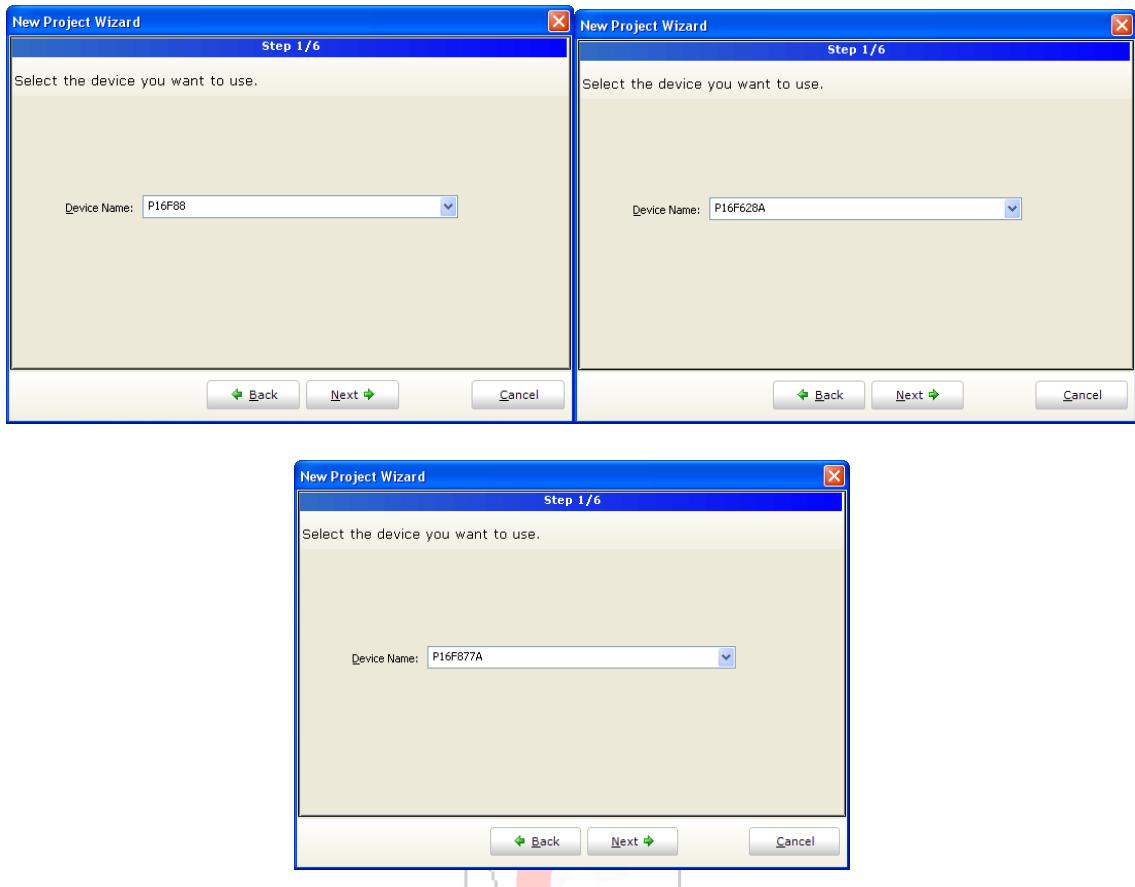


Figura 1.4 Ventana de selección del dispositivo

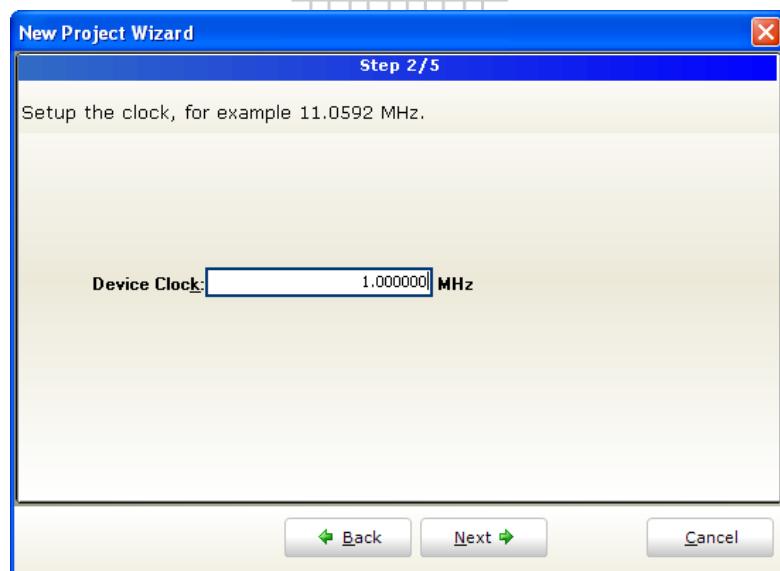


Figura 1.5 Ventana de selección de la frecuencia del dispositivo

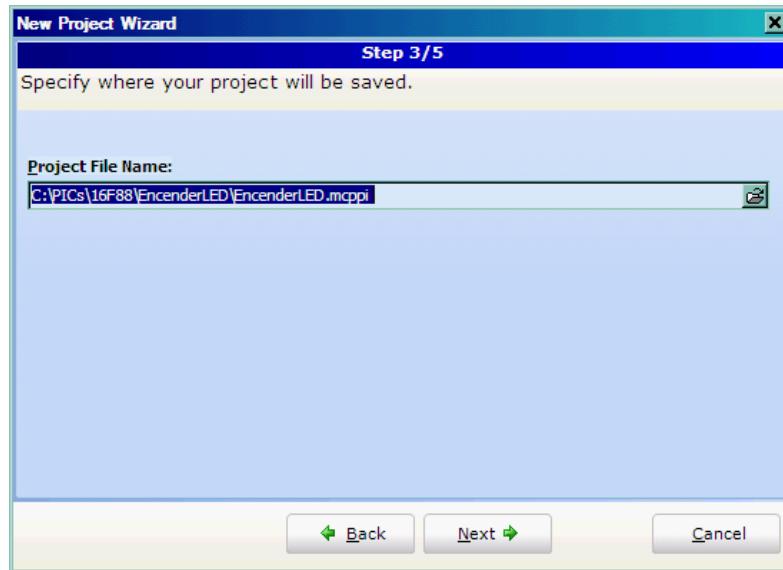


Figura 1.6 Ventana de ubicación de los archivos del proyecto

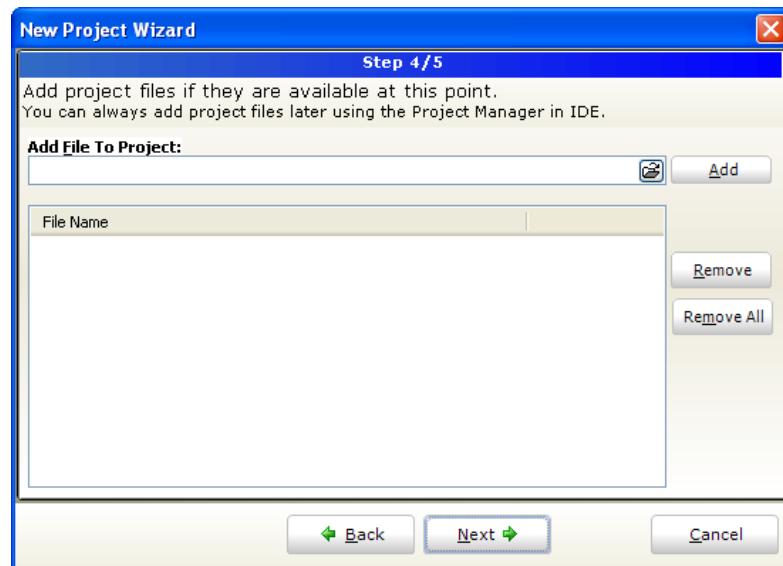


Figura 1.7 Ventana para adición de archivos al proyecto



Figura 1.8 Ventana de finalización de creación de un nuevo proyecto

A continuación establecer los bits de configuración en las palabras CONFIG1 y CONFIG2 (PIC16F88) ó CONFIG (PIC16F628A y PIC16F877A) a través del comando *Project > Edit Project* (figuras 1.9.1, 1.9.2 y 1.9.3).



Figura 1.9.1 Ventana de bits de configuración del PIC16F88

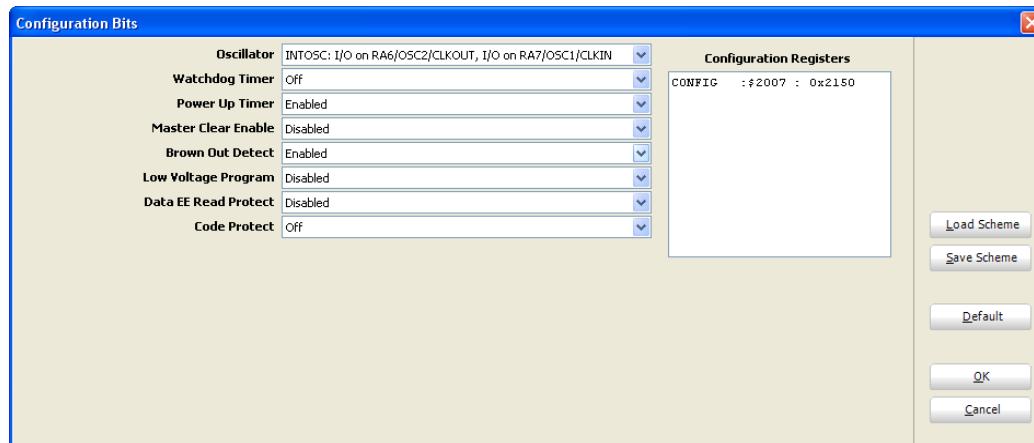


Figura 1.9.2 Ventana de bits de configuración del PIC16F628A

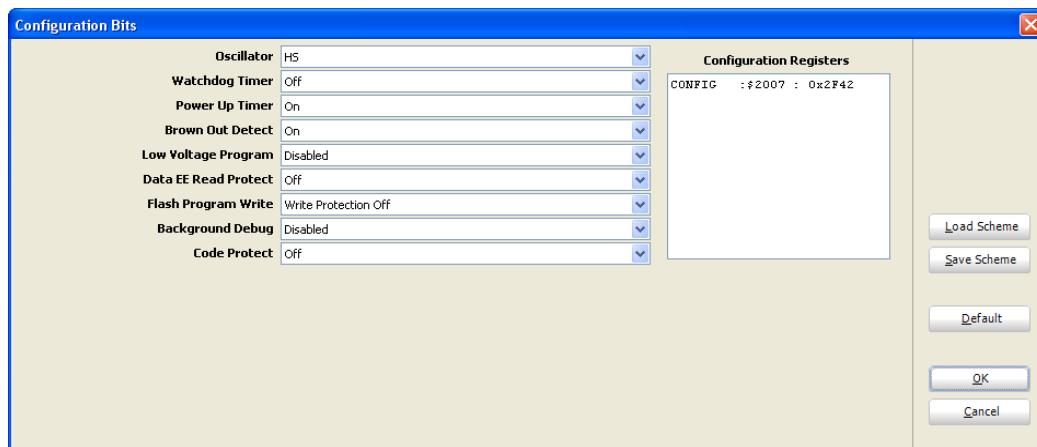


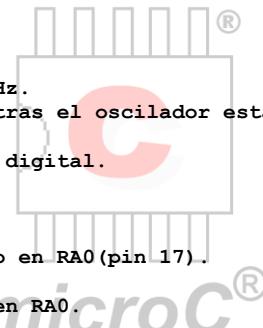
Figura 1.9.3 Ventana de bits de configuración del PIC16F877A

Por último escribir el código del programa (las primeras instrucciones permiten seleccionar la frecuencia del bloque oscilador interno del PIC16F88). Para ilustrar este proceso se ha escrito el sencillo programa **EncenderLED.c** para los PICs 16F88, 16F628A y 16F877A, descrito a continuación, que produce el parpadeo de un LED conectado en el pin RA0 con intervalos de 500 ms (nótese la simplicidad del procedimiento si se compara con la misma tarea en lenguaje ensamblador).

```
 //EncenderLED.c
//PIC16F88
//Configuración inicial básica.
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0x00; //Puerto A como salida.
while (1)
{
    RA0_bit=1; //Encender LED conectado en RA0(pin 17).
    Delay_ms(500);
    RA0_bit=0; //Apagar LED conectado en RA0.
    Delay_ms(500);
}
}

 //EncenderLED.c
//PIC16F628A
//Configuración inicial básica.
void main(){
PORTA=0x00; //Inicialización.
CMCON=0x07; //Pines RA<3:0> como E/S digital.
TRISA=0x00; //Puerto A como salida.
while (1)
{
    RA0_bit=1; //Encender LED conectado en RA0(pin 17).
    Delay_ms(500);
    RA0_bit=0; //Apagar LED conectado en RA0.
    Delay_ms(500);
}
}

 //EncenderLED.c
//PIC16F877A
//Configuración inicial básica.
void main(){
PORTA=0x00; //Inicialización.
ADCON1=0x06; //Pines RA<5:0> como E/S digital.
TRISA=0x00; //Puerto A como salida.
while (1)
{
    RA0_bit=1; //Encender LED conectado en RA0(pin 2).
```



microC®

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

```

Delay_ms(500);
RA0_bit=0; //Apagar LED conectado en RA0.
Delay_ms(500);
}
}

```

Si se desea probar este ejemplo en el mundo real, se debe compilar por medio del comando *Project>Build* (figura 1.10). El compilador generará nuevos archivos en la carpeta de proyecto creada anteriormente: EncenderLED.asm, EncenderLED.lst, EncenderLED.mcl, EncenderLED.hex y otros más. Este último se empleará para programar el microcontrolador. Puede ver un video de todo el proceso de escritura y compilación del ejemplo anterior para el PIC16F628A en nuestra página web [www.programarpicenc.com](http://www.programarpicenc.com) en la sección **Vídeos**.

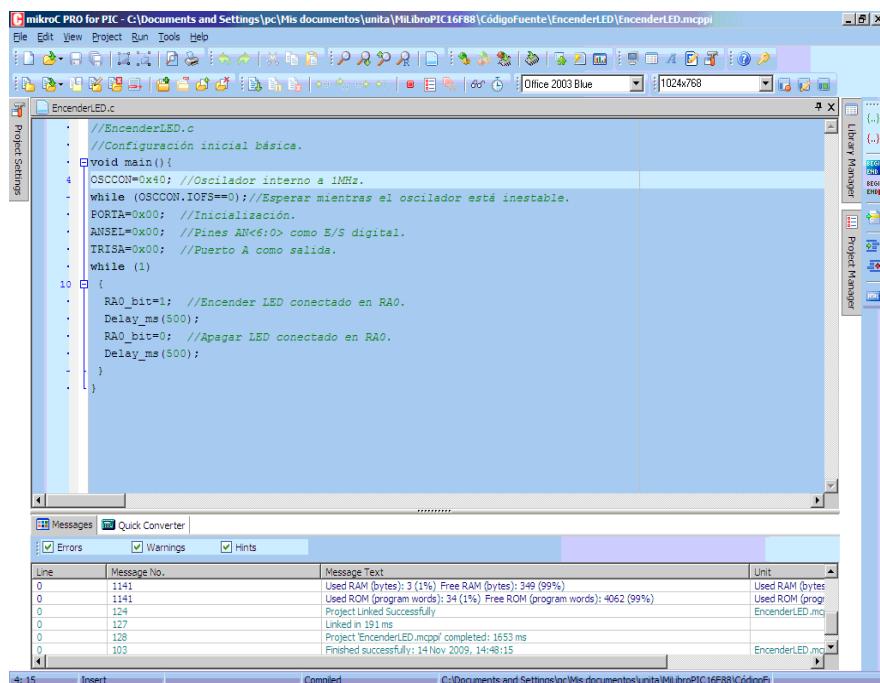


Figura 1.10 Pantalla global de mikroC™. En la parte superior se muestra el código fuente en lenguaje C y en la parte inferior los resultados de la compilación (Messages).

## 1.6 mikroC™

mikroC™ es una poderosa herramienta de desarrollo para microcontroladores PIC. Está diseñado para proporcionar al usuario la solución más simple en el desarrollo de aplicaciones para sistemas embebidos, sin comprometer el rendimiento o el control.

Los PICs y el lenguaje C se acoplan perfectamente: los PICs son los microcontroladores de 8 bits más populares actualmente, empleados en una amplia variedad de aplicaciones, y el lenguaje C, muy apreciado por su eficiencia, es la elección natural para el desarrollo de sistemas embebidos. mikroC™ se caracteriza por su IDE avanzado, compilador conforme a las normas ANSI, amplio conjunto de librerías para hardware, documentación muy fácil de comprender, y muchos ejemplos listos para su ejecución.

### 1.6.1 Características de mikroC™

- Le permite al usuario desarrollar rápidamente aplicaciones complejas:
- Escribir el código fuente en lenguaje C usando el Editor de Código incorporado (Asistentes de Parámetros y Código, *Syntax Highlighting*, Auto Corrección, Plantillas de Código, y más).
- Usar las librerías incluidas para agilizar drásticamente el proceso de desarrollo: adquisición de datos, memoria, *displays*, conversiones, comunicaciones, etc.
- Monitorear la estructura del programa, variables, y funciones en el Explorador de Código.
- Generar lenguaje ensamblador y archivos HEX estándar, compatibles con todos los programadores.

- Inspeccionar el flujo del programa y depurarlo con el Simulador (en *Software*).
- Obtener reportes detallados: mapas RAM y ROM, estadísticas de código, archivos de listado, árbol de llamadas, y más.

### 1.6.2 Los menús de mikroC™

Son los siguientes: *File*, *Edit*, *View*, *Project*, *Run*, *Tools*, *Help*. Contienen una gran variedad de comandos, los cuales se emplean para la manipulación de los archivos fuente (menú *File*), la edición del texto (menú *Edit*), el manejo de distintas ventanas (menú *View*), la manipulación de proyectos (menú *Project*), la simulación (menú *Run*), el manejo de las herramientas incorporadas (menú *Tools*) y la ayuda (menú *Help*). Cuando la situación lo requiera se explicará el uso de algunos de estos comandos. Si el lector desea puede encontrar una descripción completa seleccionando el comando *Help>Help*.

### 1.6.3 Proceso de simulación básica en mikroC™

El Simulador Software es un componente de mikroC™. Está diseñado para simular operaciones de los PICs, y asistir a los usuarios en el proceso de depuración de los programas en lenguaje C escritos para estos dispositivos. Una vez que se ha finalizado la escritura del programa, seleccionar la opción *Build Type: Release* en la ventana *Project Settings* (figura 1.11).

Después de la compilación exitosa del proyecto, se puede ejecutar el simulador seleccionando el comando *Run>Start Debugger*, esta acción abre las ventanas *Watch Values* y *Watch Clock* (figura 1.12). La línea que se va a ejecutar está resaltada en azul (por defecto).

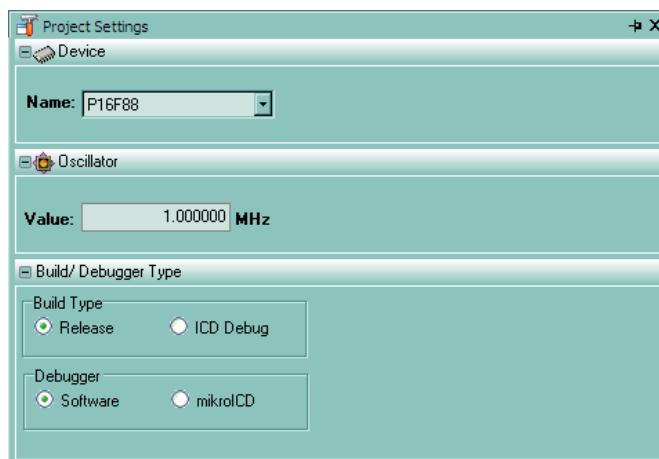


Figura 1.11 Ventana Project Settings

◆ El simulador imita el flujo del programa y la ejecución de las instrucciones, pero no puede emular completamente el comportamiento del PIC, es decir, no actualiza los temporizadores, las banderas de interrupción, etc.

La ventana *Watch Values* le permite monitorear los ítems del programa mientras realiza la simulación. En esta ventana se muestran las variables y registros del PIC, con sus direcciones y contenidos. Las variables se van actualizando a medida que se avanza en la simulación. Los ítems que han cambiado recientemente tienen color rojo.

La ventana *Watch Clock* muestra el conteo actual del número de ciclos (y tiempo) transcurridos desde la última acción del simulador. *Stopwatch* mide el tiempo parcial de ejecución (y número de ciclos) de forma acumulada desde cualquier línea y se lo puede restablecer en cualquier momento. *Delta* representa el tiempo (y número de ciclos) que tarda en la ejecución de una línea.

◆ El usuario puede cambiar el reloj en esta ventana, lo cual recalcula los tiempos para la nueva frecuencia. Al cambiar el reloj no se ve afectada la configuración actual del proyecto, sólo proporciona una simulación.

Comando		Descripción
Start Debugger	F9	Inicia el simulador
Stop Debugger	Ctrl+F2	Detiene el simulador
Pause Debugger	F6	Pausa el simulador
Step Into	F7	Ejecución paso a paso (individual)
Step Over	F8	Ejecución paso a paso (global)
Toggle Breakpoint	F5	Activa/Desactiva los <i>breakpoints</i>
Show/Hide Breakpoints	Shift+F4	Muestra/Oculta los <i>breakpoints</i>
Clear Breakpoints	Shift+Ctrl+F5	Borrar los <i>breakpoints</i>
Watch Window	Shift+F5	Muestra/Oculta la ventana <i>Watch Values</i>
View Stopwatch		Muestra/Oculta la ventana <i>Watch Clock</i>
Disassembly mode	Ctrl+D	Intercambia entre C y ensamblador

Tabla 1.4 Comandos del menú Run

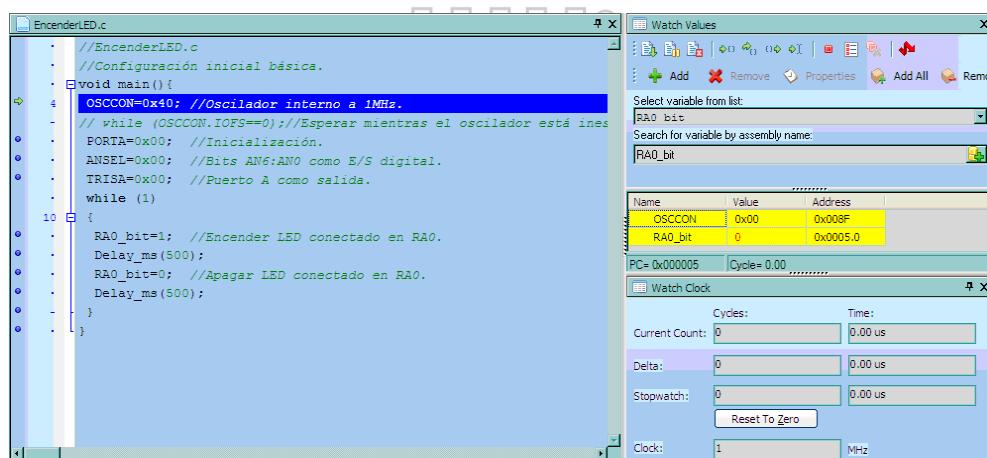


Figura 1.12 Ventanas al inicio de la simulación

Los comandos del menú *Run* (tabla 1.4) se emplean durante la simulación. Se explicará el proceso de simulación básica del ejemplo **EncenderLED.c**.

◆(Sólo para el PIC16F88) La instrucción `while (OSCCON.IOFS==0);` debe eliminarse temporalmente durante la simulación, debido a que el bit IOFS se activa únicamente en el microcontrolador (hardware) y no en el simulador, para ello lo más práctico es colocarla como un comentario y compilar nuevamente el programa. Debe notarse que normalmente los simuladores software no realizan la simulación del oscilador real empleado, sino únicamente toman en cuenta la base de tiempo para la ejecución de las instrucciones.

En este ejemplo se monitorean las variables OSCCON y RA0\_bit, para lo cual se seleccionan una a una de la lista *Select variable from list* y se las añade por medio del ícono *Add*. Se puede cambiar el formato numérico de presentación haciendo clic en la casilla *Value* correspondiente. En la simulación básica se emplea comúnmente la ejecución paso a paso a través del comando *Run>Step Into* (F7), el cual ejecuta una por una las instrucciones resaltadas. Presionar F7 para cargar el registro OSCCON con el valor 40h que fija la frecuencia del bloque oscilador interno en 1MHz, esta acción tarda 16 us (*Delta*) para llevarse a cabo. Presionar tres veces más F7 para cargar el registro PORTA (8us), ANSEL (8us) y TRISA (4us) (figura 1.13).

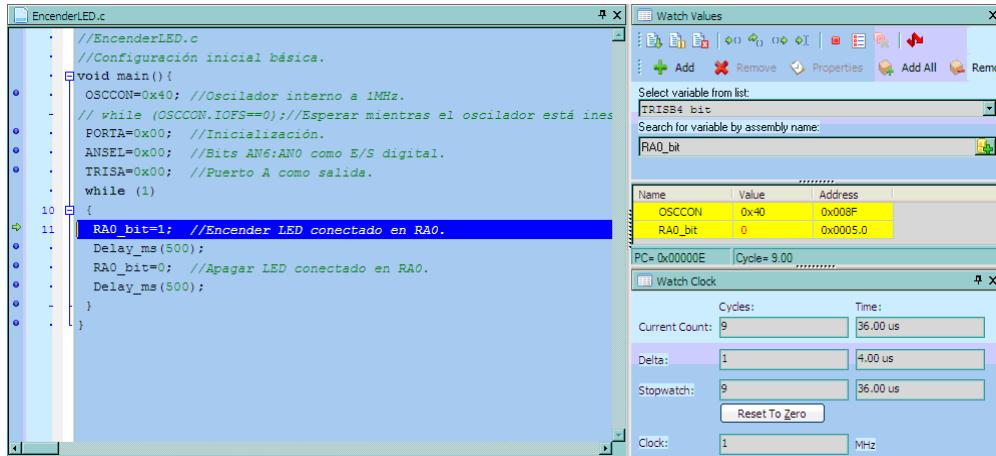


Figura 1.13 Ventanas luego de la ejecución de las cuatro primeras instrucciones

Encender el LED toma 8 us, y así permanece durante 500 ms (figura 1.14). Apagarlo toma 4 us y permanece en ese estado por 500 ms. El ciclo encendido/apagado empieza a repetirse indefinidamente de aquí en adelante.

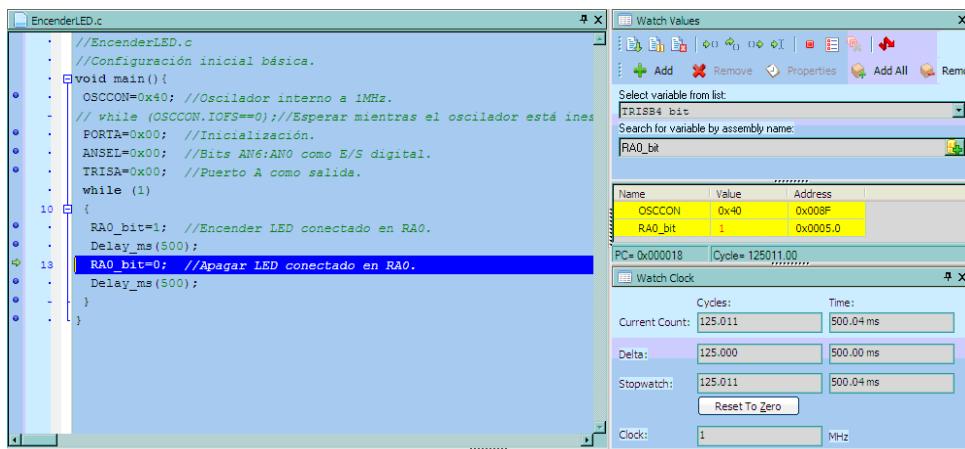


Figura 1.14 Ventanas luego de la ejecución de las seis primeras instrucciones

Para salir de la simulación se selecciona el comando *Run>Stop Debugger*.

## 1.7 CONCEPTOS BÁSICOS DE LENGUAJE C

A continuación se presentan los conceptos fundamentales más importantes, necesarios para emprender exitosamente el aprendizaje de programación de microcontroladores PIC.

### 1.7.1 Estructura básica de un programa en lenguaje C (sin funciones)

Todos los programas (código fuente) en lenguaje C tienen una estructura básica, a partir de la cual se desarrolla cualquier aplicación del usuario:

```
//Nombre_de_programa.c
//Descripción del programa.
//Autor: Ing. Penagos R.
//Declaración de variables
...
//Función principal
void main( ){
    //Instrucciones del programa.
}
```

## 1.7.2 Los siete elementos básicos de la programación

El propósito de la mayoría de los programas es resolver un problema. Los programas resuelven los problemas por medio de la manipulación de información o datos. Normalmente los programas se caracterizan por permitir el ingreso de información, tener uno o varios lugares de almacenamiento de dicha información, contar con las instrucciones para manipular estos datos y obtener algún resultado del programa que sea útil para el usuario. También, las instrucciones se pueden organizar de tal forma que algunas de ellas se ejecuten sólo cuando una condición específica (o conjunto de condiciones) sea verdadera, otras instrucciones se repitan un cierto número de veces y otras pueden ser agrupadas en bloques que se ejecutan en diferentes partes de un programa.

Lo anterior constituye una breve descripción de los siete elementos básicos de la programación: entrada de datos, tipos de datos, operaciones, salida, ejecución condicional, lazos y funciones. Una vez que se dominan estos elementos se puede afirmar que se conocen los fundamentos de la programación, con lo cual ya es posible desarrollar una gran cantidad de aplicaciones de diversa índole.

### Instrucciones básicas de programación en lenguaje C

#### 1.- Instrucción de asignación (=).-

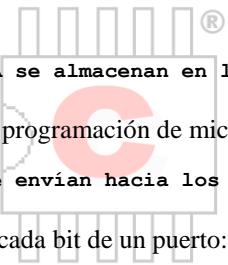
Permite asignar a una variable un valor constante, el contenido de otra variable o el resultado de una expresión matemática. La asignación va de derecha a izquierda. Por ejemplo,

```
suma=0; //El valor 0 se almacena en la variable suma.  
x0=x1; //El contenido de la variable x1 se almacena en la variable x0.  
dx=(b-a)/n; //El resultado de la expresión matemática se almacena en la variable dx.
```

#### 2.- Instrucción de entrada de datos (variable=PORTx).-

Permite el ingreso de uno o más datos a través de los pines del microcontrolador y almacenarlos en una o más variables. Por ejemplo,

```
variable=PORTA; //Los bits del puerto A se almacenan en la variable.
```



El siguiente es un caso especial utilizado en la programación de microcontroladores:

```
PORTB=PORTA; //Los bits del puerto A se envían hacia los pines del puerto B.
```

También se puede leer el estado individual de cada bit de un puerto:

```
variable=RB3_bit; //Lee el estado del pin RB3 y lo guarda en la variable.
```

#### 3.- Instrucción de salida de datos (PORTx=dato).-

Permite el envío de datos, el contenido de una variable o el resultado de una expresión matemática hacia los pines de un puerto. Por ejemplo,

```
PORTA=0x00; //Todos los pines del puerto A se ponen en 0.  
PORTB=variable; //Los bits de la variable son enviados hacia los pines del puerto B.  
PORTB=PORTA+65; //El valor del puerto A más 65 se envía hacia el puerto B.
```

Como caso especial, se pueden enviar bits individuales a cada uno de los pines de un puerto:

```
RB0_bit=0; //El pin RB0 se pone en 0.
```

#### 4.- Instrucción de decisión (if...else).-

Permite la ejecución de las **instrucciones1** si la condición es verdadera, de lo contrario se ejecutan las **instrucciones2**. Las llaves {} no son necesarias cuando hay una sola instrucción.

```
if (condición){  
  instrucciones1;  
}  
else{  
  instrucciones2;  
}
```

#### Ejemplo 1:

Si el contenido de la variable **codigo** es igual al contenido de la variable **clave**, se ejecutan las primeras cuatro instrucciones; de lo contrario se ejecutan únicamente los dos últimas instrucciones.

```
if (codigo==clave) {
    intentos=0;
    RA7_bit=1;
    Delay_1sec( );
    RA7_bit=0;
}
else{
    intentos++;
    Delay_ms(200);
}
```

**Ejemplo 2:**

Instrucción de decisión sin **else**. Esta es una variante muy utilizada cuando se desea condicionar la ejecución de un grupo de instrucciones.

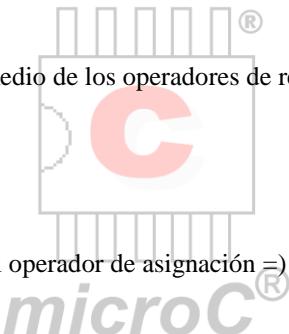
Las dos instrucciones se ejecutarán únicamente si la variable **contador** es igual a 2, de lo contrario la ejecución continúa a partir de la línea //Aquí.

```
if (contador==2){
    RB6_bit=~RB6_bit;
    contador=0;
}
//Aquí.
```

**Ejemplo 3:**

Similar al caso anterior pero con una sola instrucción. Si la variable **horas** es igual a 24 se reinicia esta variable con un valor de cero.

```
if (horas==24) horas=0;
```



**Nota 1:** Las condiciones se obtienen por medio de los operadores de relación y los operadores lógicos.

**Nota 2:** Operadores de relación:

>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
==	Igual a (nótese la diferencia con el operador de asignación =)
!=	No es igual a

**Nota 3:** Operadores lógicos:

&&	Y
	O

[www.programarpicenc.com](http://www.programarpicenc.com)  
©Ing. Juan Ricardo Penagos Plazas

**5.- Instrucción de ciclo controlado por una variable (for).-**

Permite ejecutar un grupo de instrucciones de manera repetitiva, una determinada cantidad de veces.

```
for (número de veces){
    instrucciones;
}
```

**Ejemplo 1:**

La variable **i** tiene un valor inicial de 7 (**i=7**) y un valor final de 1 (**i>=1**). Esta variable va disminuyendo de 1 en 1 (**i--**). Por lo tanto las dos instrucciones se ejecutarán en 7 ocasiones. La primera vez cuando **i=7**, la segunda cuando **i=6**, la tercera cuando **i=5** y así sucesivamente hasta la séptima vez cuando **i=1**. Luego la ejecución continúa a partir de la línea //Aquí.

```
for (i=7; i>=1; i--){
    PORTB=PORTB<<1;
    Delay_ms(500);
}
//Aquí.
```

**Ejemplo 2:**

El valor inicial de la variable es 1 y su valor final es 3. La variable **i** se va incrementando de 1 en 1 (**i++**). Por lo tanto la instrucción se ejecuta tres veces, lo que da como resultado un retardo de 3 segundos. Luego la ejecución continúa a partir de la línea //Aquí.

```
for (i=1; i<=3; i++)
  Delay_1sec();
//Aquí.
```

## 6.- Instrucción iterativa condicional (while).-

Permite ejecutar un grupo de instrucciones de manera repetitiva, mientras una condición sea verdadera. Primero se revisa la condición para determinar su valor de verdad (verdadero o falso) y luego se ejecutan las instrucciones.

```
while (condición) {
  instrucciones;
}
```

### Ejemplo 1:

La ejecución del programa permanece indefinidamente en esta línea mientras el bit IOFS del registro OSCCON sea igual a cero. Como caso particular no se ejecuta ninguna instrucción (la cual debería estar antes del punto y coma).

```
while (OSCCON.IOFS==0) ;
```

### Ejemplo 2:

Ejemplo de un lazo infinito. En lenguaje C, cualquier valor numérico diferente a cero se considera VERDADERO, y un valor numérico igual a cero se considera FALSO.

Al valor numérico del puerto A se le suma el valor 65, el resultado se envía hacia los pines del puerto B. Este proceso se repite continua e indefinidamente, debido a que la condición siempre es verdadera (1).

```
while (1)
  PORTB=PORTA+65;
```

### Ejemplo 3:

Las cuatro instrucciones encerradas por {} se ejecutarán indefinidamente mientras el valor del bit RB0 sea igual a 0.

```
while (RB0_bit==0) {
  RB1_bit=1;
  Delay_ms(500);
  RB1_bit=0;
  Delay_ms(200);
}
```



## 7.- Instrucción hacer-mientras (do...while).-

Permite ejecutar un grupo de instrucciones de manera repetitiva, mientras una condición sea verdadera. Es similar a la instrucción **while**, con la diferencia de que primero se ejecutan las instrucciones y luego se revisa la condición.

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

### Ejemplo 1:

La variable **kp** tiene un valor inicial de cero. La instrucción **kp=Keypad\_Key\_Click()**; se ejecuta y luego se revisa la condición (**!kp**). Mientras **kp** sea igual a cero (FALSO) la condición será VERDADERA (**!kp**), debido al operador de negación **!** que cambia el valor de verdad a su estado contrario. Como resultado se tendrá un lazo infinito mientras la variable **kp** siga en cero. Cuando la variable **kp** cambie de valor como consecuencia de la pulsación de una tecla, la condición será FALSA y la ejecución continuará en la línea //Aquí.

```
kp=0;
do
  kp=Keypad_Key_Click();
while (!kp);
//Aquí.
```

### Ejemplo 2:

Las cuatro instrucciones dentro de {} se ejecutarán indefinidamente mientras la variable **tecla** sea diferente a 1.

```

do{
    ingreso();
    raiz();
    pn_1();
    seg_grado();
}
while (tecla != 1);

```

**Nota:** A diferencia de la instrucción `while`, en la instrucción `do...while` las instrucciones se ejecutan por lo menos una vez.

### 8.- Instrucción de selección múltiple (switch).-

Permite la ejecución de un grupo de instrucciones de varios grupos posibles, dependiendo del valor de una variable.

```

switch (variable){
    case 1: instrucciones1;
              break;

    case 2: instrucciones2;
              break;

    case 3: instrucciones3;
              break;

    ...
    default: instrucciones;
}

```

Si la `variable` es igual a 1 se ejecutan únicamente las `instrucciones1`, si es igual a 2 se ejecutan únicamente las `instrucciones2` y así sucesivamente. Si la `variable` no es igual a ninguno de los casos (`case`) se ejecutan las instrucciones por defecto (`default`).

#### Ejemplo 1:

Esta es una función numérica que da como resultado un número hexadecimal dependiendo del valor que tenga la variable `digit`. Si `digit` es igual a 0 la función devuelve (`return`) el valor 0x3F. Si `digit` es igual a 1, la función devuelve el valor 0x06, y así sucesivamente. Este ejemplo es una variante de la instrucción `switch`, donde no aparece el elemento `default`.

```

char Bin2_7seg(char digit){
    switch (digit){
        case 0: return 0x3F;
        case 1: return 0x06;
        case 2: return 0x5B;
        case 3: return 0x4F;
        case 4: return 0x66;
    }
}

```

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

### 1.7.3 Funciones

Una función es una agrupación de instrucciones para formar una nueva instrucción creada por el programador (usuario). Empleando funciones, la solución total de un determinado problema se divide en varios subproblemas, cada uno de los cuales es resuelto por medio de una función particular, aplicando de esta manera la conocida máxima “Divide y vencerás”.

Las funciones constituyen una de las características fundamentales del lenguaje C, pues todo programa bien escrito hace uso de ellas. Para poder utilizar una función se deben cumplir los dos siguientes pasos:

**1.-Declaración de la función.-** Consiste en indicar el tipo, nombre y parámetros de la función:

```
tipo nombre ( parámetro1, parámetro2, ...);
```

**2.-Definición de la función.-** Consiste en indicar las instrucciones que forman parte de dicha función:

```

tipo nombre ( parámetro1, parámetro2, ...){
    instrucciones;
}

```

## Estructura básica de un programa en lenguaje C (con funciones).-

Todos los programas (código fuente) en lenguaje C tienen una estructura básica, a partir de la cual se desarrolla cualquier aplicación del usuario:

```
//Nombre_de_programa.c
//Descripción del programa.
//Autor: Ing. Penagos R.
//*****Declaración de funciones (prototipos)*****
...
//*****Fin de declaración de funciones*****
//=====Declaración de variables=====
...
//=====Fin de declaración de variables=====
//*****Función principal*****
void main( ){
    //Instrucciones del programa.
    ...
}
//*****Fin de función principal*****
//=====Definición de funciones=====

función1{
    instrucciones1;
}

función2{
    instrucciones2;
}
//=====Fin de definición de funciones=====
```

**Nota 1:** Los tipos de funciones más empleadas son numéricas (**char**) y nulas (**void**). Las primeras retornan (**return**) o devuelven como resultado un número, mientras que las segundas simplemente ejecutan un grupo de instrucciones.

### Ejemplo 1:

Este ejemplo se puede encontrar en el capítulo II, Puertos Digitales, del libro. La función es numérica (**char**), su nombre es **Bin2\_7seg** y tiene un parámetro **digit** de tipo **char**.

La función se utiliza como si fuera una instrucción cualquiera, tomando en cuenta el tipo de función y su(s) parámetro(s). En este ejemplo se tiene **PORTB=Bin2\_7seg (PORTA)**. Esto significa que la variable **PORTA** remplaza a la variable **digit**. Por lo tanto si **PORTA** es igual a 0, la función devuelve el valor 0x3F que será enviado al puerto B. Si **PORTA** es igual a 1, la función devuelve 0x06 que será enviado al puerto B, y así sucesivamente.

```
//7seg1.c
//Se utiliza la función Bin2_7seg que transforma un número binario a su
//equivalente en 7 segmentos.

char Bin2_7seg(char digit); //Prototipo de la función.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0);//Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1) PORTB=Bin2_7seg(PORTA);
}

char Bin2_7seg(char digit){ //Definición de la función.
switch (digit){
    case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
    case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
    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 0x67;
    }
}

```

**Ejemplo 2:**

Variante del ejemplo anterior, en el que se hace únicamente la definición de la función (sin declaración). Se debe hacer antes de la función principal, de lo contrario se producirán errores de compilación por tratar de usar una función desconocida.

```

//7seg1.c
//Se utiliza la función Bin2_7seg que transforma un número binario a su
//equivalente en 7 segmentos.

char Bin2_7seg(char digit){ //Definición de la función.
    switch (digit){
        case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
        case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
        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 0x67;
    }
}

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1) PORTB=Bin2_7seg(PORTA);
}

```

**1.7.4 Para tener en cuenta**

Los comentarios se inicián con la doble barra diagonal //.

Los signos de agrupación siempre deben estar en pareja, es decir si hay tres llaves de apertura {}, deben haber tres llaves de cierre correspondientes }}}. Lo mismo con los paréntesis () .

Los números hexadecimales se escriben comenzando siempre con 0x, por ejemplo 0x0A, 0x16, 0xFD, etc.

Los números binarios se escriben comenzando siempre con 0b, por ejemplo 0b001110, 0b11101111, etc.

Los números decimales se escriben de la forma común y corriente, por ejemplo 64, 126, 12.75, etc.

No se debe confundir el operador de asignación (=) con el operador de comparación (==) igual a.

El punto y coma (;) indica el final de una instrucción, por lo tanto hay que tener mucho cuidado para colocarlo en el lugar apropiado.

Las llaves {} no son necesarias en aquellos casos en los que únicamente se va a ejecutar una instrucción (ver los ejemplos a lo largo de este apartado).

Todo programa en lenguaje C debe tener una función principal (**main**), y su nombre no debe ser cambiado.

Los tipos de datos más utilizados se muestran en la tabla 1.5.

Tipo	Tamaño (en bytes)	Rango
<b>bit</b>	1 bit	0 ó 1
<b>char</b>	1	0...255
<b>signed char</b>	1	-128...127
<b>int</b>	2	-32768...32767
<b>unsigned</b>	2	0...65535
<b>long</b>	4	-2147483648...2147483647
<b>unsigned long</b>	4	0...4294967295
<b>float</b>	4	-1.5 * 10 <sup>45</sup> ... +3.4 * 10 <sup>38</sup>

Tabla 1.5 Tipos de datos más comunes en lenguaje C

El tipo **float** es para números con punto decimal, mientras que los demás tipos son para números enteros.

La EJECUCIÓN DE UNA INSTRUCCIÓN consiste en la realización de las operaciones especificadas en esa instrucción. De la ejecución se encarga la CPU (unidad central de proceso) del microcontrolador.

## CAPÍTULO II: PUERTOS DIGITALES

Las aplicaciones que tienen los puertos digitales son muy variadas, debido a la amplia gama de sensores y actuadores de dos estados que existen en la actualidad para el diseño de sistemas electrónicos de control, por ello es imprescindible conocer los detalles de su configuración, alcances y limitaciones. En este capítulo se explica la programación en lenguaje C de los puertos de los microcontroladores 16F88, 16F628A y 16F877A operando en modo digital.

### 2.1 PRINCIPALES CARACTERÍSTICAS

Los pines E/S (entrada/salida) de propósito general pueden considerarse como los periféricos más simples. Permiten que el PIC monitoree y controle otros dispositivos. Para añadir flexibilidad y funcionalidad al dispositivo, algunos de los pines de los puertos pueden cumplir funciones de diferentes periféricos. En general, cuando se habilita un periférico, ese pin no puede ser utilizado como un pin E/S de propósito general. El sentido de los datos (entrada o salida) se controla por el registro TRISx. El registro PORTx es el receptor (latch) de los datos de salida. Cuando se lee un puerto, el PIC lee los niveles presentes en los pines (no en el latch). Esto significa que debe ponerse mucha atención a los comandos leer-modificar-escribir (*read-modify-write RMW*) aplicados a los puertos y al cambiar el sentido de circulación en un pin de entrada a salida. Todas las operaciones de escritura (BSF, BCF) son del tipo RMW. Por lo tanto, la escritura en un puerto implica que los pines son leídos, este valor es modificado, y luego escrito en el latch de datos. Este tipo de instrucciones se debe emplear con precaución en puertos en los que se han definido entradas y salidas. Por ejemplo, una operación BSF en el bit5 del puerto B ocasionará que los ocho bits de este puerto sean leídos. Luego se efectúa la operación BSF en el bit5 y los ocho bits se escriben en el latch de salida. Si otro pin (por ejemplo el pin0 correspondiente al bit0) está definido como entrada en este momento, la señal de entrada del pin será leída y reenviada al latch de este pin, sobrescribiendo su contenido previo. Mientras el pin permanece como entrada, no hay problema. Sin embargo, si el pin0 se cambia para que opere como salida, el contenido del latch puede ser desconocido.

El siguiente ejemplo ilustra el efecto de dos instrucciones RMW de forma secuencial en un puerto. El propósito es configurar los pines RB7 y RB6 como salidas con un estado inicial igual a cero, a partir de las condiciones indicadas:

```
;Configuración inicial:  
;PORTB<7:4> Entradas  
;PORTB<3:0> Salidas  
;PORTB<7:6>=11. Tienen pull ups externos.  
;  
;          Latch      Pines  
BCF PORTB, 7 ;    01pp pppp    11pp pppp  
BCF PORTB, 6 ;    10pp pppp    11pp pppp  
BSF STATUS,RPO ;  
BCF TRISB, 7 ;    10pp pppp    11pp pppp  
BCF TRISB, 6 ;    10pp pppp    10pp pppp
```

Después del 1ero y 2do BCF se esperaría que el latch tuviera el valor 00pp ppp, sin embargo el 2do BCF hace que ingrese al latch el valor 1 del pin7. El 3ero y 4to BCF programan los pines 7 y 6 como salidas. Con el 3er BCF el valor del latch (1) es enviado al pin7. Con el 4to BCF el valor del latch (0) es enviado al pin6. El resultado es que en la salida se tiene el valor 10.

La forma correcta es la siguiente:

```
;Configuración inicial:  
;PORTB<7:4> Entradas  
;PORTB<3:0> Salidas  
;PORTB<7:6>=11. Tienen pull ups externos.  
;  
;          Latch      Pines  
BCF PORTB, 7 ;    01pp pppp    11pp pppp  
BSF STATUS,RPO ;  
BCF TRISB, 7 ;    01pp pppp    01pp pppp  
BCF STATUS,RPO ;  
BCF PORTB, 6 ;    00pp pppp    01pp pppp  
BSF STATUS,RPO ;  
BCF TRISB, 6 ;    00pp pppp    00pp pppp
```

Un pin configurado como salida y que se encuentre en alto o bajo no debe ser polarizado externamente al mismo tiempo con el propósito de cambiar su nivel lógico. Las altas corrientes de salida que resultan pueden dañar el chip. Se recomienda que al inicializar un puerto, el registro PORTx se inicialice primero, y luego el registro TRISx, ya que los valores del latch no están definidos al encendido.

## 2.2 PUERTO A

### **PIC16F88**

El puerto A es un puerto bidireccional de 8 bits que puede ser programado a través de 3 registros: PORTA (datos de entrada/salida), TRISA (sentido de circulación de los datos) y ANSEL (selección de modo analógico/digital). Al programar un bit de TRISA con un valor de 1 se consigue que el pin correspondiente del puerto A trabaje como entrada (es decir, coloca el driver de salida en alta impedancia). Al programar un bit de TRISA con 0 se logra que el pin correspondiente del puerto A opere como salida (es decir, coloca el contenido del latch de salida en el pin seleccionado). La lectura del registro PORTA, lee el estado de los pines, mientras que una escritura en él, escribe en el latch del puerto.

El fabricante recomienda la siguiente secuencia de inicialización de este puerto: escribir 0 en el registro PORTA, escribir 0 en el registro ANSEL para configurar los pines AN<6:0> como E/S digital, escribir en el registro TRISA los bits de sentido de circulación (1=entrada, 0=salida). En la hoja de especificaciones se indica que el pin RA5 sólo puede actuar como entrada, por lo que este detalle debe estar presente a la hora de diseñar una aplicación.

◆El registro TRISA (y también TRISB) tiene todos sus bits en 1 después de cualquier reset y no cambia al despertar por WDT o interrupción, por lo tanto el puerto A (y también el puerto B) están configurados como entradas inicialmente.

En la tabla 2.1.1 se indican las tecnologías de entrada y salida de cada pin de este puerto. Para tener a disposición los pines RA7:RA6 es necesario configurar el microcontrolador para que utilice el oscilador interno, y para disponer del pin RA5 se lo debe configurar como E/S digital (deshabilitar la función *Master Clear Reset*).

Pin	Entrada	Salida
RA0	TTL	CMOS
RA1	TTL	CMOS
RA2	TTL	CMOS
RA3	TTL	CMOS
RA4	ST	CMOS
RA5	ST	No disponible
RA6	ST	CMOS
RA7	ST	CMOS

Tabla 2.1.1 Tecnologías E/S del puerto A del PIC16F88 en modo digital

### **PIC16F628A**

El puerto A es un puerto bidireccional de 8 bits que puede ser programado a través de 4 registros: PORTA (datos de entrada/salida), TRISA (sentido de circulación de los datos), CMCON (control del comparador) y VRCON (control de la referencia de voltaje). Al programar un bit de TRISA con un valor de 1 se consigue que el pin correspondiente del puerto A trabaje como entrada (es decir, coloca el driver de salida en alta impedancia). Al programar un bit de TRISA con 0 se logra que el pin correspondiente del puerto A opere como salida (es decir, coloca el contenido del latch de salida en el pin seleccionado). La lectura del registro PORTA lee el estado de los pines, mientras que una escritura en él, escribe en el latch del puerto.

El fabricante recomienda la siguiente secuencia de inicialización de este puerto: escribir 0 en el registro PORTA, escribir 0x07 en el registro CMCON para configurar los pines RA<3:0> como E/S digital, escribir en el registro TRISA los bits de sentido de circulación (1=entrada, 0=salida). En la hoja de especificaciones se indica que el pin RA5 sólo puede actuar como entrada, por lo que este detalle debe estar presente a la hora de diseñar una aplicación.

◆El registro TRISA (y también TRISB) tiene todos sus bits en 1 después de cualquier reset y no cambia al despertar por WDT o interrupción, por lo tanto el puerto A (y también el puerto B) están configurados como entradas inicialmente.

Pin	Entrada	Salida
RA0	ST	CMOS
RA1	ST	CMOS
RA2	ST	CMOS
RA3	ST	CMOS
RA4	ST	Drenaje abierto
RA5	ST	No disponible
RA6	ST	CMOS
RA7	ST	CMOS

En la tabla 2.1.2 se indican las tecnologías de entrada y salida de cada pin de este puerto. Para tener a disposición los pines RA7:RA6 es necesario configurar el microcontrolador para que utilice el oscilador interno, y para disponer del pin RA5 se lo debe configurar como E/S digital (deshabilitar la función *Master Clear Reset*).

Tabla 2.1.2 Tecnologías E/S del puerto A del PIC16F628A en modo digital

## 2.2.1 Características de las tecnologías de entrada/salida

El comparador *Schmitt Trigger* (ST) se caracteriza porque su salida estará siempre en uno de los dos niveles binarios posibles (1 o 0), sin importar el nivel de voltaje de entrada (tomando en cuenta los límites máximos), no existe una región de indeterminación como ocurre en TTL. Esta característica se emplea para conformar ondas irregulares y obtener ondas digitales puras. También tiene alta inmunidad al ruido, ya que se requiere una variación importante del voltaje de entrada para que se produzca una transición en la salida.

Los transistores MOS se comportan como interruptores controlados por voltaje, mientras que los transistores bipolares empleados en la tecnología TTL se comportan como interruptores controlados por corriente. Las familias lógicas MOS son especialmente susceptibles a daños por carga electrostática. Esto es consecuencia directa de la alta impedancia de entrada. Una pequeña carga electrostática que circule por estas altas impedancias puede dar origen a voltajes peligrosos. Los CMOS están protegidos contra daño por carga estática mediante la inclusión en sus entradas de diodos zéner de protección, diseñados para conducir y limitar la magnitud del voltaje de entrada a niveles muy inferiores a los necesarios para provocar daño. Si bien los zéner por lo general cumplen con su finalidad, algunas veces no comienzan a conducir con la rapidez necesaria para evitar que el circuito integrado sufra daños. Por consiguiente, sigue siendo buena idea observar las precauciones de manejo empleadas con los dispositivos sensibles a las descargas electrostáticas (ESD). La gran ventaja de los CMOS es que utilizan solamente una fracción de la potencia que se necesita para la serie TTL, adaptándose de una forma ideal a aplicaciones que utilizan la potencia de una batería o con soporte en una batería. El inconveniente de la familia CMOS es que normalmente es más lenta que la familia TTL.

## 2.3 PUERTO B

### PIC16F88

Este es un puerto bidireccional de 8 bits. El registro de sentido de datos es TRISB. Al programar un bit de TRISB con un valor de 1 se consigue que el pin correspondiente del puerto B trabaje como entrada (es decir, coloca el driver de salida en alta impedancia). Al programar un bit de TRISB con 0 se logra que el pin correspondiente del puerto B opere como salida (es decir, coloca el contenido del latch de salida en el pin seleccionado). Cada pin de este puerto tiene un *pull up* interno (resistencia conectada a V<sub>DD</sub>). El bit #RBPU del registro OPTION\_REG puede activar o desactivar esta función. Cuando el puerto se configura como salida las *pull ups* se desactivan automáticamente y también cada vez que se enciende el PIC (*Power On Reset POR*). Este puerto se puede programar por medio de 4 registros: PORTB, TRISB, OPTION\_REG (activar/desactivar las *pull ups*) y ANSEL. En la tabla 2.2.1 se indican las tecnologías de entrada y salida de cada pin de este puerto.

Pin	Entrada	Salida
RB0	TTL	TTL
RB1	TTL	CMOS
RB2	TTL	CMOS
RB3	TTL	TTL
RB4	TTL	CMOS
RB5	TTL	TTL
RB6	TTL	TTL
RB7	TTL	TTL

Tabla 2.2.1 Tecnologías E/S del puerto B del PIC16F88 en modo digital.

Cuando se habiliten funciones de periféricos de este puerto, se debe tener cuidado al definir los bits del registro TRISB. Algunos periféricos modifican estos bits para hacer que algunos pines operen como salida y otros como entrada. Esta modificación tiene vigencia mientras el periférico está habilitado, por lo tanto se debería evitar el uso de instrucciones RMW con el registro TRISB.

### PIC16F628A

Este es un puerto bidireccional de 8 bits. El registro de sentido de datos es TRISB. Al programar un bit de TRISB con un valor de 1 se consigue que el pin correspondiente del puerto B trabaje como entrada (es decir, coloca el driver de salida en alta impedancia). Al programar un bit de TRISB con 0 se logra que el pin correspondiente del puerto B opere como salida (es decir, coloca el contenido del latch de salida en el pin seleccionado). Cada pin de este puerto tiene un *pull up* interno (200 uA aproximadamente). El bit #RBPU del registro OPTION\_REG puede activar o desactivar esta función. Cuando el puerto se configura como salida las *pull ups* se desactivan automáticamente y también cada vez que se enciende el PIC (*Power On Reset POR*). Este puerto se puede programar por medio de 3 registros: PORTB, TRISB, OPTION\_REG (activar/desactivar las *pull ups*). En la tabla 2.2.2 se indican las tecnologías de entrada y salida de cada pin de este puerto.

Pin	Entrada	Salida
RB0	TTL	CMOS
RB1	TTL	CMOS
RB2	TTL	CMOS
RB3	TTL	CMOS
RB4	TTL	CMOS
RB5	TTL	CMOS
RB6	TTL	CMOS
RB7	TTL	CMOS

Tabla 2.2.2 Tecnologías E/S del puerto B del PIC16F628A en modo digital.

Cuando se habiliten funciones de periféricos de este puerto, se debe tener cuidado al definir los bits del registro TRISB. Algunos periféricos modifican estos bits para hacer que algunos pines operen como salida y otros como entrada. Esta modificación tiene vigencia mientras el periférico está habilitado.

## 2.4 PUERTOS A, B, C, D y E del PIC16F877A

El PIC16F877A posee cinco puertos con un total de 33 pines, lo que le brinda a este microcontrolador una gran ventaja con respecto a otros PICs que tienen un número de pines relativamente reducido. En la tabla 2.3 se pueden apreciar en detalle las características de estos puertos.

Puerto	No. de bits	Tecnología	Registros	Estado inicial en POR
<b>A</b>	<b>6</b>	TTL excepto RA4. ST para RA4 (como salida es drenaje abierto).	<b>PORTA</b> <b>TRISA</b> <b>ADCON1</b>	Entradas analógicas.
<b>B</b>	<b>8</b>	TTL	<b>PORTB</b> <b>TRISB</b>	Entradas digitales.
<b>C</b>	<b>8</b>	ST	<b>PORTC</b> <b>TRISC</b>	Entradas digitales.
<b>D</b>	<b>8</b>	ST	<b>PORTD</b> <b>TRISD</b>	Entradas digitales.
<b>E</b>	<b>3</b>	ST	<b>PORTE</b> <b>TRISE</b> <b>ADCON1</b>	Entradas analógicas.

Tabla 2.3 Puertos del PIC16F877A

La configuración de los puertos A y E se realiza de forma similar, programando el registro ADCON1 con un valor de 0x06 para que los pines RA<5:0> y RE<2:0> operen como E/S digital, debido a que inicialmente estos pines están configurados como entradas analógicas.

◆ Los registros TRISx tienen todos sus bits en 1 después de cualquier reset y no cambian al despertar por WDT o interrupción, por lo tanto los puertos A, B, C, D y E están configurados como entradas inicialmente.

## 2.5 VALORES MÁXIMOS ABSOLUTOS

En la hoja de especificaciones del fabricante se indican los valores máximos para las intensidades de corriente de los pines del microcontrolador (tablas 2.4.1 y 2.4.2), los cuales deben ser tomados en cuenta para los diseños de las aplicaciones prácticas. Las líneas de los puertos son capaces de entregar niveles TTL cuando el voltaje de alimentación aplicado en V<sub>DD</sub> es de 5 V.

Pines	Máxima corriente de entrada (mA)	Máxima corriente de salida (mA)
Cualquier pin E/S	25	25
Puerto A en total	100	100
Puerto B en total	100	100
Pin V <sub>DD</sub>	200 (16F88) 250 (16F628A)	No se aplica a este pin
Pin V <sub>SS</sub>	No se aplica a este pin	200 (16F88) 300 (16F628A)

Tabla 2.4.1 Corrientes máximas permitidas en los PICs 16F88 y 16F628A

Pines	Máxima corriente de entrada (mA)	Máxima corriente de salida (mA)
Cualquier pin E/S	25	25
Puertos A, B y E (combinados)	200	200
Puertos C y D (combinados)	200	200
Pin V <sub>DD</sub>	250	No se aplica a este pin
Pin V <sub>SS</sub>	No se aplica a este pin	300

Tabla 2.4.2 Corrientes máximas permitidas en el PIC16F877A

## 2.6 EJEMPLOS DE PROGRAMACIÓN

### PIC16F88 Y 16F628A

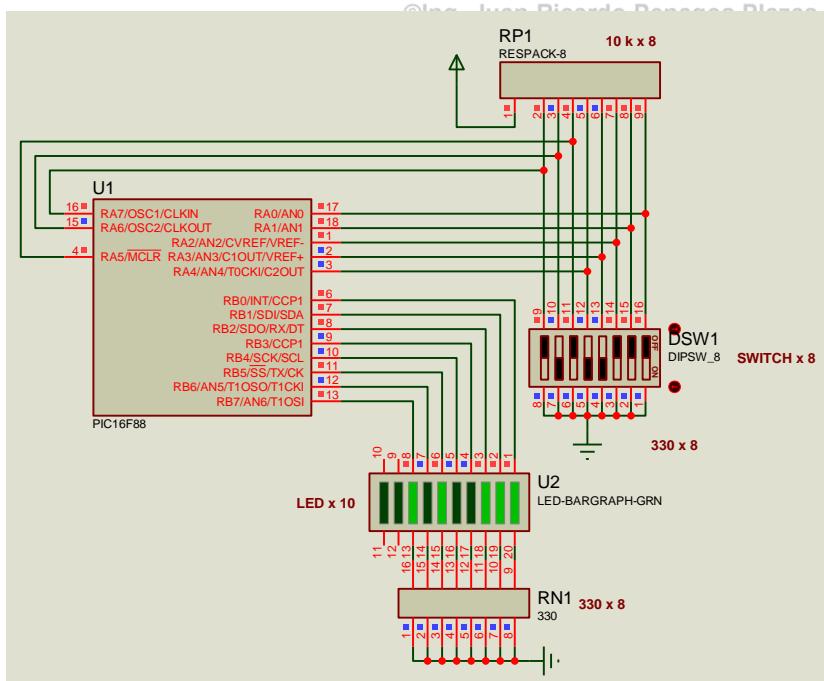
A continuación se muestra un conjunto de ejemplos de programación de los **puertos A (entrada) y B (salida)**. Estos ejemplos se han desarrollado para el PIC16F88 (figura 2.1.1), pero también se aplican al PIC16F628A con las siguientes advertencias:

1. Las conexiones eléctricas son las mismas para el PIC16F628A ya que es compatible pin a pin con el PIC16F88.
2. El código fuente es el mismo para el PIC16F628A tomando en cuenta las respectivas diferencias en la inicialización del oscilador y los puertos.
3. Se ha seleccionado una frecuencia de operación de 1MHz para el PIC16F88 (a menos que se indique lo contrario) y 4MHz para el PIC16F628A (en todos los casos).
4. **(Sólo para el PIC16F88)** Para la simulación, tanto en mikroC™ como en ISIS de PROTEUS, deben tomarse muy en cuenta las notas del numeral 1.6.3 Proceso de simulación básica.

### PIC16F877A

Este PIC tiene que ser polarizado como se indica en la figura 2.1.2 para obtener los resultados esperados. En la definición del problema debe tomarse en cuenta que **la entrada de datos se realiza por el puerto B** (se utilizan las pull-ups con lo que no es necesario el empleo de resistores externos) y **la salida de datos se realiza por el puerto C** (figura 2.1.3). Se emplea un oscilador de 4MHz en todos los casos. El código fuente se encuentra en la carpeta correspondiente que acompaña a este libro.

Para que no haya dudas al respecto, se ha escrito el primer ejemplo para el PIC16F88, el PIC16F628A y el 16F877A. Lo mismo se aplica al resto de ejemplos de este capítulo.



- En la entrada:
  - Un interruptor cerrado representa un 0 lógico.
  - Un interruptor abierto representa un 1 lógico.
- En la salida:
  - Un LED apagado representa un 0 lógico.
  - Un LED encendido representa un 1 lógico.
  - En los ejemplos pertinentes: se emplea un **display de 7 segmentos** de cátodo común (**k**) con punto decimal (**dp**).

Figura 2.1.1 Circuito de los problemas Puerto1.c a Puerto9.c (PIC16F88 y 16F628A)

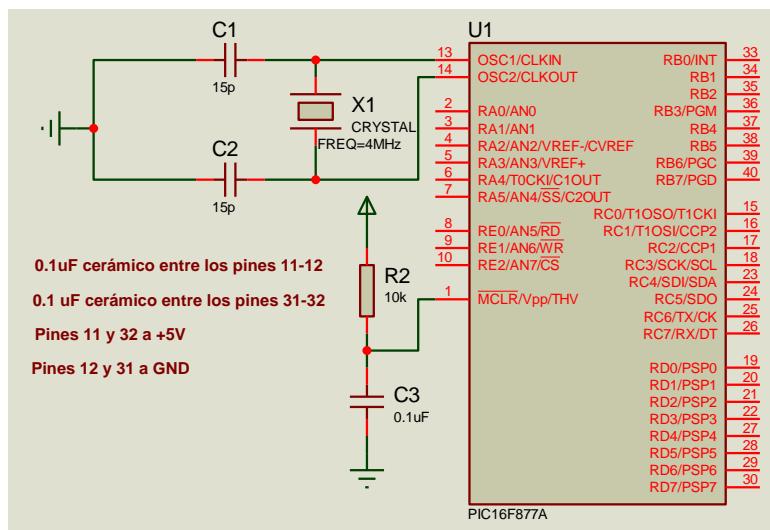


Figura 2.1.2 Polarización básica del PIC16F877A

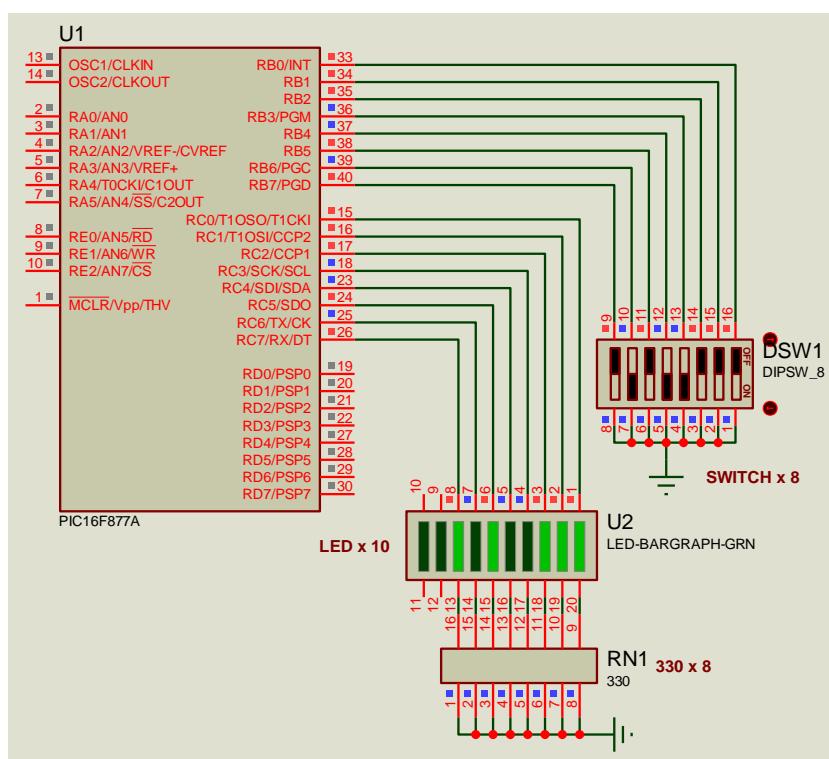


Figura 2.1.3 Circuito de los problemas Puerto1.c a Puerto9.c (PIC16F877A)

**Puertos1.c:** Por el puerto B se obtiene el dato de las ocho líneas del puerto A al que está conectado un arreglo de interruptores (*dip switch*). Por ejemplo, si por el puerto A se introduce 10100111 por el puerto B aparecerá 10100111.

```


//Puertos1.c
//PIC16F88
void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1) PORTB=PORTA;
}


//Puertos1.c
//PIC16F628A
void main() {
PORTA=0x00; //Inicialización.
PORTB=0x00;
CMCON=0x07; //Pines RA<3:0> como E/S digital.
}

```

```

TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1) PORTB=PORTA;
}


//Puertos1.c
//PIC16F877A
//Entrada: Puerto B.
//Salida: Puerto C.
//El puerto B está configurado inicialmente como entrada.
void main(){
PORTC=0x00; //Inicialización.
TRISC=0x00; //Puerto C como salida.
NOT_RBPU_bit=0; //Habilitar las pull-ups del puerto B.
while (1) PORTC=PORTB;
}

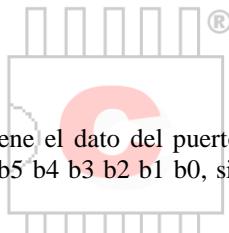
```

**Puertos2.c:** Por el puerto B se obtiene el dato de las ocho líneas del puerto A, al que está conectado un *dip switch*, sumándole el valor de una constante, por ejemplo 65 decimal. Es decir  $(\text{PORTB}) = (\text{PORTA}) + 65$ .

```


//Puertos2.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1)
    PORTB=PORTA+65;
}

```



**Puertos3.c:** Por el puerto B se obtiene el dato del puerto A, pero los bits pares de la salida se fijan siempre a 1. El orden de los bits será b7 b6 b5 b4 b3 b2 b1 b0, siendo los bits pares: b6 b4 b2 b0. Observe el ejemplo de la tabla 2.4.

	b7	b6	b5	b4	b3	b2	b1	b0
Puerto A	0	1	0	0	1	0	1	1
Puerto B	0	1	0	1	1	1	1	1

www.programarpicenc.com

Tabla 2.4 Ejemplo de entrada/salida para el programa Puertos3.c

```


//Puertos3.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1)
    PORTB=PORTA|0b01010101; //Operación OR (con bits).
}

```

**Puertos4.c:** Por el puerto B se obtiene el dato del puerto A invertidos los unos y los ceros.

```


//Puertos4.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1)

```

```

PORTB=~PORTA; //Operación NOT (con bits).
}

```

**Puertos5.c:** Por el puerto B se obtiene el dato del puerto A desplazando un bit hacia la izquierda, por la derecha entrará un 1.

```

 //Puertos5.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1)
    PORTB=(PORTA<<1)+1; //Operación SHIFT LEFT.
}

```

**Puertos6.c:** Por el puerto B se obtiene el dato del puerto A invirtiendo los bits pares. Los impares se dejan como en la entrada.

```

 //Puertos6.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1)
    PORTB=PORTA^0b01010101; //Operación XOR.
}

```



**Puertos7.c:** El puerto B, que actúa como salida es controlado por el bit 0 del puerto A (RA0), que actúa como entrada, de manera que:

- Si RA0 = 1, se encienden todos los LEDs de salida.
- Si RA0 = 0, se encienden únicamente los LEDs del nibble alto.

```

 //Puertos7.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1){
    if (RA0_bit==1) PORTB=0xFF; //Encender todos los LEDs.
    if (RA0_bit==0) PORTB=0xF0; //Encender los LEDs del nibble alto.
}
}

```

**Puertos8.c:** Compara el dato introducido por el puerto A con un número. Se tienen las dos siguientes posibilidades:

- Si PORTA  $\geq$  “Número” se encienden todos los LEDs.
- Si PORTA < “Número” se activan los LEDs pares de la salida.

```

 //Puertos8.c
char Numero=34; //No colocar tilde (`).

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1){

```

```

if (PORTA >= Numero) PORTB=0xFF; //Encender todos los LEDs.
if (PORTA < Numero) PORTB=0b01010101; //Encender los LEDs pares.
}
}

```

**Puertos9.c:** Compara el dato del puerto A con un número. Se tienen tres posibilidades:

- Si PORTA = Número, se encienden todos los LEDs de salida.
- Si PORTA > Número, se activan los LEDs pares de salida.
- Si PORTA < Número, se encienden los LEDs del *nibble* alto.

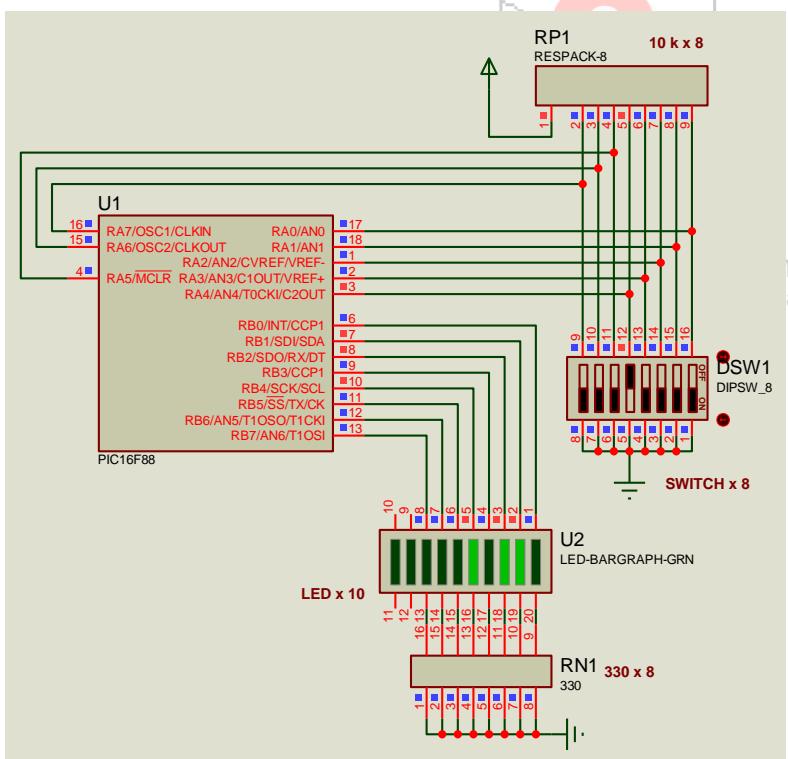
**//Puertos9.c**

```

//Puertos9.c
char Numero=145; //No colocar tilde (`).
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.
TRISB=0x00; //Puerto B como salida.
while (1){
  if (PORTA == Numero) PORTB=0xFF; //Encender todos los LEDs.
  if (PORTA > Numero) PORTB=0b01010101; //Encender los LEDs pares.
  if (PORTA < Numero) PORTB=0xF0; //Encender los LEDs del nibble alto.
}
}

```

El Manejador de Librerías (*Library Manager*) habilita la manipulación de forma sencilla de las librerías empleadas en un proyecto. En la ventana se observa una lista completa de todas las librerías almacenadas en la carpeta *Uses*. La librería que se necesita se añade a un proyecto seleccionándola por medio del cajetín  junto al nombre. Únicamente las librerías seleccionadas serán enlazadas. En el ejemplo siguiente se emplea la función *Dec2Bcd* de la librería *Conversions* para transformar un número decimal a su equivalente BCD.



**BCD\_01.c:** Un número de 8 bits ( $\leq 99$ ) ingresado por el puerto A es convertido a BCD y el resultado se presenta en los 8 LEDs conectados al puerto B. *Conversions* .

Figura 2.2 Circuito del problema BCD\_01.c

**//BCD\_01.c**

```

//BCD_01.c
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISA=0xFF; //Puerto A como entrada.

```

```

TRISB=0x00; //Puerto B como salida.
while (1)
    PORTB=Dec2Bcd(PORTA); //Añadir la librería Conversions.
}

```

**TVerdad1.c:** Implementar la siguiente tabla de verdad.

C (RA2)	B (RA1)	A (RA0)	Y (RB1)	X (RB0)
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

Tabla 2.5 Tabla de verdad para el programa TVerdad1.c

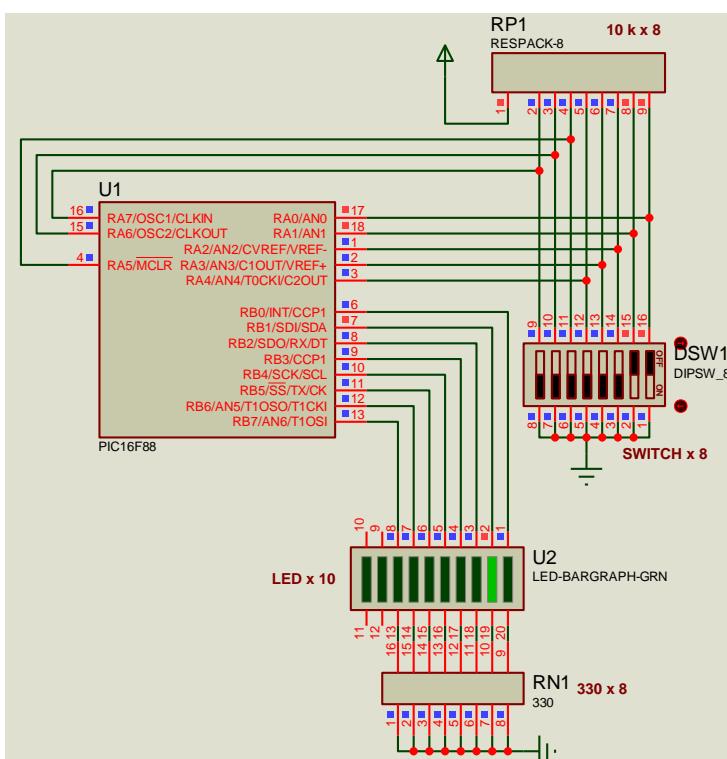


Figura 2.3 Circuito del problema TVerdad1.c

```

 //TVerdad1.c
void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1{
    if (PORTA==0b000) PORTB=0b00;
    if (PORTA==0b001) PORTB=0b11;
    if (PORTA==0b010) PORTB=0b01;
    if (PORTA==0b011) PORTB=0b10;
    if (PORTA==0b100) PORTB=0b00;
    if (PORTA==0b101) PORTB=0b01;
    if (PORTA==0b110) PORTB=0b10;
    if (PORTA==0b111) PORTB=0b01;
}
}

```

El siguiente es un ejemplo de cómo declarar, definir y emplear una función.

**7seg1.c:** En un *display* de 7 segmentos conectado al puerto B se visualiza la cantidad leída por el puerto A. Por ejemplo, si por el puerto A se ingresa 0b0001001 (9) en el *display* se observa el número 9. **k=cátodo común, dp=punto decimal.**

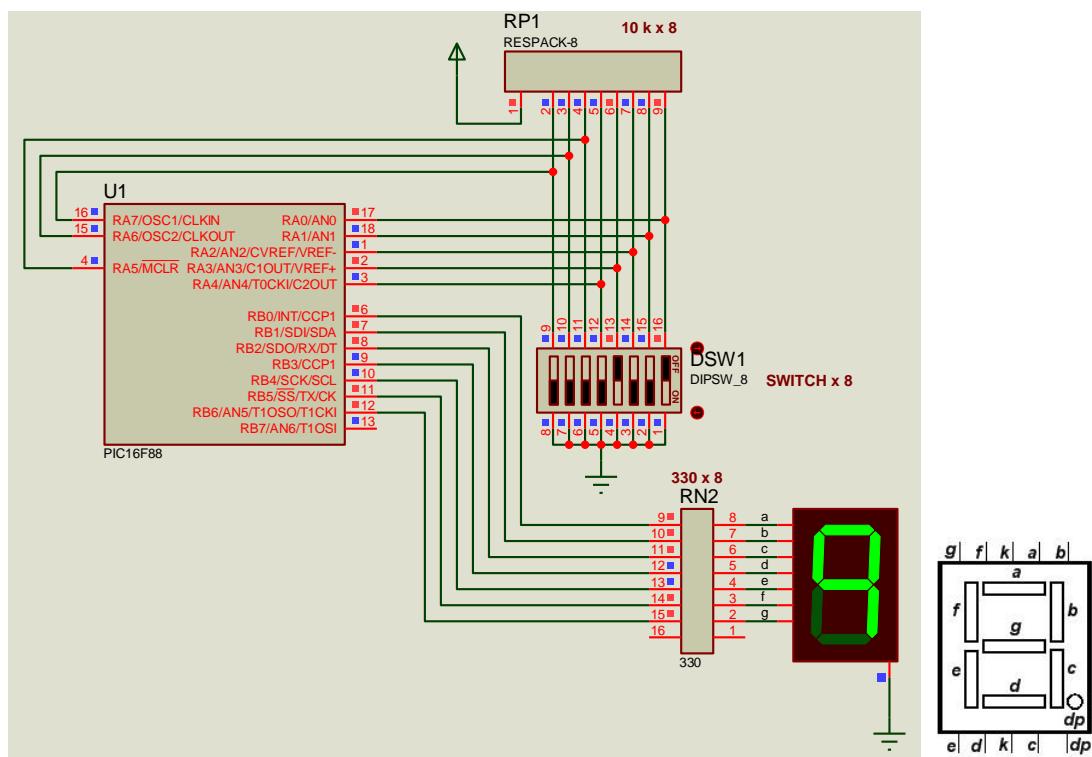


Figura 2.4.1 Circuito del problema 7seg1.c (PIC16F88 y 16F628A)

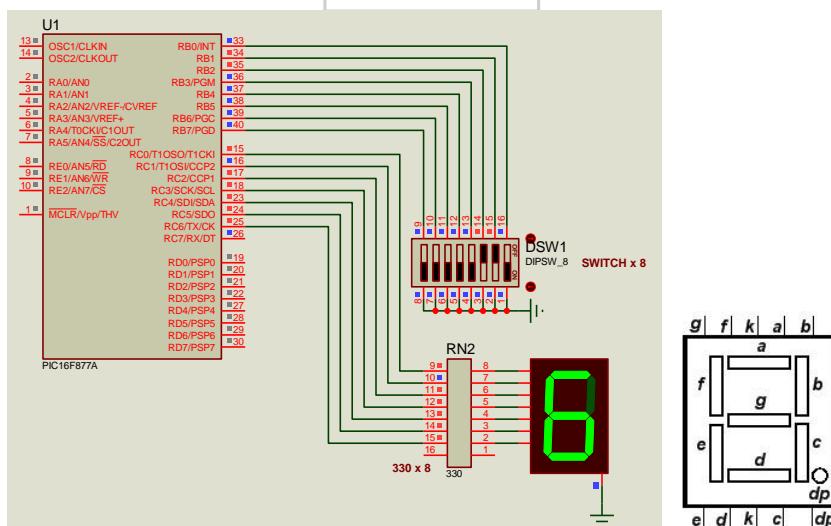


Figura 2.4.2 Circuito del problema 7seg1.c (PIC16F877A)

```


//7seg1.c
//Se utiliza la función Bin2_7seg que transforma un número binario a su
//equivalente en 7 segmentos.
char Bin2_7seg(char digit); //Prototipo de la función.

void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.


```

```

TRISB=0x00; //Puerto B como salida.
while (1) PORTB=Bin2_7seg(PORTA);
}

char Bin2_7seg(char digit){ //Definición de la función.
    switch (digit){
        case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
        case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
        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 0x67;
    }
}

```

Los siguientes ejemplos muestran el uso de algunas de las funciones incorporadas que permiten realizar, de manera muy cómoda, retardos precisos. Se pueden comprobar los tiempos en la simulación, observando la ventana *Watch Clock*. Recuerde que un ciclo de instrucción equivale a cuatro ciclos de reloj; es decir, si la

frecuencia de reloj es de 1 MHz (PIC16F88) el ciclo de instrucción tiene una duración de  $\frac{1}{1.000.000} \times 4 = 4\mu s$ .

También se puede contabilizar el tiempo multiplicando el número de ciclos de instrucción por 4 us. Por ejemplo, 75.000 ciclos x 4 us = 300 ms.

Para una frecuencia de reloj de 4MHz (PIC16F628A y 16F877A) el ciclo de instrucción tiene una duración de 1 us, por lo tanto el tiempo se puede calcular multiplicando el número de ciclos de instrucción por 1 us.

**Retardo1.c:** Un LED conectado en RB0 se enciende durante 400 ms y se apaga durante 300 ms cíclicamente.

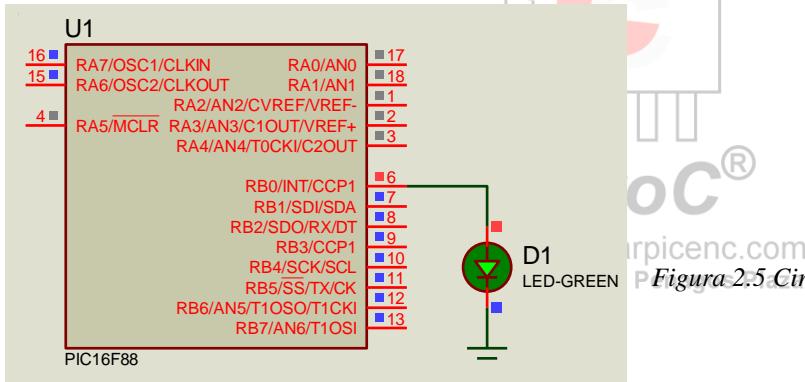


Figura 2.5 Circuito del problema Retardo1.c

```

 //Retardo1.c
//Retardo1.c
void main(){
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00; //Inicialización.
    ANSEL=0x00; //Pines AN<6:0> como E/S digital.
    TRISB=0x00; //Puerto B como salida.
    while (1){
        RB0_bit=1;
        Delay_ms(400);
        RB0_bit=0;
        Delay_ms(300);
    }
}

```

**Retardo2.c:** Los diodos pares conectados al puerto B se encienden durante 0,5 s y los impares permanecen apagados. Después se invierten durante el mismo tiempo y se repite el ciclo indefinidamente.

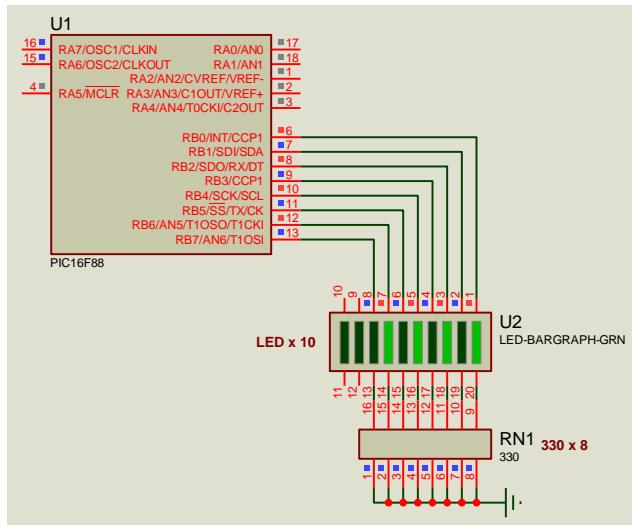


Figura 2.6.1 Circuito del problema Retardo2.c  
(PIC16F88 y 16F628A)

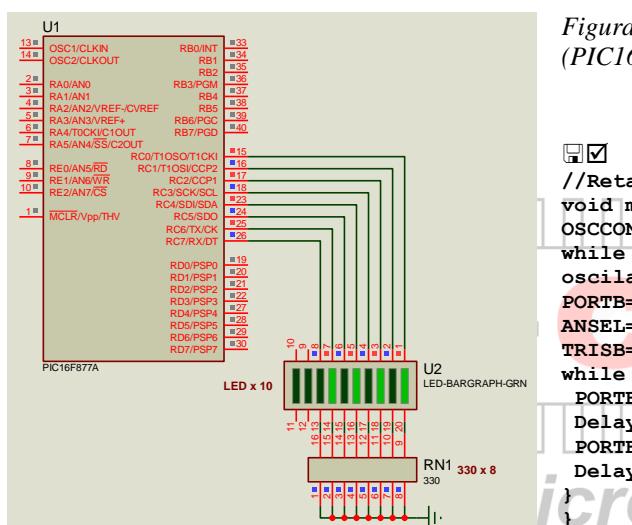
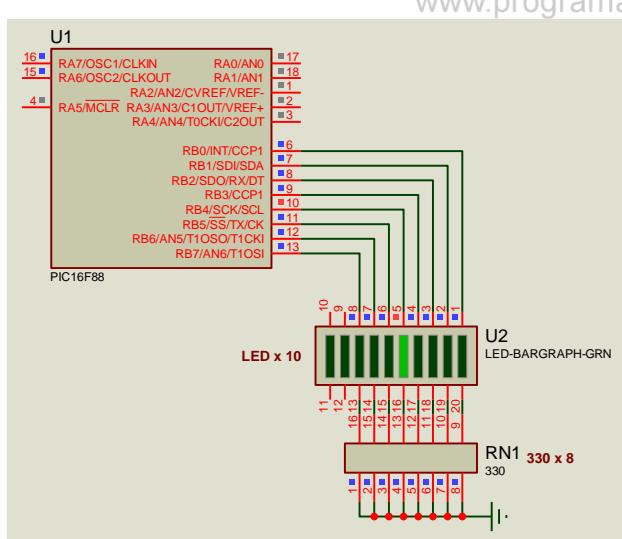


Figura 2.6.2 Circuito del problema Retardo2.c  
(PIC16F877A)

```

 //Retardo2.c
void main() {
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el
    oscilador está inestable.
    PORTB=0x00; //Inicialización.
    ANSEL=0x00; //Pines AN<6:0> como E/S digital.
    TRISB=0x00; //Puerto B como salida.
    while (1) {
        PORTB=0b01010101;
        Delay_ms(500);
        PORTB=0b10101010;
        Delay_ms(500);
    }
}

```



**Retardo3.c:** Por la barra de LEDs conectada al puerto B un LED encendido rota a la izquierda durante 0,3 s en cada posición. Cuando llega al final comienza a rotar a la derecha durante 0,5 s en cada posición. El ciclo se repite indefinidamente.

```

 //Retardo3.c
char i;
void main() {
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00; //Inicialización.
}

```

Figura 2.7 Circuito del problema Retardo3.c

```

ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.

PORTB=0x01; //Encendido de RB0 por primera vez.
Delay_ms(300);
while (1){
  for (i=7;i>=1;i--) {
    PORTB=PORTB<<1;
    Delay_ms(300);
  }

  for (i=7;i>=1;i--) {
    PORTB=PORTB>>1;
    Delay_ms(500);
  }
}
}

```

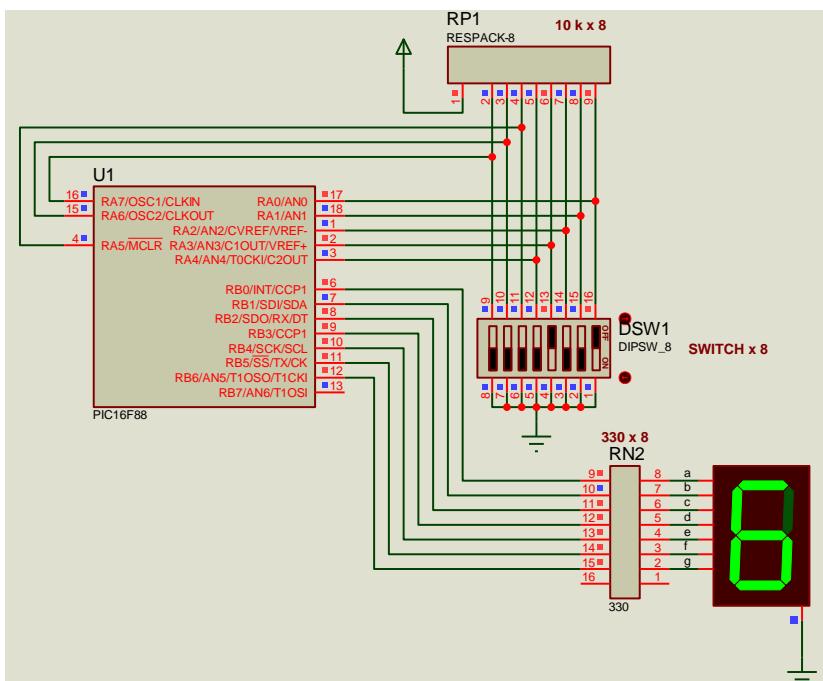
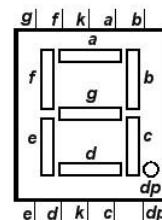


Figura 2.8 Circuito del problema Retardo4.c

**Retardo4.c:** El *display* de 7 segmentos visualiza un contador descendente desde la cantidad leída por el puerto A hasta cero y vuelve a repetir. Cada dígito se visualizará durante un segundo. Por ejemplo, si por el puerto A se introduce 0b00001001 (9) en el *display* se visualizarán los números: 9, 8, 7, 6, 5, ..., 0. Si la entrada es mayor que 9 o es 0, se enciende únicamente el segmento horizontal central.



```

 //Retardo4.c
char Bin2_7seg(char digit); //Prototipo de la función.

signed char numero,i;

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0);//Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1){
  numero=PORTA; //Leer el valor del puerto A.
  if (numero>9 || numero==0){ //Si es mayor que 9 o igual a cero.
    PORTB=0x40; //Encender el segmento horizontal central.
    continue; //Y esperar por un nuevo valor.
  }

  for (i=numero;i>=0;i--){ //Si está entre 9 y 1.
    PORTB=Bin2_7seg(i);
    Delay_1sec();
  }
}

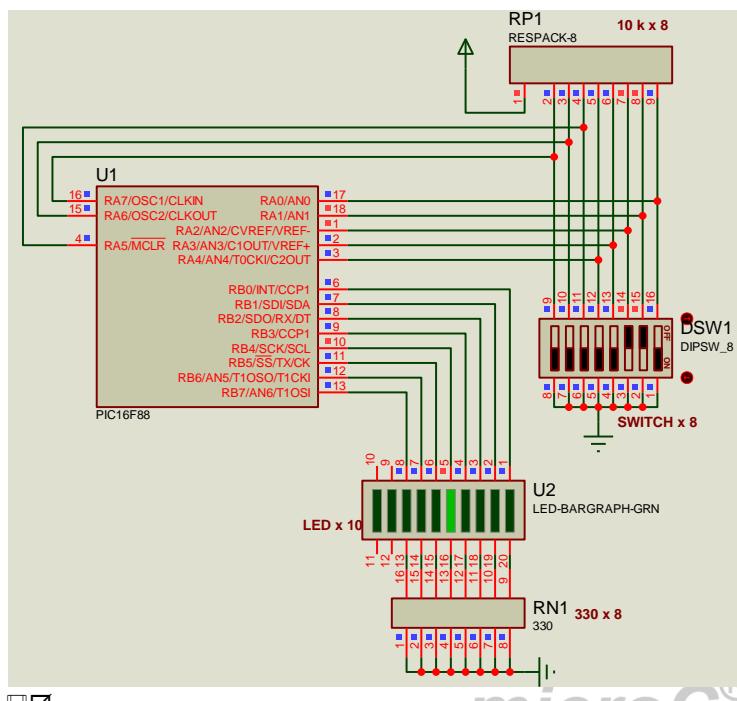
char Bin2_7seg(char digit){ //Definición de la función.

```

```

switch (digit){
    case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
    case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
    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 0x67;
}
}

```



**Retardo5.c:** Por la barra de LEDs se visualiza un juego de luces cualquiera. La velocidad del movimiento será fijada por los bits RA2:RA0, de manera que se visualice cada posición durante un tiempo:

- Si PORTA=000 (0), los LEDs están apagados.
- Si PORTA=001 (1), cada posición se visualiza 100 ms.
- Si PORTA=010 (2), cada posición se visualiza 200 ms.
- Y así sucesivamente, hasta...
- Si PORTA=111 (7), cada posición se visualiza 700 ms.

Figura 2.9 Circuito del problema  
Retardo5.c

```

 Retardo5.c
//Retardo5.c
//Enciende un LED empezando por RB0 y se va desplazando hasta RB7,
//permanece en cada posición un múltiplo de 100 ms, y luego vuelve
//a empezar.
signed char i,j,factor;

void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1){
    factor=PORTA & 0b00000111;//Leer el puerto A y tomar RA2:RA0.
    if (factor==0){ //Si el valor ingresado es 0.
        PORTB=0x00; //No encender ningún LED.
        continue; //Y esperar por un nuevo valor.
    }
    PORTB=0x01; //Encendido de RB0 por primera vez.
    for (j=factor;j>=1;j--) //Retardo=factor*100 ms.
        Delay_ms(100);

    for (i=7;i>=1;i--){
        PORTB=PORTB<<1;
        for (j=factor;j>=1;j--) //Retardo=factor*100 ms.
            Delay_ms(100);
    }
}
}

```

En el siguiente ejemplo se emplea el oscilador interno con una frecuencia de 4 MHz, para lograr mayor precisión en los retardos. El lector puede probar con otras frecuencias, si así lo desea. En la ventana *Watch Clock* se utiliza el *Stopwatch* para cronometrar con exactitud el tiempo de nivel alto (1) y bajo (0) del pin RB7 y, por ensayo y error, encontrar los valores correctos para los retardos.

**Retardo6.c:** Por la línea RB7 se genera una onda cuadrada de 1 kHz en la que cada semiperíodo durará 500 us exactamente. Tomar en cuenta los tiempos que tardan en ejecutarse las instrucciones. Para calcular los valores de los retardos hay que ayudarse del simulador de mikroC™ y la ventana *Watch Clock*. 4MHz☒.

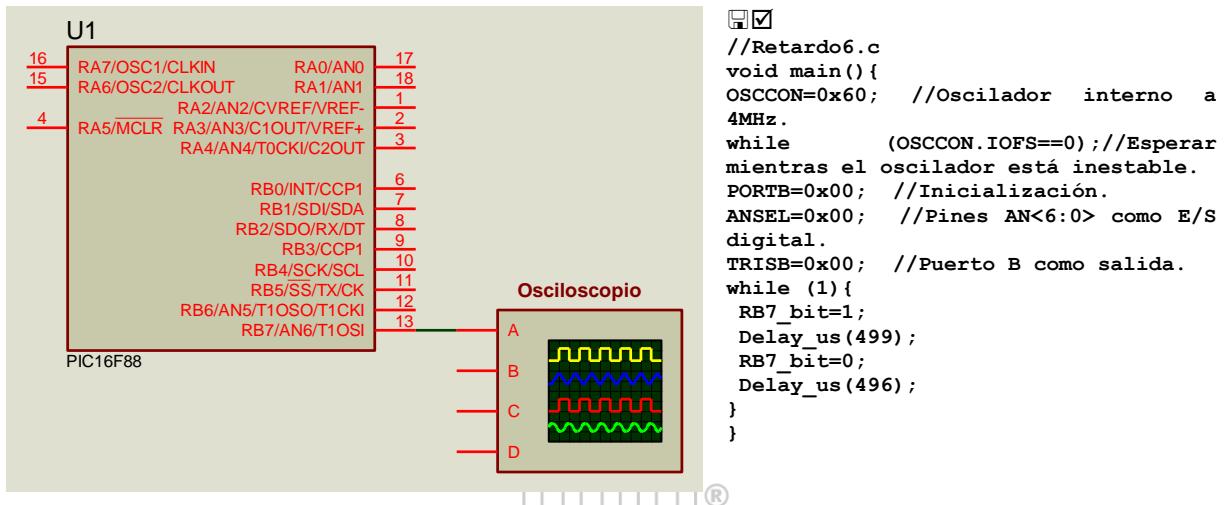
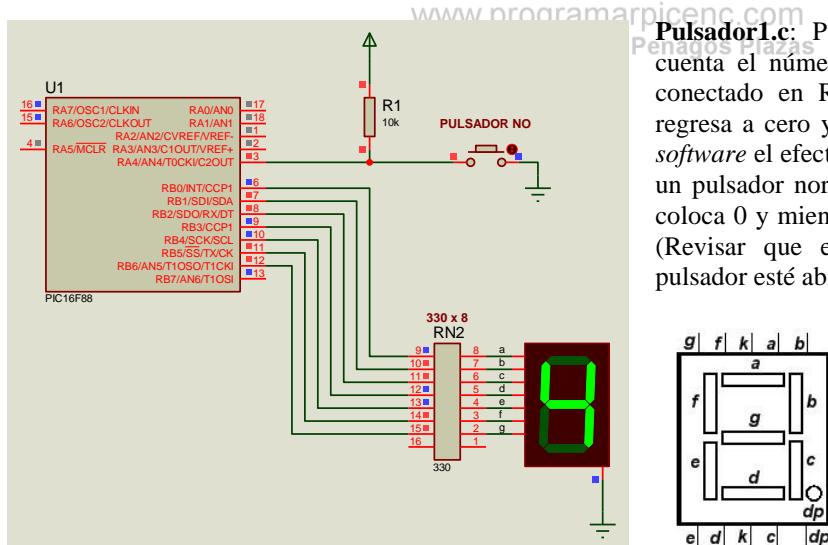


Figura 2.10 Circuito del problema Retardo6.c

En el ejemplo que sigue se puede apreciar el empleo de la función *Button* (**Button (&PORTA, 4, 1, 0)**) de la librería del mismo nombre. Esta función elimina el efecto de rebote (*bouncing*) que se produce al presionar un pulsador. Se debe añadir la librería *Button*. Los parámetros utilizados son los siguientes: **&PORTA**, puerto de conexión del pulsador; **4**, el número del pin; **1**, tiempo (ms) de espera (*debounce time*) y **0**, nivel lógico cuando el pulsador está presionado (ó 1 cuando está libre).

La función devuelve Verdadero (255) o Falso (0), dependiendo del estado en que se encuentre el pulsador. En el ejemplo, devuelve Verdadero (255) siempre y cuando el nivel lógico sea 0 después de 1 ms de mantenerlo presionado.



**Pulsador1.c:** Por el *display* de 7 segmentos se cuenta el número de activaciones de un pulsador conectado en RA4. Cuando supere 9, el conteo regresa a cero y comienza de nuevo. Eliminar por *software* el efecto de rebote del pulsador. Se emplea un pulsador normalmente abierto (*NO*), al cerrarlo coloca 0 y mientras está abierto coloca 1 en el pin. (Revisar que el interruptor en paralelo con el pulsador esté abierto). *Button☒*.

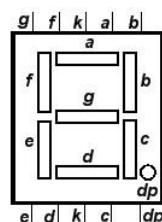


Figura 2.11.1 Circuito del problema Pulsador1.c (PIC16F88 y 16F628A)

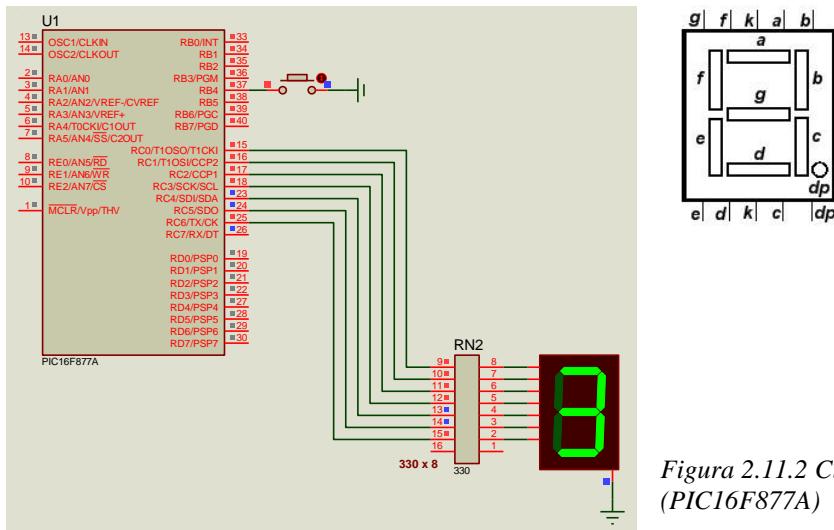


Figura 2.11.2 Circuito del problema Pulsador1.c (PIC16F877A)

```


//Pulsador1.c
char Bin2_7seg(char digit); //Prototipo de la función.

char contador=0,estado=1;

void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1){
if (Button(&PORTA,4,1,0)) estado=0; //Si se pulsa.
if (estado==0 && Button(&PORTA,4,1,1));//Si se pulsa y se libera.
{
contador++;
if (contador>9) contador=0;
estado=1;
}
PORTB=Bin2_7seg(contador);
}
}

char Bin2_7seg(char digit){ //Definición de la función.
switch (digit)
{
case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
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 0x67;
}
}
}

```

**microC®**

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

**Pulsador2.c:** Simula un dado electrónico que cuenta de 1 a 6. Mientras se mantiene activado un pulsador en RA4, el *display* contará de 1 a 6 continuamente, manteniéndose un instante en cada valor. Cuando se libere permanecerá en el último valor. Eliminar por *software* el efecto de rebote del pulsador. Se emplea un pulsador normalmente abierto (*NO*), al cerrarlo coloca 0 y mientras está abierto coloca 1 en el pin. (Revisar que el interruptor en paralelo con el pulsador esté abierto). *Button*☒.

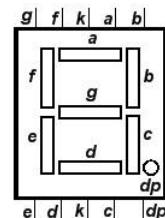
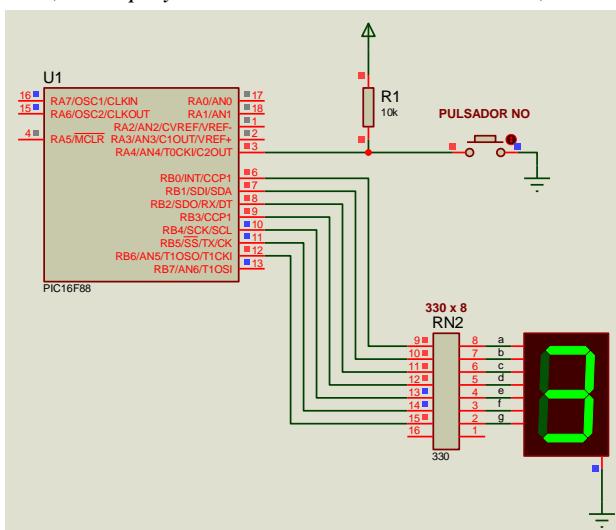


Figura 2.12 Circuito del problema Pulsador2.c

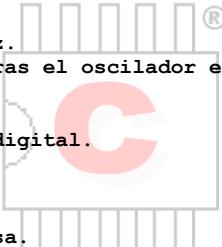
```


//Pulsador2.c
char Bin2_7seg(char digit); //Prototipo de la función.

char contador=1;
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
PORTB=0x00;
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
TRISB=0x00; //Puerto B como salida.
while (1)
{
if (Button(&PORTA,4,1,0)) //Si se pulsa.
{
contador++;
if (contador>6) contador=1;
}
PORTB=Bin2_7seg(contador);
}
}

char Bin2_7seg(char digit){ //Definición de la función.
switch (digit)
{ case 0: return 0x3F; //0x3F es el código 7-segmentos del 0.
case 1: return 0x06; //0x06 es el código 7-segmentos del 1.
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 0x67;
}
}

```



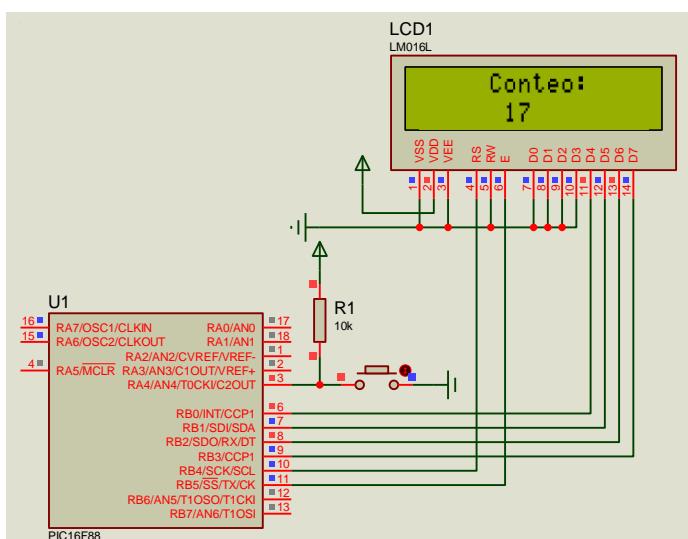
microC®

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO III: LCD

Las pantallas de cristal líquido (LCD) se han popularizado mucho en los últimos años, debido a su gran versatilidad para presentar mensajes de texto (fijos y en movimiento), valores numéricos y símbolos especiales, su precio reducido, su bajo consumo de potencia, el requerimiento de solo 6 pines del PIC para su conexión y su facilidad de programación en lenguajes de alto nivel (por ejemplo, lenguaje C). Desde todo punto de vista el empleo de LCDs debería considerarse como la primera opción a la hora de decidir por un dispositivo de presentación alfanumérica, excepto cuando las condiciones de iluminación ambiental no sean las más favorables. En este último caso se debería pensar en el empleo de *displays* de 7 segmentos, que aunque no tienen la misma versatilidad tienen la ventaja innegable de sus mejores características de visibilidad aún en los ambientes más desfavorables. En la actualidad existen diversos modelos de LCDs, aunque los más comunes son los de 16 caracteres por dos filas, gobernados por el microcontrolador Hitachi HD44780, que se ha convertido en el estándar de facto para las aplicaciones con PICs. Específicamente se hará referencia al módulo LM016L, que tiene las características mencionadas, aunque cualquier otro módulo LCD con el controlador HD44780 o compatible se puede programar con las funciones indicadas aquí.

### 3.1 CONEXIÓN DEL LCD AL PIC



La conexión más recomendable del módulo LCD requiere 4 pines para los datos (D7:D4), 1 pin para habilitar/deshabilitar el LCD (E) y 1 pin para los modos comando/carácter (RS). En las figuras 3.1.1 y 3.1.2 se indica la forma de conectar el módulo LCD a los PICs 16F88, 16F628A y 16F877A.

Figura 3.1.1 Conexión del LCD al PIC16F88 (16F628A) con 4 bits

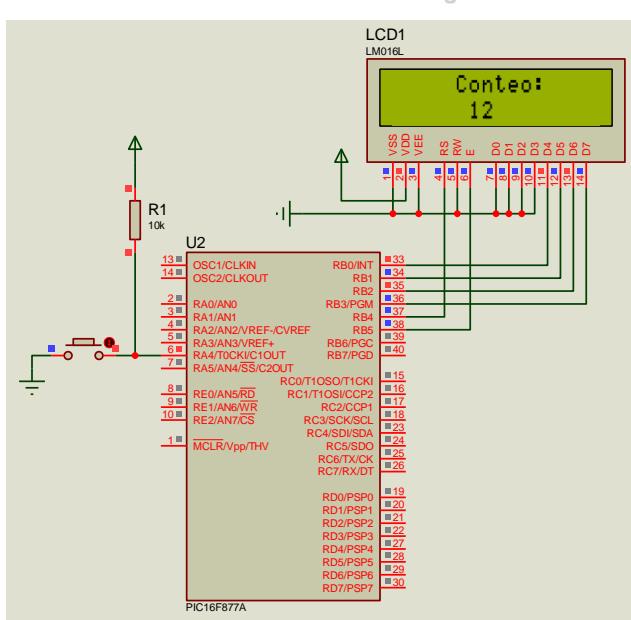


Figura 3.1.2 Conexión del LCD al PIC16F877A con 4 bits

### 3.2 FUNCIONES DE mikroC™ PARA LCD

mikroC™ proporciona una librería para comunicación con LCDs (con el controlador HD44780 o compatibles) a través de un interfaz de 4 bits para datos. Para el trabajo con el módulo LCD se debe añadir la librería *Lcd*, que contiene las funciones listadas en la tabla 3.1. Para poder utilizar estas funciones se debe declarar previamente un total de 12 variables: 6 que definen

los pines del PIC y otras 6 que permiten programar su sentido de circulación de datos (se detallarán en los ejemplos de este capítulo).

Función	Descripción
Lcd_Init()	Inicializa el módulo LCD
Lcd_Out(fila, columna, texto)	Visualiza texto en la posición especificada
Lcd_Out_CP(texto)	Visualiza texto en la posición actual del cursor
Lcd_Chr(fila, columna, carácter)	Visualiza un carácter en la posición especificada
Lcd_Chr_CP(carácter)	Visualiza un carácter en la posición actual del cursor
Lcd_Cmd(comando)	Envía un comando al LCD

Tabla 3.1 Funciones para LCD de la librería Lcd de mikroC™

La lista de comandos disponibles predefinidos en mikroC™ para el control del LCD se muestra en la tabla 3.2.

Comando LCD	Propósito
_LCD_FIRST_ROW	Mueve el cursor a la primera fila
_LCD_SECOND_ROW	Mueve el cursor a la segunda fila
_LCD_CLEAR	Borra la pantalla
_LCD_RETURN_HOME	Retorna el cursor a su origen, retorna una pantalla desplazada a su posición original. La RAM del LCD no se ve afectada
_LCD_CURSOR_OFF	Apaga el cursor
_LCD_UNDERLINE_ON	Enciende el cursor de subrayado
_LCD_BLINK_CURSOR_ON	Enciende el cursor con parpadeo
_LCD_MOVE_CURSOR_LEFT	Mueve el cursor a la izquierda sin modificar la RAM del LCD
_LCD_MOVE_CURSOR_RIGHT	Mueve el cursor a la derecha sin modificar la RAM del LCD
_LCD_TURN_ON	Enciende el módulo LCD
_LCD_TURN_OFF	Apaga el módulo LCD
_LCD_SHIFT_LEFT	Desplaza la pantalla a la izquierda sin modificar la RAM del LCD
_LCD_SHIFT_RIGHT	Desplaza la pantalla a la derecha sin modificar la RAM del LCD

Tabla 3.2 Lista de comandos disponibles para el control del LCD

### 3.3 EJEMPLOS DE MENSAJES DE TEXTO FIJOS Y EN MOVIMIENTO CON VALORES NUMÉRICOS

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

En el siguiente ejemplo se emplea la función *ByteToStr* de la librería *Conversions*, para convertir el contenido de la variable “contador” (un byte) en una cadena de caracteres (*string*) y así poderlo visualizar en el LCD con la función *Lcd\_Out*.

**LCD1.c:** Cada vez que presiona el pulsador conectado en RA4 se incrementa un contador que se visualiza en el centro de la segunda línea de la pantalla (figuras 3.1.1 y 3.1.2). Si la cuenta supera 100, el conteo se reinicia desde 0. En el centro de la primera línea se muestra la palabra “Conteo.”. *Conversions*  *LCD*  *Button* .

```


//LCD1.c
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RB4_bit;

```

```

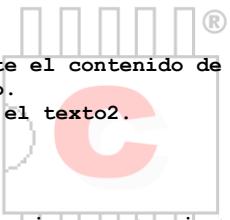
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

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;
// Fin de declaración de variables de conexión.

char contador=0,estado=1,Texto1[]="Conteo:", texto2[4];

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,6,Texto1); //Escribe el Texto1.
while (1)
{
if (Button(&PORTA,4,1,0)) estado=0; //Si se pulsa.
if (estado==0 && Button(&PORTA,4,1,1)) //Si se pulsa y se libera.
{
contador++; // "contador" contiene el número de pulsaciones.
if (contador>100) contador=0;
estado=1;
}
ByteToStr(contador, texto2); //Convierte el contenido de la variable
//en texto.
Lcd_Out(2,6,Texto2); //Escribe el texto2.
}
}

```



**LCD2.c:** Igual que el anterior, pero se incrementa mientras se mantenga presionado el pulsador, una cuenta cada 200 ms (figuras 3.1.1 y 3.1.2). *Conversions*  *LCD*  *Button*

```

 //LCD2.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador=0, Texto1[]="Conteo:", texto2[4];

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,6,Texto1); //Escribe el Texto1.
while (1)
{
if (Button(&PORTA,4,1,0)) //Si se pulsa.
{
contador++;
}
}
}

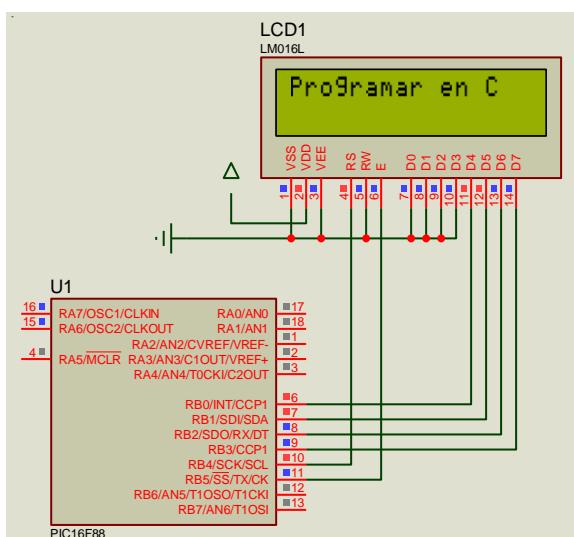
```

*microc*®  
www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

```

    if (contador>100) contador=0;
}
ByteToStr(contador, texto2);
Lcd_Out(2,6, texto2);
Delay_ms(200);
}
}

```



**LCD3.c:** En la pantalla se visualizan varios mensajes, uno tras otro. Cada mensaje permanece durante 2 s. Entre mensaje y mensaje la pantalla permanece apagada 200 ms. **LCD☒.**

Figura 3.2 Circuito del problema LCD3.c

//LCD3.c  
void Retardo();  
//Declaración de las 12 variables necesarias para la conexión 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;

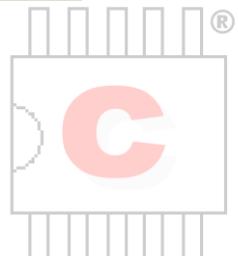
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Fin de declaración de variables de conexión.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

while (1)
{
    Lcd_Out(1,1,"Programar en C");
    Retardo();
    Lcd_Out(1,1,"es su mejor ");
    Retardo();
    Lcd_Out(1,1,"opcion.Sin duda.");
    Retardo();
}

void Retardo(){
Delay_ms(2000);
Lcd_Cmd(_LCD_TURN_OFF);
Delay_ms(200);
Lcd_Cmd(_LCD_TURN_ON);
Lcd_Cmd(_LCD_CLEAR);
}

```



microC®

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

**LCD4.c:** En la primera línea aparecerá un mensaje fijo. En la segunda un mensaje parpadeante (circuito de la figura 3.2). **LCD☒.**

```

 //LCD4.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,1,"Texto fijo.");
while (1)
{
    Lcd_Out(2,1,"Parpadeante.");
    Delay_ms(500);
    Lcd_Out(2,1,""); //Borra el texto.
    Delay_ms(500);
}
}

```



El siguiente ejemplo muestra como se pueden visualizar mensajes de más de 16 caracteres desplazando la pantalla hacia la derecha con el comando \_LCD\_SHIFT\_RIGHT. Imagine que el texto está fijo y la pantalla se mueve sobre él hacia la derecha, entonces se tiene la sensación de que el texto se mueve a la izquierda. Los mensajes largos tienen una longitud máxima de 40 caracteres por línea, y el contenido de una línea es independiente del contenido de la otra.

**microC®**

**LCD5.c:** En la pantalla se visualiza un mensaje largo (de más de 16 caracteres) que se desplaza a lo largo de la pantalla (circuito de la figura 3.2). **LCD**

```

 //LCD5.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char i;

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
while (1)
{
    Lcd_Cmd(_LCD_CLEAR);

```

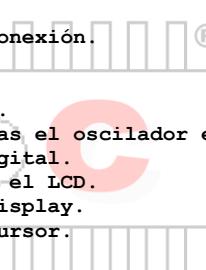
```

Lcd_Out(1,1,"Mensajes largos: maximo de 40 caracteres.");
Delay_ms(400);
for (i=40;i>0;i--)
{
  Lcd_Cmd(_LCD_SHIFT_RIGHT); //Desplaza la pantalla (no el texto) a la
                             //derecha.
  Delay_ms(250);
}
}
}

```

**LCD6.c:** En la pantalla se visualiza “Cerrado” o “Abierto”, según si un pulsador conectado en RA4 está presionado o no (circuito de las figuras 3.1.1 y 3.1.2). *LCD* . *Button* .

```


//LCD6.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
while (1)
{
  if (Button(&PORTA,4,1,0)) //Si se pulsa.
  {
    Lcd_Out(1,6,"Cerrado..."); //Continua;
    Lcd_Out(1,6,"Abierto...");
  }
}
}

```

**microC®**

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

Para presentar **números decimales en el LCD** se emplea el siguiente procedimiento por medio de la función *FloatToStr* de la librería *Conversions*. Este ejemplo presenta las tres primeras cifras significativas (y el punto decimal en la posición correspondiente):

```

char texto[15];           //Arreglo para almacenar el resultado de la conversión.
float variable=10.71;     //Variable real con un valor inicial de 10,71

void main(){
...
FloatToStr(variable,texto); //La variable texto contiene "10.70999".
Lcd_Ch(1,1,*text); //Presenta la 1era cifra significativa (1).
Lcd_Ch(1,2,*text+1); //Presenta la 2da cifra significativa (0).
Lcd_Ch(1,3,*text+2); //Presenta el punto decimal.
Lcd_Ch(1,4,*text+3); //Presenta la 3era cifra significativa (7).
}

```

### 3.4 CREACIÓN DE CARACTERES ESPECIALES

En algunas aplicaciones puede ser muy útil, y a la vez atractivo para el usuario, presentar información empleando caracteres especiales que no se encuentran en la ROM del Generador de Caracteres (CGROM) del módulo LCD, la cual es capaz de generar 160 tipos estandarizados de caracteres de 5x7 puntos. El usuario dispone de un área de memoria RAM que puede escribir con sus propios caracteres, para luego visualizarlos a voluntad.

### 3.4.1 RAM del Generador de Caracteres (CGRAM)

La CGRAM tiene un tamaño de 64 bytes, en los que el usuario puede programar 8 caracteres propios (No. 0 a No. 7) de 5x7 puntos.

Para programar y visualizar un carácter en la CGRAM se realizan los siguientes pasos:

1. Ubicar el puntero de la CGRAM en la fila No. 0 del carácter No. x (x= 0, 1, 2, ..., 7).
2. Escribir la fila seleccionada del carácter seleccionado.
3. Como el No. de fila se incrementa automáticamente, retornar al paso 2 hasta escribir las 6 filas restantes.
4. Retornar el Contador de Direcciones (AC) a una dirección de la RAM de Datos en Pantalla (DDRAM).
5. Visualizar el carácter No. x en la dirección DDRAM seleccionada en el paso 4.

La herramienta *LCD Custom Character* permite generar el código fuente de forma rápida y efectiva para implementar los pasos anteriores.

### 3.4.2 Herramienta LCD Custom Character de mikroC™

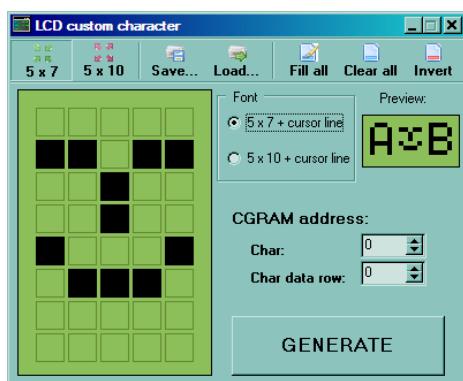


Figura 3.3 Ventana de dibujo de caracteres personalizados

La herramienta *LCD Custom Character* del menú *Tools* se emplea para el diseño de caracteres personalizados siguiendo un procedimiento muy sencillo. En una matriz gráfica de 5x7 puntos (la octava fila es para el cursor) se dibuja el símbolo deseado haciendo clic en los puntos correspondientes (figura 3.3). Una vez que el diseño está terminado y a gusto del usuario, se hace clic en la opción *GENERATE*, lo cual produce el código fuente que se deberá insertar en la aplicación, copiéndolo primero en el portapapeles (figura 3.4). La función *CustomChar(fila, columna)* permite visualizar el carácter en una posición determinada. Se puede generar un máximo de 8 caracteres para una aplicación, ya que esto está limitado por la memoria interna del módulo LCD (CGRAM).

```

Code
mikroPascal | mikroBasic | mikroC |
const char character[] = {0,27,4,4,17,14,0,0};

void CustomChar(char pos_row, char pos_char) {
    char i;
    LCD_Cmd(64);
    for (i = 0; i<=7; i++) LCD_Ch_Cp(character[i]);
    LCD_Cmd(LCD_RETURN_HOME);
    LCD_Ch(pos_row, pos_char, 0);
}

```

Figura 3.4 Ventana de código generado

◆ El comando `LCD_RETURN_HOME` debe ser corregido a `_LCD_RETURN_HOME`.

A continuación se muestra un ejemplo completo del empleo de esta herramienta.

El código generado para cada símbolo ha sido modificado renombrando la variable *character* para cada función, y el nombre de función ha sido cambiado para reflejar la tarea particular que realiza. El primer carácter debe estar asociado al número 0 (*Char: 0*), el segundo al número 1 (*Char: 1*), y así sucesivamente hasta el cuarto carácter (*Char: 3*). El número de fila (*Char data row*) debería estar siempre en 0, a menos que se diga lo contrario.

**LCD7.c:** Visualizar los siguientes símbolos (figura 3.5): un parlante (*Char: 0*), una pila (*Char: 1*), una campana (*Char: 2*) y un reloj (*Char: 3*). El significado de cada uno depende del contexto de aplicación. Por ejemplo, el parlante puede significar que una alarma sonora está en alerta, la pila puede indicar que el voltaje de alimentación de un circuito está en el valor correcto, la campana indicaría que hay un fallo en una de las fuentes de alimentación de un circuito, y, el reloj puede representar un temporizador activado (circuito de la figura 3.2). *LCD*✓.

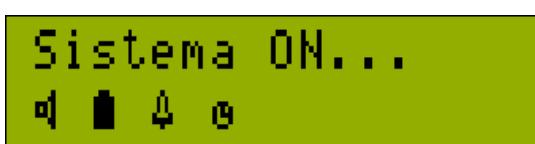


Figura 3.5 Símbolos del ejemplo LCD7.c

```


//LCD7.c
//Corregir el comando LCD_RETURN_HOME a _LCD_RETURN_HOME.
//La variable character ha sido renombrada en cada caso.
//Parlante
const char character0[] = {1,3,29,21,29,3,1,0};           //character0
void Parlante(char pos_row, char pos_char) {
    char i;
    LCD_Cmd(64);
    for (i = 0; i<=7; i++) LCD_Ch_Cp(character0[i]); //character0
    LCD_Cmd(_LCD_RETURN_HOME);
    LCD_Ch(pos_row, pos_char, 0);
}

//Pila
const char character1[] = {14,31,31,31,31,31,0};      //character1
void Pila(char pos_row, char pos_char) {
    char i;
    LCD_Cmd(72);
    for (i = 0; i<=7; i++) LCD_Ch_Cp(character1[i]); //character1
    LCD_Cmd(_LCD_RETURN_HOME);
    LCD_Ch(pos_row, pos_char, 1);
}

//Campana
const char character2[] = {4,10,10,10,17,31,4,0};
void Campana(char pos_row, char pos_char) {
    char i;
    LCD_Cmd(80);
    for (i = 0; i<=7; i++) LCD_Ch_Cp(character2[i]);
    LCD_Cmd(_LCD_RETURN_HOME);
    LCD_Ch(pos_row, pos_char, 2);
}

//Reloj
const char character3[] = {0,0,14,21,23,17,14,0};
void Reloj(char pos_row, char pos_char) {
    char i;
    LCD_Cmd(88);
    for (i = 0; i<=7; i++) LCD_Ch_Cp(character3[i]);
    LCD_Cmd(_LCD_RETURN_HOME);
    LCD_Ch(pos_row, pos_char, 3);
}

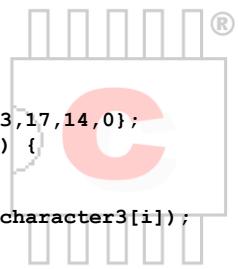
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

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;
// Fin de declaración de variables de conexión.

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,1,"Sistema ON...");

Parlante(2,1); //Símbolo en la fila 2, columna 1.
Pila(2,3); //Símbolo en la fila 2, columna 3.
Campana(2,5); //Símbolo en la fila 2, columna 5.
Reloj(2,7); //Símbolo en la fila 2, columna 7.
}

```



www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO IV: EEPROM DE DATOS

Esta memoria tiene la capacidad para ser programada y reprogramada por la CPU del PIC, para asegurar que en caso de una falla o desconexión de la energía los valores o variables críticas se puedan mantener en una memoria no volátil. La EEPROM es muy útil en procesos que deben continuar a partir del último dato obtenido cuando se ha producido una interrupción en la energía. mikroC™ incluye una librería con funciones que facilitan enormemente el trabajo con la EEPROM.

### 4.1 FUNCIONES DE mikroC™ PARA LA EEPROM

En la tabla 4.1 se describen las funciones que permiten escribir en (y leer de) la EEPROM de datos del PIC.

Función	Descripción
EEPROM_Read(dirección)	Retorna un byte de la dirección especificada
EEPROM_Write(dirección, dato)	Escribe un dato en la dirección especificada

Tabla 4.1 Funciones para trabajo con la EEPROM

Ambas funciones soportan PICs inclusive con más de 256 bytes de EEPROM (recuerde que los PICs 16F88 y 16F877A tienen 256 bytes, mientras que el 16F628A tiene 128 bytes). Todas las interrupciones serán deshabilitadas durante la ejecución de la función *EEPROM\_Write* (el bit GIE del registro INTCON será igual a cero). Al finalizar la ejecución, la función restaura el estado previo de este bit. Se debe asegurar un retardo mínimo de 20 ms entre el uso sucesivo de las funciones *EEPROM\_Write* y *EEPROM\_Read*; de lo contrario, aunque el PIC escribirá el valor correcto, la lectura con *EEPROM\_Read* puede dar un resultado indefinido. Para tener a disposición estas funciones se debe añadir la librería *EEPROM*.

### 4.2 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

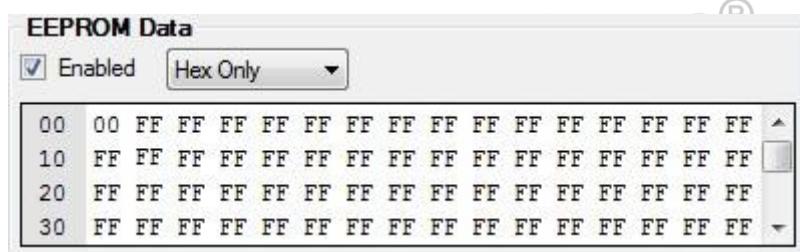


Figura 3.6 Campo de Datos EEPROM del programa PICkit2

En los ejemplos EEPROM1.c y EEPROM2.c se debe grabar el valor inicial del contador (00) en la primera dirección EEPROM (0x00)

en el momento de la programación. Para ello, una vez que el programa de aplicación PICkit2 v2.61 está listo, se hace clic en el espacio **EEPROM Data** sobre el dato que se desea cambiar y se introduce por medio del teclado (figura 3.6). A continuación grabar el PIC con el comando *Write*. Todos los detalles del proceso de grabación se encuentran en el **Apéndice D**.

**EEPROM1.c:** Cada vez que el PIC es reiniciado se incrementa un contador que se guarda en la primera posición EEPROM y es visualizado en el LCD (circuito de la figura 3.2). *Conversions*  *LCD*  *EEPROM*

```

 EEPROM1.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador, texto[4];

void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

contador=EEPROM_Read(0x00);
ByteToStr(contador, texto);
Lcd_Out(1,1,texto);
contador++;
EEPROM_Write(0x00,contador);
}

```

**EEPROM2.c:** Control de las máquinas de tipo “Su turno”. En el LCD se visualiza el número de turno actual. Este se incrementa cada vez que se presiona el pulsador conectado en RA4. La EEPROM almacena el último número visualizado, de forma que ante un fallo de alimentación se reanude la cuenta en el último número. Cuando el turno supera el número 100, el conteo regresa a 0 (circuito de las figuras 3.1.1 y 3.1.2). *Conversions*  *LCD*  *Button*  *EEPROM* .



```

//EEPROM2.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador=0,estado=1,texto[4]; //Ing. Juan Ricardo Penagos Plazas

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

while (1){
    contador=EEPROM_Read(0x00);
    ByteToStr(contador, texto);
    Lcd_Out(1,1,"Su turno:");
    Lcd_Out(1,10,texto);
    if (Button(&PORTA,4,1,0)) estado=0; //Si se pulsa.
    if (estado==0 && Button(&PORTA,4,1,1)) //Si se pulsa y se libera.
    {
        contador++;
        if (contador>100) contador=0;
        estado=1;
        EEPROM_Write(0x00,contador);
        Delay_ms(20);
    }
}
}

```



**microC®**

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO V: TEMPORIZADORES Y CONTADORES

### 5.1 Timer0 (16F88, 16F628A y 16F877A)

Este temporizador/contador tiene las siguientes características:

- Temporizador/contador de 8 bits.
- Se puede leer y escribir.
- Prescaler programable de 8 bits.
- Selección de reloj interno o externo.
- Genera una interrupción al desbordarse desde 0xFF a 0x00.
- Selección de flanco del reloj externo.

La operación del Timer0 se controla a través del registro OPTION\_REG (figura 5.1). En el modo temporizador (TOCS=0), se produce un incremento del registro TMR0 cada ciclo de instrucción (prescaler asignado al *Watchdog Timer WDT*). Si se escribe en el registro TMR0, no se produce el incremento durante los dos siguientes ciclos de instrucción; este hecho debe tenerse muy en cuenta por parte del usuario y, de ser necesario, ajustar el valor escrito en TMR0.

OPTION_REG: OPTION CONTROL REGISTER (ADDRESS 81h, 181h)																																		
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1																											
RBPU	INTEDG	TOCS	T0SE	PSA	PS2	PS1	PS0																											
bit 7							bit 0																											
<b>RBPU:</b> PORTB Pull-up Enable bit 1 = PORTB pull-ups are disabled 0 = PORTB pull-ups are enabled by individual port latch values																																		
<b>INTEDG:</b> Interrupt Edge Select bit 1 = Interrupt on rising edge of RB0/INT pin 0 = Interrupt on falling edge of RB0/INT pin																																		
<b>TOCS:</b> TMR0 Clock Source Select bit 1 = Transition on RA4/T0CKI/C2OUT pin 0 = Internal instruction cycle clock (CLK0)																																		
<b>T0SE:</b> TMR0 Source Edge Select bit 1 = Increment on high-to-low transition on RA4/T0CKI/C2OUT pin 0 = Increment on low-to-high transition on RA4/T0CKI/C2OUT pin																																		
<b>PSA:</b> Prescaler Assignment bit 1 = Prescaler is assigned to the WDT 0 = Prescaler is assigned to the Timer0 module																																		
<b>PS&lt;2:0&gt;:</b> Prescaler Rate Select bits																																		
<table border="1"> <thead> <tr> <th>Bit Value</th> <th>TMR0 Rate</th> <th>WDT Rate</th> </tr> </thead> <tbody> <tr><td>000</td><td>1 : 2</td><td>1 : 1</td></tr> <tr><td>001</td><td>1 : 4</td><td>1 : 2</td></tr> <tr><td>010</td><td>1 : 8</td><td>1 : 4</td></tr> <tr><td>011</td><td>1 : 16</td><td>1 : 8</td></tr> <tr><td>100</td><td>1 : 32</td><td>1 : 16</td></tr> <tr><td>101</td><td>1 : 64</td><td>1 : 32</td></tr> <tr><td>110</td><td>1 : 128</td><td>1 : 64</td></tr> <tr><td>111</td><td>1 : 256</td><td>1 : 128</td></tr> </tbody> </table>								Bit Value	TMR0 Rate	WDT Rate	000	1 : 2	1 : 1	001	1 : 4	1 : 2	010	1 : 8	1 : 4	011	1 : 16	1 : 8	100	1 : 32	1 : 16	101	1 : 64	1 : 32	110	1 : 128	1 : 64	111	1 : 256	1 : 128
Bit Value	TMR0 Rate	WDT Rate																																
000	1 : 2	1 : 1																																
001	1 : 4	1 : 2																																
010	1 : 8	1 : 4																																
011	1 : 16	1 : 8																																
100	1 : 32	1 : 16																																
101	1 : 64	1 : 32																																
110	1 : 128	1 : 64																																
111	1 : 256	1 : 128																																

Figura 5.1 Bits del registro OPTION\_REG

En el modo contador (TOCS=1), se produce un incremento por cada transición ascendente (T0SE=0) o descendente (T0SE=1) en el pin RA4.

#### 5.1.1 Prescaler

◆ Un prescaler es un circuito que reduce la frecuencia que ingresa a un temporizador/contador dividiéndola para un determinado valor (figura 5.2). Por ejemplo, si la relación es 1:8, el prescaler entrega una frecuencia igual a la octava parte de la frecuencia del oscilador.

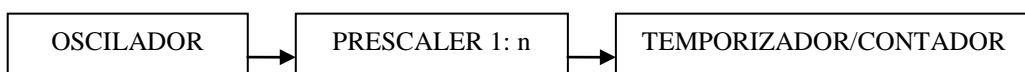


Figura 5.2 Prescaler actuando como divisor de frecuencia

El prescaler es compartido entre el Timer0 y el Temporizador de Vigilancia (*Watchdog Timer WDT*), y no se puede leer ni escribir. Cuando se asigna el prescaler al Timer0 no puede ser utilizado por el *WDT* al mismo tiempo, y viceversa.

Los bits PSA y PS<sub>2:0</sub> determinan la asignación y la relación de división del prescaler. Cuando se asigna al Timer0, todas las instrucciones de escritura en el registro TMR0 reinician el prescaler. Cuando se asigna al WDT, una instrucción CLRWDT reinicia el prescaler y también el WDT.

### 5.1.2 Ejemplos de programación del Timer0

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

**Timer0\_1.c:** Cada vez que se actúe sobre el pulsador conectado en RA4 se incrementa un contador que se visualiza en el LCD (circuito de las figuras 3.1.1 y 3.1.2). *Conversions*  *LCD* .

```


//Timer0_1.c
//El registro OPTION_REG tiene todos sus bits en 1 después del encendido
//por lo tanto el Timer0 actúa como contador, incrementa en transición
//descendente y el prescaler está asignado al WDT.
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.
char texto[4];

void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Pines AN6:0 como E/S digital.
TMR0=0; //Inicializa el registro TMR0.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,6,"Conteo:");
while (1)
{
ByteToStr(TMR0,texto);
Lcd_Out(2,6,texto);
}
}
}

```

Cuando el Timer0 trabaja como temporizador cuenta los ciclos de instrucción (sin prescaler) o los ciclos que recibe del prescaler. Como es un contador ascendente el TMR0 debe ser cargado con el valor de los ciclos que se desean contar restados de 256 que es el valor de desbordamiento. Por ejemplo, para contar 28 ciclos (de instrucción/prescaler), se carga el TMR0 con 256-28=228. El ciclo de instrucción tiene una duración de 4 us para una frecuencia de oscilador de 1 MHz (16F88). Sin prescaler el Timer0 mediría un tiempo de 28x4x1 us = 112 us. Con un prescaler 1:8, el tiempo medido sería 28x4x8 us = 896 us. De manera general, el intervalo de temporización  $T$  se puede calcular con la siguiente fórmula:

$$T = N \times T_{CI} \times n$$

Donde:

- $N$  = número de ciclos de instrucción/prescaler
- $T_{CI}$  = período del ciclo de instrucción
- $n$  = valor del prescaler

Mientras que el valor de carga  $Q$  del TMR0 se calcula así:

$$Q=256-N$$

Para medir 500 us, con un prescaler 1:1 (prescaler asignado al *Watchdog Timer WDT*) y un  $T_{Cl} = 4$  us se necesitan  $500/4 = 125$  ciclos de instrucción. El valor inicial del TMR0 debe ser  $256-125=131$ . Este valor se emplea en el siguiente ejemplo.

Para los **PICs 16F628A y 16F877A** (4 MHz) se tienen los siguientes valores:  $T_{Cl} = 1$  us, N=500/1=500 ciclos de instrucción, pero el valor máximo de N puede ser 256. Por lo tanto se tiene que trabajar con un prescaler 1:4 para obtener los mismos resultados que con una frecuencia de oscilador de 1MHz (observe el código fuente en la carpeta correspondiente).

**Timer0\_2.c:** Por la línea RB7 se genera una onda cuadrada de 1 kHz en la que cada semiperíodo durará 500 us (aproximadamente). El cálculo del valor inicial del TMR0 se hará de forma simple, sin tomar en cuenta los tiempos que tardan en ejecutarse las instrucciones.

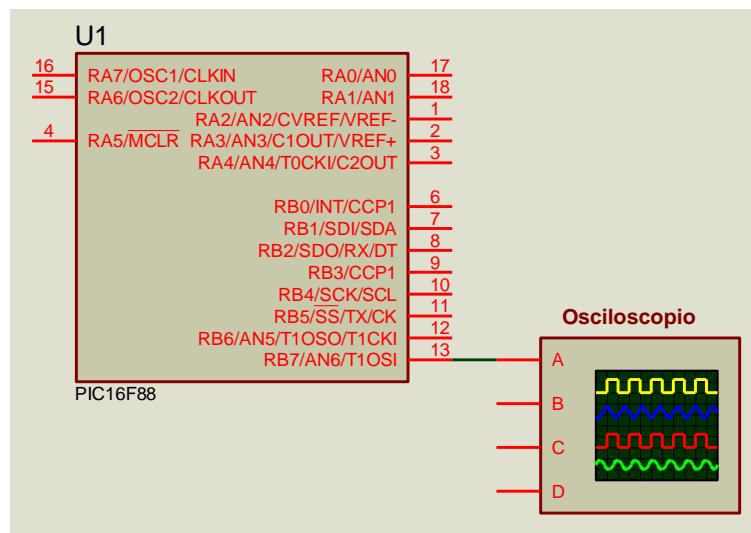


Figura 5.3 Circuito del problema Timer0\_2.c

```

 //Timer0_2.c
//El registro OPTION_REG tiene todos sus bits en 1 después del encendido
//por lo tanto el Timer0 actúa como contador, incrementa en transición
//descendente y el prescaler está asignado al WDT.

//Esta función genera un retardo de 500 us aproximadamente.
void Timer0_500us()
{
    TMR0=131;           //Valor inicial del TMR0.
    TMROIF_bit=0;       //Borra la bandera de interrupción.
    while (TMROIF_bit==0); //Verifica la bandera de interrupción.
}

void main()
{
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00; //Inicialización.
    ANSEL=0x00; //Pines AN<6:0> como E/S digital.
    TRISB=0x00; //Puerto B como salida.
    TOCS_bit=0; //Timer0 como temporizador.

    while (1)
    {
        RB7_bit=1;
        Timer0_500us();
        RB7_bit=0;
        Timer0_500us();
    }
}

```

**Timer0\_3.c:** Cuenta el número de veces que se presiona un pulsador durante 1 s, rebotes incluidos (circuito de las figuras 3.1.1 y 3.1.2). *Conversions*  *LCD*

```


//Timer0_3.c
//El registro OPTION_REG tiene todos sus bits en 1 después del encendido
//por lo tanto el Timer0 actúa como contador, incrementa en transición
//descendente y el prescaler está asignado al WDT.
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.
char texto[4], numflancos;
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Pines AN<6:0> como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
while (1)
{
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
TMR0=0; //Inicializa el registro TMR0.
Lcd_Out(1,6,"Pulse..."); //Espera hasta soltar el pulsador.
Delay_1sec();
numflancos=TMR0;
Lcd_Out(1,6,"Suelte..."); //Espera hasta las nuevas pulsaciones.
ByteToStr(numflancos,texto);
Lcd_Out(1,6,"Puls.+Reb.=");
Lcd_Out(2,6,texto);
Delay_ms(5000);
}
}

```



*microC®*

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## **CAPÍTULO VI: CARACTERÍSTICAS ESPECIALES Y OTROS RECURSOS**

### **PIC16F88**

Este dispositivo tiene un conjunto de características destinadas a maximizar la confiabilidad, minimizar el costo por medio de la eliminación de componentes externos, proporcionar ahorro de energía y protección del código.

#### **6.1 RESET**

El PIC 16F88 diferencia varias clases de reset:

- Reset al encendido (POR).
- Reset #MCLR durante operación normal.
- Reset #MCLR durante *Sleep*.
- Reset por WDT durante operación normal.
- Despertar por WDT durante *Sleep*.
- Reset por Desvanecimiento (BOR).

Algunos registros no se ven afectados por un reset de cualquier tipo. Su estado es desconocido en POR y no cambia en cualquier otro reset. La mayoría de los otros registros son restablecidos a un *estado de reset* en POR, reset #MCLR y WDT, reset #MCLR durante *Sleep* y BOR. No son afectados por un despertar por WDT, que es visto como una reanudación de la operación normal. Los bits #TO y #PD del registro STATUS se emplean en *software* para determinar la naturaleza del reset. En un POR, BOR o despertar, la CPU requiere aproximadamente 5-10 us para estar lista para la ejecución del código (si PWRT no está habilitado). En las tablas 6.1 y 6.2 se describen los estados de reset de todos los registros.

Condition	Program Counter	STATUS Register	PCON Register
Power-on Reset	000h	0001 1xxxx	----- - - 0x
MCLR Reset during normal operation	000h	000u uuuuu	----- - - uu
MCLR Reset during Sleep	000h	0001 0uuuu	----- - - uu
WDT Reset	000h	0000 1uuuu	----- - - uu
WDT Wake-up	PC + 1	uuuu0 0uuuu	----- - - uu
Brown-out Reset	000h	0001 1uuuu	----- - - u0
Interrupt Wake-up from Sleep	PC + 1 <sup>(1)</sup>	uuuu1 0uuuu	----- - - uu

**Legend:** u = unchanged, x = unknown, - = unimplemented bit, read as '0'

**Note 1:** When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

Tabla 6.1 Estados de reset para registros especiales

Register	Power-on Reset, Brown-out Reset	MCLR Reset, WDT Reset	Wake-up via WDT or Interrupt
W	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF	N/A	N/A	N/A
TMR0	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	0000h	0000h	PC + 1 <sup>(2)</sup>
STATUS	0001 1xxx	000q quuu <sup>(3)</sup>	uuuq quuu <sup>(3)</sup>
FSR	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA (PIC16F87)	xxxx 0000	uuuu 0000	uuuu uuuu
PORTA (PIC16F88)	xxxx0 0000	uuu0 0000	uuuu uuuu
PORTB (PIC16F87)	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTB (PIC16F87)	00xx xxxx	00uu uuuu	uuuu uuuu
PCLATH	---0 0000	---0 0000	---u uuuu
INTCON	0000 000x	0000 000u	uuuu uuuu <sup>(1)</sup>
PIR1	-000 0000	-000 0000	-uuu uuuu <sup>(1)</sup>
PIR2	00-0 ----	00-0 ----	uu-u ---- <sup>(1)</sup>
TMR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	-000 0000	-uuu uuuu	-uuu uuuu
TMR2	0000 0000	0000 0000	uuuu uuuu
T2CON	-000 0000	-000 0000	-uuu uuuu
SSPBUF	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPCON	0000 0000	0000 0000	uuuu uuuu
CCPR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	--00 0000	--00 0000	--uu uuuu
RCSTA	0000 000x	0000 000x	uuuu uuuu
TXREG	0000 0000	0000 0000	uuuu uuuu
RCREG	0000 0000	0000 0000	uuuu uuuu
ADRESH	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	0000 00-0	0000 00-0	uuuu uu-0
OPTION_REG	1111 1111	1111 1111	uuuu uuuu
TRISA	1111 1111	1111 1111	uuuu uuuu
TRISB	1111 1111	1111 1111	uuuu uuuu
PIE1	-000 0000	-000 0000	-uuu uuuu
PIE2	00-0 ----	00-0 ----	uu-u ----
PCON	---- --0q	---- --uu	---- --uu
OSCCON	-000 0000	-000 0000	-uuu uuuu
OSCTUNE	--00 0000	--00 0000	--uu uuuu
PR2	1111 1111	1111 1111	1111 1111
SSPADD	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	0000 0000	0000 0000	uuuu uuuu
TXSTA	0000 -010	0000 -010	uuuu -uuu
SPBRG	0000 0000	0000 0000	uuuu uuuu
ANSEL	-111 1111	-111 1111	-111 1111
CMCON	0000 0111	0000 0111	uuuu u111
CVRCON	000- 0000	000- 0000	uuu- uuuu
WDTCON	---0 1000	---0 1000	--uu uuuu
ADRESL	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON1	0000 ----	0000 ----	uuuu ----
EEDATA	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATH	--xx xxxx	--uu uuuu	--uu uuuu
EEADRH	---- -xxx	---- -uuu	---- -uuu
EECON1	x--x x000	u--x u000	u--u uuuu
EECON2	-----	-----	-----

**Legend:** *u* = unchanged, *x* = unknown, *-* = unimplemented bit, read as '0', *q* = value depends on condition

**Note 1:** One or more bits in INTCON, PIR1 and PR2 will be affected (to cause wake-up).

**2:** When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

3: Ver la tabla 6.1 para los estados de reset específicos.

Tabla 6.2 Estados de inicialización para todos los registros

## 6.2 RESET MAESTRO #MCLR

El PIC 16F88 tiene un filtro de ruido en la entrada de reset #MCLR el cual detecta e ignora los pulsos pequeños. El fabricante recomienda que el pin RA5/#MCLR/V<sub>PP</sub> no se conecte directamente a V<sub>DD</sub>, sino que se lo haga de acuerdo al circuito de la figura 6.1. Este circuito se requiere únicamente si la fuente V<sub>DD</sub> alcanza su valor nominal de forma muy lenta durante el encendido.

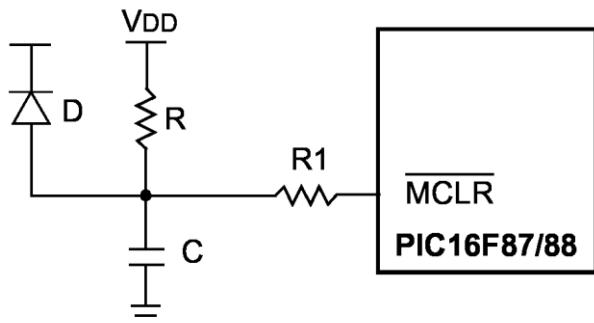


Figura 6.1 Circuito externo de reset al encendido.  $R < 40k\Omega$ ,  $1k\Omega < R_1 < 10k\Omega$ ,  $C = 0,1\mu F$

El pin RA5/#MCLR/V<sub>PP</sub> puede ser configurado para #MCLR (por defecto), o como pin E/S (RA5), por medio del bit MCLRE de la palabra CONFIG1.

## 6.3 RESET AL ENCENDIDO (Power-on Reset POR)

Se genera un pulso POR en el chip cuando se detecta un incremento de la fuente V<sub>DD</sub> (en el intervalo de 1,2 V a 1,7 V). Cuando se inicia la operación normal (el chip sale del reset), los parámetros de operación (voltaje, frecuencia, temperatura, etc.) tienen que cumplirse para asegurar el funcionamiento. Si estas condiciones no se cumplen, se debe mantener el Reset hasta que se reúnan las condiciones de operación.

## 6.4 TEMPORIZADOR DE ENCENDIDO (Power-up Timer PWRT)

Este temporizador emplea el oscilador INTRC como fuente de reloj. Proporciona un retardo fijo de 72 ms durante el encendido. Está diseñado para mantener al PIC en reset mientras la fuente de alimentación se estabiliza y puede ser habilitado o deshabilitado por medio del bit #PWRTE de la palabra CONFIG1.

## 6.5 TEMPORIZADOR DE ENCENDIDO DEL OSCILADOR (Oscillator Start-up Timer OST)

Proporciona un retardo de 1024 ciclos de oscilador (de la entrada RA7/OSC1) después de que el retardo PWRT ha concluido (si está habilitado). Esto asegura que el cristal o resonador ha arrancado y se ha estabilizado. Destinado a mantener el PIC en reset hasta que el oscilador de cristal se estabilice. Es invocado únicamente en los modos XT, LP y HS y únicamente en POR o al despertar (*wake-up from Sleep*).

## 6.6 RESET POR DESVANEZIMIENTO (Brown-out Reset BOR)

El bit BOREN de la palabra CONFIG1 puede habilitar o deshabilitar este reset. Si el voltaje V<sub>DD</sub> cae por debajo de 4V (aproximadamente) por más de 100 us (aproximadamente), entonces se produce el reset del chip. Si V<sub>DD</sub> cae por debajo de 4 V por un tiempo menor, no ocurre el reset. Una vez que se produce el reset, este se mantiene hasta que V<sub>DD</sub> supera los 4 V.

## 6.7 SECUENCIA GENERAL DE ENCENDIDO

El retardo PWRT (si está habilitado) comienza después de un POR. Entonces, OST empieza a contar 1024 ciclos de oscilador cuando PWRT finaliza (LP, XT, HS). Cuando OST finaliza, el dispositivo sale del reset.

Si #MCLR se mantiene en 0 el tiempo suficiente, todos los retardos finalizan. Al llevar #MCLR a 1 se iniciará la ejecución inmediatamente. Esto es útil para realizar pruebas, o para sincronizar más de un PIC en paralelo.

## 6.8 TEMPORIZADOR DE VIGILANCIA (Watchdog Timer WDT)

El propósito del WDT en un PIC es producir un reset cada cierto período de tiempo con lo cual se reinicia la ejecución del programa, con la finalidad de evitar que el dispositivo entre en un lazo infinito (se “cuelgue”) o se quede en una espera muy prolongada por un determinado evento que no ocurre. Durante la operación normal, el WDT genera un reset del dispositivo después del final de su período (*período WDT*). El reset puede evitarse si se reinicia el WDT por medio de la ejecución de la instrucción CLRWDT antes del final de su período. Si el dispositivo está en modo *Sleep*, el WDT ocasiona que se despierte el dispositivo y continúe con la operación normal (sin producir reset), esto se conoce como *despertar WDT*. El usuario tiene la posibilidad de emplear un prescaler para el WDT (prescaler de 16 bits) y para el Timer0 (prescaler de 8 bits) al mismo tiempo. El *período WDT* puede extenderse hasta 268 s, usando el prescaler de 16 bits y el *postscaler* (prescaler de 8 bits) simultáneamente, cuando el bit PSA del registro OPTION\_REG es igual a 1 (prescaler de 8 bits asignado al WDT).

◆ Un postscaler es un circuito que reduce la frecuencia de generación de interrupciones (o reset por WDT) de un temporizador/contador.

### 6.8.1 Oscilador WDT

El WDT obtiene su base de tiempo del oscilador INTRC de 31.25 kHz. El valor del registro WDTCON es ‘---0 1000’ (prescaler de 16 bits, 1:512) en todos los resets. Esto da una base nominal de  $\left(\frac{31.25\text{kHz}}{512}\right)^{-1} = 16,38\text{ms}$ , que es compatible con la base de tiempo generada en versiones anteriores de los PICs16.

◆ Cuando el OST es invocado, el WDT se mantiene en reset, debido a que el contador del WDT es empleado por el OST para realizar el conteo. Cuando finaliza el conteo OST, el WDT empezará a contar (si está habilitado).

Se ha añadido un nuevo prescaler para el WDT (figura 6.2). Este prescaler es de 16 bits y puede ser programado con los bits WDTPS<3:0> del registro WDTCON (figura 6.3), para dividir el oscilador RC para 32 hasta 65536, dándole a la base de tiempo del WDT un rango nominal de  $\left(\frac{31.25\text{kHz}}{32}\right)^{-1} = 1\text{ms}$  hasta  $\left(\frac{31.25\text{kHz}}{65536}\right)^{-1} = 2,097\text{s}$ .

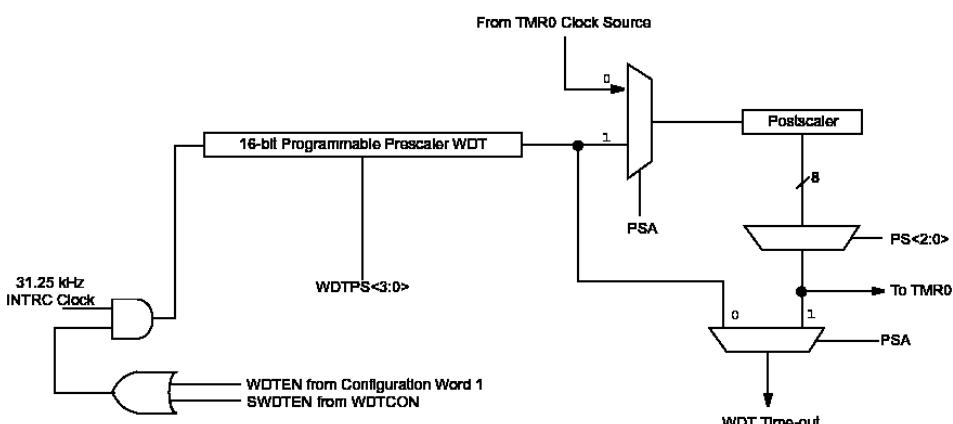


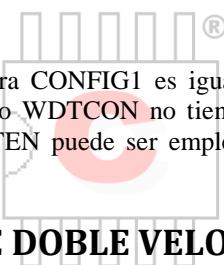
Figura 6.2 Diagrama de bloques del WDT (Postscaler es el prescaler de 8 bits del Timer0)

WDTCON: WATCHDOG CONTROL REGISTER (ADDRESS 105h)							
U-0	U-0	U-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0
			WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN <sup>(1)</sup>
bit 7							bit 0
bit 7-5	<b>Unimplemented: Read as '0'</b>						
bit 4-1	<b>WDTPS&lt;3:0&gt;: Watchdog Timer Period Select bits</b>						
	Bit	Prescale					
	Value	Rate					
	0000	= 1:32					
	0001	= 1:64					
	0010	= 1:128					
	0011	= 1:256					
	0100	= 1:512					
	0101	= 1:1024					
	0110	= 1:2048					
	0111	= 1:4096					
	1000	= 1:8192					
	1001	= 1:16384					
	1010	= 1:32768					
	1011	= 1:65536					
bit 0	<b>SWDTEN: Software Enable/Disable for Watchdog Timer bit<sup>(1)</sup></b>						
	1	= WDT is turned on					
	0	= WDT is turned off					
	<b>Note 1:</b> If WDTE configuration bit = 1, then WDT is always enabled, irrespective of this control bit. If WDTE configuration bit = 0, then it is possible to turn WDT on/off with this control bit.						

Figura 6.3 Bits del registro WDTCON

## 6.8.2 Control del WDT

Cuando el bit WDTE de la palabra CONFIG1 es igual a 1 (WDT habilitado), el WDT funciona continuamente y el bit SWDTEN del registro WDTCON no tiene ningún efecto. Si se borra el bit WDTE (WDT deshabilitado), entonces el bit SWDTEN puede ser empleado para encender y apagar el WDT en el instante deseado de forma muy precisa.



## 6.9 MODO DE ENCENDIDO DE DOBLE VELOCIDAD

Este modo minimiza el retardo entre el encendido del oscilador y el comienzo de la ejecución del código y puede ser seleccionado con el bit IESO de la palabra CONFIG2. Esto se consigue empleando inicialmente el oscilador INTRC para la ejecución del código hasta que el oscilador primario se estabilice.

www.programarpicenc.com  
Chesca Pineda de Leon, Plazas

Si este modo está habilitado y cualquiera de las siguientes condiciones existe, el sistema comenzará la ejecución con el oscilador INTRC. Como resultado se tiene que la ejecución comienza casi de inmediato con un retardo mínimo:

- POR y después de PWRT (si está habilitado).
- O después de un despertar.
- O un reset cuando está funcionando con el oscilador T1OSC o INTRC (después de un reset, SCS<1:0> siempre están en '00').

◆Después de cualquier reset, los bits IRCF están en 0 y la selección de frecuencia se ve forzada a 31.25 kHz. El usuario puede modificar los bits IRCF para seleccionar una frecuencia mayor del oscilador interno.

Si el oscilador primario es distinto de cualquiera de los modos XT, LP o HS, entonces el encendido de doble velocidad está deshabilitado debido a que el oscilador primario no requiere tiempo alguno para estabilizarse luego de POR, o al despertar.

Si los bits IRCF del registro OSCCON se han configurado con un valor distinto de cero antes de ingresar en el modo *Sleep*, el reloj del sistema provendrá de INTOSC. El bit IOFS del registro OSCCON permanecerá en 0 hasta que INTOSC se estabilice. Esto permite que el usuario pueda determinar cuándo emplear el oscilador interno para aplicaciones críticas.

Al revisar el estado del bit OSTS del registro OSCCON se puede confirmar si el oscilador primario está listo, de lo contrario este bit permanece en 0.

Cuando el dispositivo es autoconfigurado en el modo INT RC después de POR o despertar, las reglas para seleccionar otros modos de oscilador aún se aplican, lo que quiere decir que los bits SCS<1:0> del registro OSCCON pueden ser modificados antes de que finalice el período OST. Esto permitiría a la aplicación despertar, ejecutar unas pocas instrucciones empleando el oscilador INT RC y retornar al modo *Sleep* sin tener que esperar a que el oscilador primario se estabilice.

- ◆ La ejecución de una instrucción SLEEP cancela el período OST y ocasiona que el bit OSTS permanezca en 0.

### 6.9.1 Secuencia de encendido en doble velocidad

1. Despertar, Reset o POR.
2. Bits del registro OSCCON configurados para INT RC (31.25 kHz).
3. Comienza la ejecución de las instrucciones.
4. OST cuenta 1024 ciclos de oscilador.
5. Finaliza OST, espera por transición descendente de INT RC.
6. OSTS=1.
7. El reloj del sistema permanece en 0 durante 8 transiciones descendentes del nuevo reloj (LP, XT o HS).
8. Se produce la conmutación a la fuente primaria de reloj (LP, XT o HS).

El software puede leer el bit OSTS para determinar cuándo se produce la conmutación, de tal forma que se puedan realizar los ajustes necesarios en la temporización.

## 6.10 OSCILADOR DE SEGURIDAD (Fail-safe Clock Monitor FSCM)

Está diseñado para permitir que el dispositivo continúe su operación en el caso de una falla en el oscilador. Esta función es habilitada con el bit FCME en la palabra CONFIG2.

Si falla el oscilador, FSCM conmuta el reloj del sistema al oscilador interno. El sistema continúa operando con el oscilador interno hasta que finalice la condición de falla. La condición de falla finaliza con un reset, la ejecución de una instrucción SLEEP o la escritura en el registro OSCCON.

La frecuencia del oscilador interno dependerá de los bits IRCF. Puede seleccionarse otra fuente de reloj por medio de los bits IRCF y SCS del registro OSCCON.

Mientras el dispositivo está en este modo, un reset producirá la finalización del modo de seguridad. Si el reloj primario está configurado para un cristal, el dispositivo continúa la ejecución con el oscilador interno hasta que finaliza el período OST. Se puede ejecutar una instrucción SLEEP o una escritura en los bits SCS (diferente de '00'), para colocar el chip en un modo de baja potencia.

- ◆ El modo de doble velocidad se habilita automáticamente cuando se habilita el oscilador de seguridad.

Si se produce un reset en el modo de oscilador de seguridad y la fuente primaria de reloj es EC o RC, entonces el dispositivo será conmutado inmediatamente al modo EC o RC.

## 6.11 MODO DE BAJO CONSUMO (Sleep)

Se ingresa a este modo por la ejecución de la instrucción SLEEP. Si está habilitado, el WDT es reiniciado y se mantiene en funcionamiento, el bit #PD del registro STATUS se borra, el bit #TO del registro STATUS se pone en 1 y el oscilador se apaga. Los puertos E/S mantienen el estado que tenían antes de la instrucción SLEEP. El pin #MCLR tiene que estar en nivel alto.

### 6.11.1 Despertar (Wake-up from Sleep)

Se puede producir un despertar por cualquiera de los siguientes eventos:

1. Reset #MCLR externo.
2. Despertar por WDT (si está habilitado).
3. Interrupción del pin RB0/INT, interrupción RB (variación en la entrada de los pines PORTB<7:4>) o una interrupción de cualquier periférico.

El reset #MCLR ocasiona un reset del dispositivo. El resto de eventos se consideran como una continuación de la ejecución del programa y producen un despertar. Los bits #TO y #PD en el registro STATUS

pueden emplearse para determinar la causa del reset. El bit #PD, que está en 1 al encendido, pasa a 0 cuando se ingresa al modo *Sleep*. El bit #TO pasa a 0 si finaliza el período WDT y produce un despertar.

Las interrupciones de los siguientes periféricos pueden producir un despertar:

1. Interrupción del Timer1 (tiene que estar operando como contador asincrónico).
2. Interrupción en modo de captura del módulo CCP.
3. Detección de un evento especial (Timer1 en modo asincrónico con reloj externo).
4. Interrupción del módulo SSP por detección de bit de parada/arranque.
5. Recepción/transmisión del módulo SSP en modo esclavo (SPI/I<sup>2</sup>C).
6. Conversión A/D (cuando la fuente de reloj A/D es RC).
7. Al completar una operación de escritura EEPROM.
8. La salida del comparador cambia de estado.
9. Recepción/transmisión USART (modo esclavo sincrónico).

Cuando se ejecuta la instrucción SLEEP, la siguiente instrucción (PC+1) es *capturada*. Para que se produzca un despertar debido a una interrupción, la interrupción debe estar habilitada. El despertar ocurre sin importar el estado del bit GIE del registro INTCON. Si GIE=0 (deshabilitado), la ejecución continúa con la siguiente instrucción ubicada inmediatamente después de SLEEP. Si GIE=1 (habilitado), el dispositivo ejecuta la siguiente instrucción ubicada inmediatamente después de SLEEP y luego salta a la dirección de interrupción (0x0004).

### 6.11.2 Despertar usando interrupciones

Cuando las interrupciones globales están deshabilitadas (GIE=0) y cualquier fuente de interrupción tiene sus bits de habilitación y bandera en un valor igual a 1, una de las siguientes posibilidades ocurrirá:

- Si la interrupción ocurre antes de la ejecución de una instrucción SLEEP, ésta se completará como una instrucción NOP. Por lo tanto, el WDT y el prescaler y postscaler del WDT (si está habilitado) no serán borrados, el bit #TO no se pondrá en 1 y el bit #PD no será borrado.
- Si la interrupción ocurre durante o después de la ejecución de una instrucción SLEEP, el dispositivo se despertará inmediatamente. La instrucción SLEEP se ejecutará completamente antes del despertar. Por lo tanto, el WDT y el prescaler y postscaler del WDT (si está habilitado) serán borrados, el bit #TO se pondrá en 1 y el bit #PD será borrado.

## PIC16F628A y 16F877A



Este dispositivo tiene un conjunto de características destinadas a maximizar la confiabilidad, minimizar el costo por medio de la eliminación de componentes externos, proporcionar ahorro de energía y protección del código.

### 6.12 RESET

Los PICs 16F628A y 16F877A diferencian varias clases de reset:

- Reset al encendido (POR).
- Reset #MCLR durante operación normal.
- Reset #MCLR durante *Sleep*.
- Reset por WDT durante operación normal.
- Despertar por WDT durante *Sleep*.
- Reset por Desvanecimiento (BOR).

Algunos registros no se ven afectados por un reset de cualquier tipo. Su estado es desconocido en POR y no cambia en cualquier otro reset. La mayoría de los otros registros son restablecidos a un *estado de reset* en POR, reset #MCLR y WDT, reset #MCLR durante *Sleep* y BOR. No son afectados por un despertar por WDT, que es visto como una reanudación de la operación normal. Los bits #TO y #PD del registro STATUS se emplean en *software* para determinar la naturaleza del reset. En las tablas 6.3, 6.4.1 y 6.4.2 se describen los estados de reset de todos los registros. El reset #MCLR tiene un filtro de ruido para detectar e ignorar los pulsos pequeños.

Condition	Program Counter	Status Register	PCON Register
Power-on Reset	000h	0001 xxxx	---- 1-0x
MCLR Reset during normal operation	000h	000u uuuu	---- 1-uu
MCLR Reset during Sleep	000h	0001 0uuu	---- 1-uu
WDT Reset	000h	0000 uuuu	---- 1-uu
WDT Wake-up	PC + 1	uuu0 0uuu	---- u-uu
Brown-out Reset	000h	000x xuuu	---- 1-u0
Interrupt Wake-up from Sleep	PC + 1 <sup>(1)</sup>	uuu1 0uuu	---- u-uu

Legend: u = unchanged, x = unknown, - = unimplemented bit, reads as '0'.

Note 1: When the wake-up is due to an interrupt and global enable bit, GIE is set, the PC is loaded with the interrupt vector (0004h) after execution of PC + 1.

Tabla 6.3 Estados de reset para registros especiales

Register	Address	Power-on Reset	• MCLR Reset during normal operation • MCLR Reset during Sleep • WDT Reset • Brown-out Reset <sup>(1)</sup>	• Wake-up from Sleep <sup>(7)</sup> through interrupt • Wake-up from Sleep <sup>(7)</sup> through WDT time out
W	—	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
INDF	00h, 80h, 100h, 180h	—	—	—
TMR0	01h, 101h	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
PCL	02h, 82h, 102h, 182h	0000 0000	0000 0000	PC + 1 <sup>(3)</sup>
STATUS	03h, 83h, 103h, 183h	0001 1xxx	000q quuu <sup>(4)</sup>	uuuq 0uuu <sup>(4)</sup>
FSR	04h, 84h, 104h, 184h	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
PORTA	05h	xxxx 0000	xxxx 0000	uuuuuu uuuuuu
PORTB	06h, 106h	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
PCLATH	0Ah, 8Ah, 10Ah, 18Ah	--0 0000	--0 0000	--u uuuuu
INTCON	0Bh, 8Bh, 10Bh, 18Bh	0000 000x	0000 000u	uuuuu uqqq <sup>(2)</sup>
PIR1	0Ch	0000 -000	0000 -000	qqqq -qqqq <sup>(2)</sup>
TMR1L	0Eh	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
TMR1H	0Fh	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
T1CON	10h	--00 0000	--uu uuuuu <sup>(6)</sup>	--uu uuuuu
TMR2	11h	0000 0000	0000 0000	uuuuuu uuuuuu
T2CON	12h	-000 0000	-000 0000	-uuuu uuuuu
CCPR1L	15h	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
CCPR1H	16h	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
CCP1CON	17h	--00 0000	--00 0000	--uu uuuuu
RCSTA	18h	0000 000x	0000 000x	uuuuuu uuuuuu
TXREG	19h	0000 0000	0000 0000	uuuuuu uuuuuu
RCREG	1Ah	0000 0000	0000 0000	uuuuuu uuuuuu
CΜCON	1Fh	0000 0000	0000 0000	uu-- uuuuu
OPTION	81h, 181h	1111 1111	1111 1111	uuuuuu uuuuuu
TRISA	85h	1111 1111	1111 1111	uuuuuu uuuuuu
TRISB	86h, 186h	1111 1111	1111 1111	uuuuuu uuuuuu
PIE1	8Ch	0000 -000	0000 -000	uuuuuu -uuuu
PCON	8Eh	---- 1-0x	---- 1-uq <sup>(1,5)</sup>	---- u-uu
PR2	92h	1111 1111	1111 1111	uuuu uuuuu
TXSTA	98h	0000 -010	0000 -010	uuuu -uuuu
SPBRG	99h	0000 0000	0000 0000	uuuu uuuuu
EEDATA	9Ah	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
EEADR	9Bh	xxxx xxxx	uuuuuu uuuuuu	uuuuuu uuuuuu
EECON1	9Ch	---- x000	---- q000	---- uuuuu
EECON2	9Dh	—	—	—
VRCON	9Fh	000- 0000	000- 0000	uuuu- uuuuu

**Legend:** u = unchanged, x = unknown, - = unimplemented bit, reads as '0', q = value depends on condition.

**Note 1:** If VDD goes too low, Power-on Reset will be activated and registers will be affected differently.

2: One or more bits in INTCON and PIR1 will be affected (to cause wake-up).

3: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

4: Ver la tabla 6.3 para los estados de reset específicos.

5: If Reset was due to brown-out, then bit 0 = 0. All other Resets will cause bit 0 = u.

6: Reset to '1-00 0000' on a Brown-out Reset (BOR).

7: Peripherals generating interrupts for wake-up from Sleep will change the resulting bits in the associated registers.

Tabla 6.4.1 Estados de inicialización para todos los registros (PIC16F628A)

Register	Devices				Power-on Reset, Brown-out Reset	MCLR Reset, WDT Reset	Wake-up via WDT or Interrupt
W	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
INDF	73A	74A	76A	77A	N/A	N/A	N/A
TMR0	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
PCL	73A	74A	76A	77A	0000 0000	0000 0000	PC + 1 <sup>(2)</sup>
STATUS	73A	74A	76A	77A	0001 1xxx	000q qnnn <sup>(3)</sup>	nnnq qnnn <sup>(3)</sup>
FSR	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
PORTA	73A	74A	76A	77A	--0x 0000	--0u 0000	--nu nnnn
PORTB	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
PORTC	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
PORTD	74A			77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
PORTE		74A			---- -xxx	---- -nnn	---- -nnn
PCLATH	73A	74A	76A	77A	--0 0000	--0 0000	--nu nnnn
INTCON	73A	74A	76A	77A	0000 000x	0000 000u	nnnn nnnn <sup>(1)</sup>
PIR1	73A		76A		r000 0000	r000 0000	nnnn nnnn <sup>(1)</sup>
		74A		77A	0000 0000	0000 0000	nnnn nnnn <sup>(1)</sup>
PIR2	73A	74A	76A	77A	-0-0 0--0	-0-0 0--0	-u-u u--u <sup>(1)</sup>
TMR1L	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
TMR1H	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
T1CON	73A	74A	76A	77A	--00 0000	--nu nnnn	--nu nnnn
TMR2	73A	74A	76A	77A	0000 0000	0000 0000	nnnn nnnn
T2CON	73A	74A	76A	77A	-000 0000	-000 0000	-nnn nnnn
SSPBUF	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
SSPCON	73A	74A	76A	77A	0000 0000	0000 0000	nnnn nnnn
CCPR1L	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
CCPR1H	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
CCP1CON	73A	74A	76A	77A	--00 0000	--00 0000	--nu nnnn
RCSTA	73A	74A	76A	77A	0000 000x	0000 000x	nnnn nnnn
TXREG	73A	74A	76A	77A	0000 0000	0000 0000	nnnn nnnn
RCREG	73A	74A	76A	77A	0000 0000	0000 0000	nnnn nnnn
CCPR2L	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
CCPR2H	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
CCP2CON	73A	74A	76A	77A	0000 0000	0000 0000	nnnn nnnn
ADRESH	73A	74A	76A	77A	xxxx xxxx	nnnn nnnn	nnnn nnnn
ADCON0	73A	74A	76A	77A	0000 00-0	0000 00-0	nnnn nu-n
OPTION_REG	73A	74A	76A	77A	1111 1111	1111 1111	nnnn nnnn
TRISA	73A	74A	76A	77A	--11 1111	--11 1111	--nu nnnn
TRISB	73A	74A	76A	77A	1111 1111	1111 1111	nnnn nnnn
TRISC	73A	74A	76A	77A	1111 1111	1111 1111	nnnn nnnn

Register	Devices			Power-on Reset, Brown-out Reset	MCLR Reset, WDT Reset	Wake-up via WDT or Interrupt	
TRISD	74A		77A	1111 1111	1111 1111	uuuu uuuu	
TRISE	74A		77A	0000 -111	0000 -111	uuuu -uuu	
PIE1	73A	74A	76A	r000 0000	r000 0000	ruuu uuuu	
	74A		77A	0000 0000	0000 0000	uuuu uuuu	
PIE2	73A	74A	76A	77A	-0-0 0--0	-0-0 0--0	-u-u u--u
PCON	73A	74A	76A	77A	---- --qq	---- --uu	---- --uu
SSPCON2	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
PR2	73A	74A	76A	77A	1111 1111	1111 1111	1111 1111
SSPADD	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	73A	74A	76A	77A	--00 0000	--00 0000	--uu uuuu
TXSTA	73A	74A	76A	77A	0000 -010	0000 -010	uuuu -uuu
SPBRG	73A	74A	76A	77A	0000 0000	0000 0000	uuuu uuuu
CMDCON	73A	974	76A	77A	0000 0111	0000 0111	uuuu uuuu
CVRCON	73A	74A	76A	77A	000- 0000	000- 0000	uuu- uuuu
ADRESL	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON1	73A	74A	76A	77A	00-- 0000	00-- 0000	uu- - uuuu
EEDATA	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADR	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEDATH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EEADDRH	73A	74A	76A	77A	xxxx xxxx	uuuu uuuu	uuuu uuuu
EECON1	73A	74A	76A	77A	x--- x000	u--- u000	u--- uuuu
EECON2	73A	74A	76A	77A	-----	-----	-----

**Legend:** u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition, r = reserved, maintain clear. Shaded cells indicate conditions do not apply for the designated device.

- Note 1:** One or more bits in INTCON, PIR1 and/or PIR2 will be affected (to cause wake-up).
- 2:** When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).
- 3:** Ver la tabla 6.3 para los estados de reset específicos.

Tabla 6.4.2 Estados de inicialización para todos los registros (PIC16F877A)

## 6.13 RESET AL ENCENDIDO (Power-on Reset POR)

El circuito POR en el chip mantiene el dispositivo en reset hasta que se detecta un incremento de la fuente V<sub>DD</sub> (en el intervalo de 1,2 V a 1,7 V). Este circuito no produce reset cuando el voltaje V<sub>DD</sub> disminuye. Cuando se inicia la operación normal (el chip sale del reset), los parámetros de operación (voltaje, frecuencia, temperatura, etc.) tienen que cumplirse para asegurar el funcionamiento. Si estas condiciones no se cumplen, se debe mantener el reset por medio de #MCLR, BOR o PWRT hasta que se cumplan las condiciones de operación.

## 6.14 TEMPORIZADOR DE ENCENDIDO (Power-up Timer PWRT)

Este temporizador emplea un oscilador interno RC. Proporciona un retardo fijo de 72 ms luego del POR o BOR (si está habilitado). Está diseñado para mantener al PIC en reset mientras la fuente de alimentación alcanza un nivel aceptable y puede ser habilitado o deshabilitado por medio del bit #PWRTE de la palabra de configuración CONFIG. Se recomienda habilitarlo cuando también se ha habilitado el reset por desvanecimiento BOR.

## 6.15 TEMPORIZADOR DE ENCENDIDO DEL OSCILADOR (Oscillator Start-up Timer OST)

Proporciona un retardo de 1024 ciclos de oscilador (de la entrada OSC1) después de que el retardo PWRT ha concluido (si está habilitado). Esto asegura que el cristal o resonador ha arrancado y se ha estabilizado. Destinado a mantener el PIC en reset hasta que el oscilador de cristal se estabilice. Es invocado únicamente en los modos XT, LP y HS y únicamente en POR o al despertar (*wake-up from Sleep*).

## 6.16 RESET POR DESVANECIMIENTO (Brown-out Reset BOR)

El bit BOREN de la palabra CONFIG puede habilitar o deshabilitar este reset. Si el voltaje  $V_{DD}$  cae por debajo de  $V_{BOR}$  durante un tiempo mayor que  $T_{BOR}$ , entonces se produce el reset del chip. Si  $V_{DD}$  cae por debajo de  $V_{BOR}$  por un tiempo menor, no ocurre el reset. En cualquier reset (POR, BOR, WDT, etc.), el chip se mantiene en reset hasta que  $V_{DD}$  alcanza un nivel mayor que  $V_{BOR}$ . A continuación entra en acción el PWRT (si está habilitado) y mantiene el reset por 72 ms más.

Si  $V_{DD}$  cae por debajo de  $V_{BOR}$  mientras está en ejecución PWRT, el chip retornará al reset BOR y el PWRT será reiniciado. Una vez que  $V_{DD}$  supera  $V_{BOR}$ , el PWRT mantendrá el reset por 72 ms. La figura 6.4 muestra las situaciones típicas de desvanecimiento.

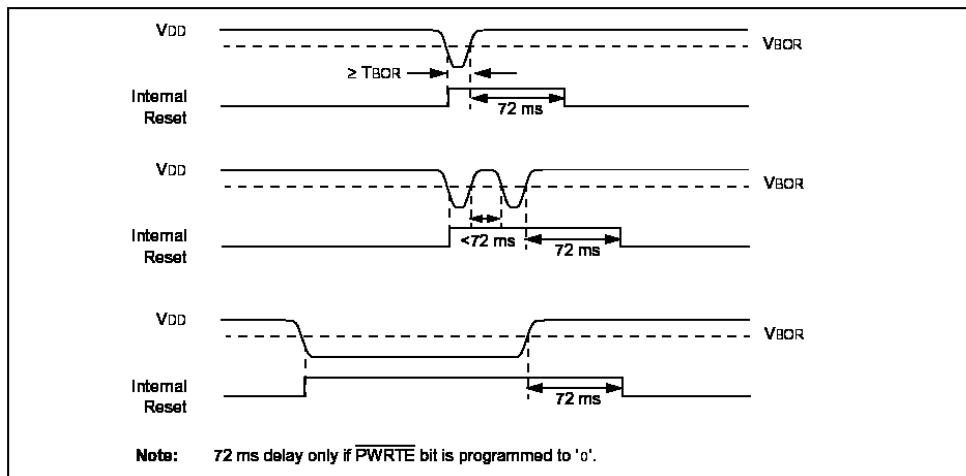


Figura 6.4 Situaciones de desvanecimiento con PWRT habilitado

## 6.17 SECUENCIA GENERAL DE ENCENDIDO

Primero se activa el reset POR, y a continuación el retardo PWRT (si está habilitado). Entonces se activa OST. Cuando OST finaliza, el dispositivo sale del reset. El tiempo total de encendido dependerá de la configuración del oscilador y del estado del bit #PWRTE. Las figuras 6.5, 6.6, 6.7 y 6.8 esquematizan las secuencias de encendido en diferentes condiciones.

Dado que la temporización se mide desde el pulso POR, si #MCLR se mantiene en 0 el tiempo suficiente, todos los retardos finalizan. Al llevar #MCLR a 1 se iniciará la ejecución inmediatamente (figura 6.6). Esto es útil para realizar pruebas, o para sincronizar más de un PIC en paralelo.

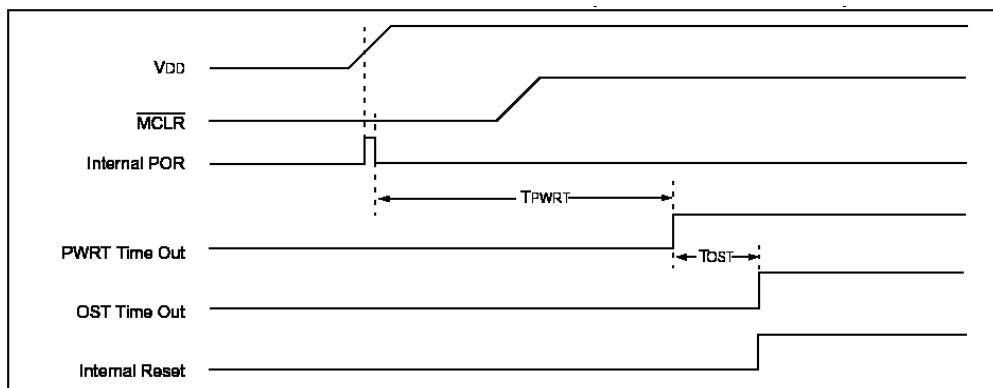
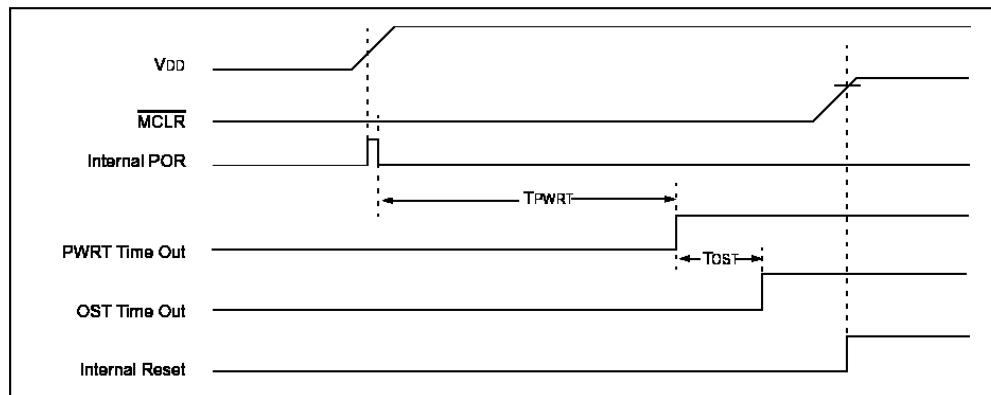
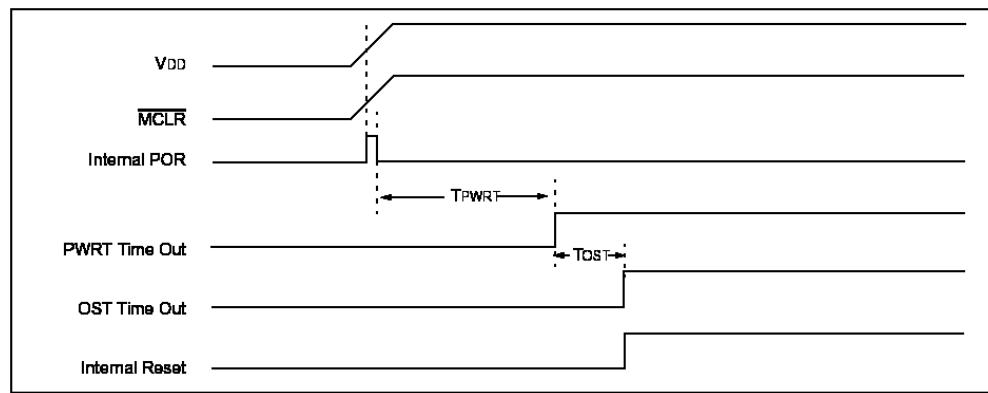
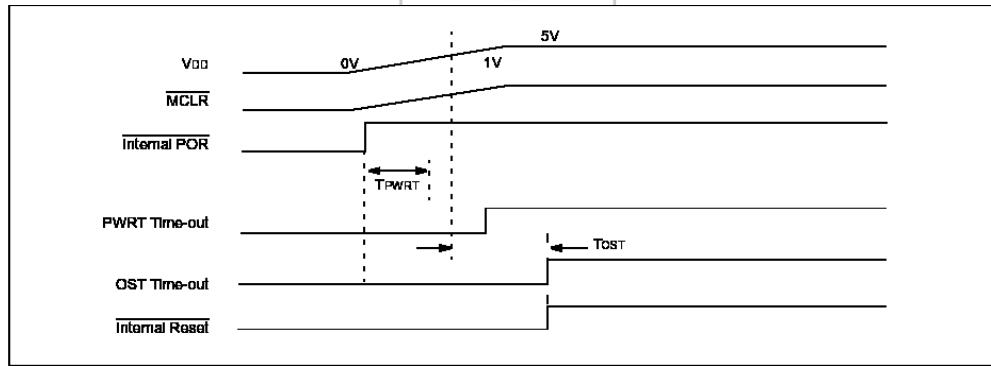


Figura 6.5 Secuencia de encendido (#MCLR desconectado de  $V_{DD}$ ). Caso 1

Figura 6.6 Secuencia de encendido (#MCLR desconectado de V<sub>DD</sub>). Caso 2Figura 6.7 Secuencia de encendido (#MCLR conectado a V<sub>DD</sub>)Figura 6.8 Secuencia de encendido (#MCLR conectado a V<sub>DD</sub> por medio de una red RC)

## 6.18 TEMPORIZADOR DE VIGILANCIA (Watchdog Timer WDT)

El propósito del WDT en un PIC es producir un reset cada cierto período de tiempo con lo cual se reinicia la ejecución del programa, con la finalidad de evitar que el dispositivo entre en un lazo infinito (se “cuelgue”) o se quede en una espera muy prolongada por un determinado evento que no ocurre. El WDT opera en base a un oscilador interno RC (de uso exclusivo) que no requiere de la adición de componentes externos. Durante la operación normal, el WDT genera un reset del dispositivo después del final de su período (*período WDT*). Si el dispositivo está en modo *Sleep*, el WDT ocasiona que se despierte el dispositivo y continúe con la operación normal (sin producir reset), esto se conoce como *despertar WDT*. El WDT puede habilitarse por medio del bit WDTE de la palabra de configuración CONFIG.

El WDT (figura 6.9) tiene un período nominal de 18 ms (sin prescaler). Si se desean períodos mayores se puede emplear un postscaler de hasta 1:128, por medio de la programación adecuada de los bits PSA y PS<2:0> del registro OPTION\_REG. Con esto se pueden lograr períodos de hasta 2,3 s.

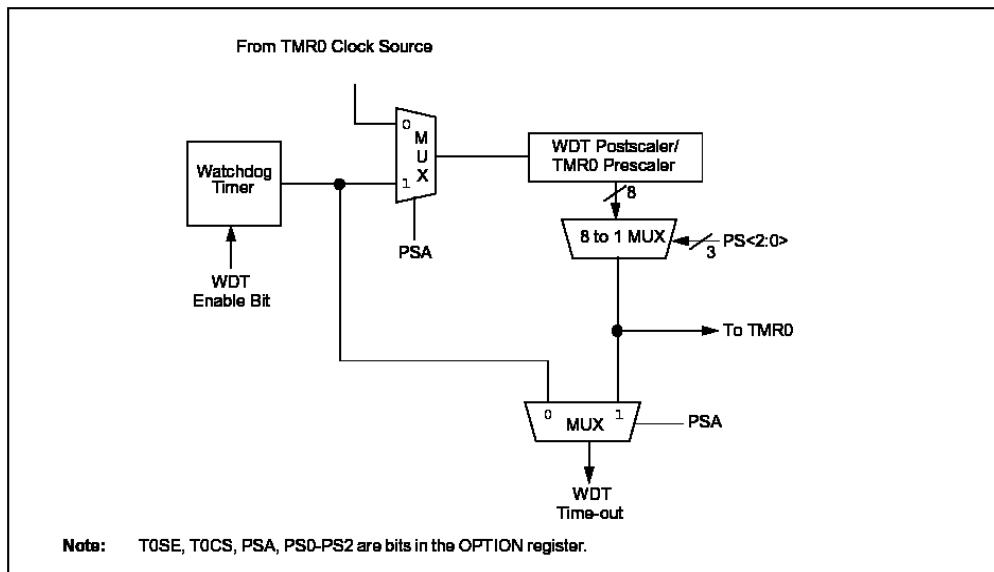


Figura 6.9 Diagrama de bloques del WDT

Las instrucciones CLRWDY y SLEEP reinician el WDT y el postscaler (si está asignado al WDT) y evitan la generación del reset.

◆ Un postscaler es un circuito que reduce la frecuencia de generación de interrupciones (o reset por WDT) de un temporizador/contador.

## 6.19 MODO DE BAJO CONSUMO (Sleep)

Se ingresa a este modo por la ejecución de la instrucción SLEEP. Si está habilitado, el WDT es reiniciado y se mantiene en funcionamiento, el bit #PD del registro STATUS se borra, el bit #TO del registro STATUS se pone en 1 y el oscilador se apaga. Los puertos E/S mantienen el estado que tenían antes de la instrucción SLEEP. El pin #MCLR tiene que estar en nivel alto.

### 6.19.1 Despertar (Wake-up from Sleep)

Se puede producir un despertar por cualquiera de los siguientes eventos:

1. Reset #MCLR externo.
2. Despertar por WDT (si está habilitado).
3. Interrupción del pin RB0/INT, interrupción RB (variación en la entrada de los pines PORTB<7:4>) o una interrupción de cualquier periférico.

El reset #MCLR ocasiona un reset del dispositivo. El resto de eventos se consideran como una continuación de la ejecución del programa y producen un despertar. Los bits #TO y #PD en el registro STATUS pueden emplearse para determinar la causa del reset. El bit #PD, que está en 1 al encendido, pasa a 0 cuando se ingresa al modo *Sleep*. El bit #TO pasa a 0 si finaliza el período WDT y produce un despertar.

Cuando se ejecuta la instrucción SLEEP, la siguiente instrucción (PC+1) es *capturada*. Para que se produzca un despertar debido a una interrupción, la interrupción debe estar habilitada. El despertar ocurre sin importar el estado del bit GIE del registro INTCON. Si GIE=0 (deshabilitado), la ejecución continúa con la siguiente instrucción ubicada inmediatamente después de SLEEP. Si GIE=1 (habilitado), el dispositivo ejecuta la siguiente instrucción ubicada inmediatamente después de SLEEP y luego salta a la dirección de interrupción (0x0004). Si las interrupciones globales están deshabilitadas (GIE=0), pero cualquier fuente de interrupción tiene sus bits de habilitación y bandera de interrupción en 1, el dispositivo no entrará en modo *Sleep*. La instrucción SLEEP se ejecuta como una instrucción NOP. El WDT es reiniciado cuando el dispositivo se despierta, sin importar la causa del despertar.

## 6.20 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

**(16F88)** Este primer ejemplo muestra cómo programar el WDT para que produzca un reset cada cierto tiempo. Debe notarse que el prescaler de 16 bits permanece en su valor inicial (1:512), por lo que la base de tiempo es de 16,38 ms, mientras que el prescaler de 8 bits (postscaler) se programa con un valor de 1:32, lo que da un *período WDT* =  $16,38 \times 32 = 524,16$  ms (aproximadamente 0,5 s). Otra característica importante de este ejemplo es cómo se puede añadir código ensamblador por medio de la declaración *asm*. El WDT permanece deshabilitado en la palabra CONFIG1 (WDTEN=0). Revisar por medio del comando *Project>Edit Project* y seleccionar *Watchdog Timer: Off*. Así el encendido/apagado del WDT se realiza de forma precisa por medio del bit SWDTEN del registro WDTCON.

**(16F628A y 16F877A)** El prescaler está asignado inicialmente al WDT y se lo ha programado con un valor de 1:32 lo que da un *período WDT*= $18 \times 32=576$  ms (aproximadamente 0,5 s). El WDT se debe encender por medio del bit WDTE de la palabra de configuración CONFIG.

**WDT1.c:** El PIC se pone en modo de bajo consumo. El despertar se producirá cada vez que el WDT concluya su período, en ese momento se producirá un incremento de un contador que se visualizará en la pantalla y nuevamente volverá al modo de bajo consumo. El proceso debe repetirse cada medio segundo aproximadamente (circuito de la figura 3.2). *Conversions*  *LCD* .

```

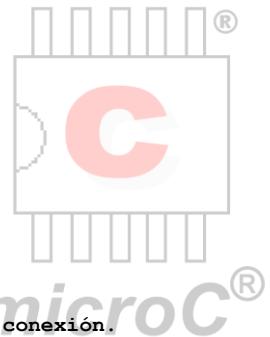

//WDT1.c
//OPTION_REG tiene todos sus bits en 1 después de cualquier
//reset y no cambia al despertar por WDT o interrupción, por lo tanto
//el Timer0 actúa como contador, incrementa en transición
//descendente y el prescaler está asignado al WDT.

//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char texto[4], contador=0;
void main() {
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
WDTCON=0x08; //Prescaler 1:512 (Base de tiempo 16,38 ms).
OPTION_REG=0xFD; //Postscaler 1:32 (Período WDT = 524,16 ms).
SWDTEN_bit=1; //WDT encendido.
while (1)
{
ByteToStr(contador, texto);
Lcd_Out(1,6,texto);
asm SLEEP //Modo de bajo consumo durante 524.16 ms.
contador++;
}
}

```



**microC®**

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

En el ejemplo **WDT2.c**, al encender el PIC el bit #TO del registro STATUS es igual a 1, por lo tanto la ejecución continúa a partir de //Reset por otra causa. Si no se presiona el pulsador, se llega al final del período WDT, lo que produce el reset (#TO=0) y se ingresa en la ejecución de las instrucciones a partir de //Reset por WDT. Aquí se presenta el mensaje y luego el PIC ingresa en el modo *Sleep*, del cual sale debido a un despertar por WDT y continúa sin producir reset.

Si se presiona el pulsador, la instrucción CLRWDTH reinicia el WDT e impide el reset.

**WDT2.c:** Cada vez que se presione el pulsador conectado en RA4, se incrementará un contador que se visualizará en el LCD. Si tarda más de 1 segundo en pulsar, el WDT se activará y en la pantalla aparecerá un mensaje (circuito de las figuras 3.1.1 y 3.1.2). *Conversions*  *LCD*  *Button* .

```


//WDT2.c
//OPTION_REG tiene todos sus bits en 1 después de cualquier
//reset y no cambia al despertar por WDT o interrupción, por lo tanto
//el Timer0 actúa como contador, incrementa en transición
//descendente y el prescaler está asignado al WDT.

//Variables de conexión 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;
// Fin de declaración de variables de conexión.

char texto[4], contador=0, estado=1;

void main()
{
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFF==0); //Esperar mientras el oscilador está inestable.
    ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
    Lcd_Init(); //Inicializa el LCD.
    Lcd_Cmd(_LCD_CLEAR); //Borra el display.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
    WDTCON=0x08; //Prescaler 1:512 (Base de tiempo 16,38 ms).
    OPTION_REG=0xFE; //Postscaler 1:64 (Período WDT = 1048,32 ms).

    //Reset por WDT.
    if (NOT_TO_bit==0)
    {
        while (1)
        {
            Lcd_Out(1,1,"Eres LENTO");
            asm SLEEP //Modo de bajo consumo durante 1048,32 ms.
        }
    }

    //Reset por otra causa.
    ByteToStr(contador,texto);
    Lcd_Out(1,1,texto);
    Lcd_Out_CP(" pulsaciones");
    SWDTEN_bit=1; //WDT encendido.
    while (1)
    {
        if (Button(&PORTA,4,1,0)) //Si se pulsa.
        {
            estado=0;
            asm CLRWDAT //Pulsador cerrado: Reiniciar WDT.
        }

        if (estado==0 && Button(&PORTA,4,1,1)) //Si se pulsa y se libera.
        {
            contador++;
            if (contador>100) contador=0;
            ByteToStr(contador,texto);
            Lcd_Out(1,1,texto);
            Lcd_Out_CP(" pulsaciones");
            estado=1;
        }
    }
}

```



www.programarpicenc.com

©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO VII: INTERRUPCIONES

Una interrupción, como el nombre lo sugiere, es un evento que hace que el microcontrolador deje de realizar lo que está haciendo y pase a ejecutar otra tarea. Al finalizar retorna a su actividad inicial.

El PIC16F88 tiene hasta 12 fuentes de interrupción, el PIC16F628A tiene 10 y el PIC16F877A tiene 15. El registro INTCON (figuras 7.1.1 y 7.1.2) contiene las banderas de interrupción generadas por diferentes eventos. También contiene los bits de habilitación global y particular de las distintas fuentes de interrupción (observar las diferencias en algunos bits de este registro a lo largo de los siguientes temas).

INTCON: INTERRUPT CONTROL REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
<b>GIE</b>	<b>PEIE</b>	<b>TMROIE</b>	<b>INT0IE</b>	<b>RBIE</b>	<b>TMROIF</b>	<b>INT0IF</b>	<b>RBIF</b>
bit 7							bit 0
<b>bit 7</b>	<b>GIE: Global Interrupt Enable bit</b> 1 = Enables all unmasked interrupts 0 = Disables all interrupts						
<b>bit 6</b>	<b>PEIE: Peripheral Interrupt Enable bit</b> 1 = Enables all unmasked peripheral interrupts 0 = Disables all peripheral interrupts						
<b>bit 5</b>	<b>TMROIE: TMRO Overflow Interrupt Enable bit</b> 1 = Enables the TMRO interrupt 0 = Disables the TMRO interrupt						
<b>bit 4</b>	<b>INT0IE: RB0/INT External Interrupt Enable bit</b> 1 = Enables the RB0/INT external interrupt 0 = Disables the RB0/INT external interrupt						
<b>bit 3</b>	<b>RBIE: RB Port Change Interrupt Enable bit</b> 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt						
<b>bit 2</b>	<b>TMROIF: TMRO Overflow Interrupt Flag bit</b> 1 = TMRO register has overflowed (must be cleared in software) 0 = TMRO register did not overflow						
<b>bit 1</b>	<b>INT0IF: RB0/INT External Interrupt Flag bit</b> 1 = The RB0/INT external interrupt occurred (must be cleared in software) 0 = The RB0/INT external interrupt did not occur						
<b>bit 0</b>	<b>RBIF: RB Port Change Interrupt Flag bit</b> A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared. 1 = At least one of the RB7:RB4 pins changed state (must be cleared in software) 0 = None of the RB7:RB4 pins have changed state						

Figura 7.1.1 Bits del registro INTCON (16F88)

- ◆ Las banderas de interrupción se activan independientemente del estado de sus bits de habilitación o del bit de habilitación global GIE.

El bit GIE del registro INTCON permite habilitar o deshabilitar la generación de interrupciones. Cuando están habilitadas (GIE=1) y el bit de habilitación particular y la bandera correspondiente se activan, se produce un salto al vector de interrupción (dirección 0x0004). Las interrupciones individuales pueden habilitarse/deshabilitarse a través de sus bits de habilitación en diferentes registros. El bit GIE se borra al producirse un reset, por lo tanto la generación de interrupciones está deshabilitada normalmente.

La instrucción RETFIE se emplea para salir de la *rutina de servicio a la interrupción* (ISR), así como reabilitar la generación de interrupciones.

Las banderas de las interrupciones INT, RB y del Timer0 se encuentran en el registro INTCON. Las banderas de interrupción de los periféricos están contenidas en los registros PIR1 y PIR2 (16F877A), mientras que los bits de habilitación correspondientes se encuentran en los registros PIE1 y PIE2 (16F877A). El bit de habilitación de interrupciones de periféricos (PEIE) está en el registro INTCON.

INTCON – INTERRUPT CONTROL REGISTER {ADDRESS: 0Bh, 8Bh, 10Bh, 18Bh}							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
bit 7 GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF bit 0

bit 7	GIE: Global Interrupt Enable bit 1 = Enables all un-masked interrupts 0 = Disables all interrupts
bit 6	PEIE: Peripheral Interrupt Enable bit 1 = Enables all un-masked peripheral interrupts 0 = Disables all peripheral interrupts
bit 5	TOIE: TMR0 Overflow Interrupt Enable bit 1 = Enables the TMR0 interrupt 0 = Disables the TMR0 interrupt
bit 4	INTE: RB0/INT External Interrupt Enable bit 1 = Enables the RB0/INT external interrupt 0 = Disables the RB0/INT external interrupt
bit 3	RBIE: RB Port Change Interrupt Enable bit 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt
bit 2	TOIF: TMR0 Overflow Interrupt Flag bit 1 = TMR0 register has overflowed (must be cleared in software) 0 = TMR0 register did not overflow
bit 1	INTF: RB0/INT External Interrupt Flag bit 1 = The RB0/INT external interrupt occurred (must be cleared in software) 0 = The RB0/INT external interrupt did not occur
bit 0	RBIF: RB Port Change Interrupt Flag bit 1 = When at least one of the RB<7:4> pins changes state (must be cleared in software) 0 = None of the RB<7:4> pins have changed state

Figura 7.1.2 Bits del registro INTCON (16F628A y 16F877A)

Cuando se brinda atención a una interrupción, el bit GIE es borrado para deshabilitar cualquier interrupción adicional, la dirección de retorno es guardada (*pushed*) en la pila (*stack*) y el contador de programa (PC) es cargado con el valor 0x0004. Una vez dentro de la ISR, la fuente de la interrupción se puede determinar analizando las banderas de interrupción. Las banderas tienen que ser borradas por *software* antes de rehabilitar las interrupciones, para evitar interrupciones repetitivas.

Las interrupciones externas INT o RB pueden generarse cada cierto tiempo como mínimo, que va desde los tres a cuatro ciclos de instrucción, esto depende del instante en que se genera la interrupción. Las banderas de interrupción se activan independientemente del bit de habilitación particular, del bit PEIE o del bit GIE.

## 7.1 INTERRUPCIÓN INT

La interrupción externa en el pin RB0/INT se activa por flanco ascendente o descendente, dependiendo del bit INTEDG del registro OPTION\_REG. Cuando aparece una transición válida en el pin RB0/INT, la bandera INT0IF del registro INTCON toma un valor de 1. Esta interrupción puede ser habilitada/deshabilitada con el bit INT0IE del registro INTCON. La bandera INT0IF tiene que ser borrada por *software* dentro de la ISR antes de rehabilitar esta interrupción. La interrupción INT puede despertar al PIC, si el bit INT0IE se programó en 1 antes de ingresar al modo *Sleep*. El estado del bit GIE determina si se produce o no el salto al vector de interrupción después del despertar (los detalles aparecen en las secciones **6.11 MODO DE BAJO CONSUMO (Sleep) PIC16F88** o **6.19 MODO DE BAJO CONSUMO (Sleep) PIC16F628A y PIC16F877A**).

## 7.2 INTERRUPCIÓN DEL Timer0

El desbordamiento del registro TMR0 (desde 0xFF a 0x00) genera una interrupción, lo cual hace que el bit TMR0IF del registro INTCON sea igual a 1. La generación de esta interrupción se puede habilitar/deshabilitar con el bit TMR0IE del registro INTCON. El bit TMR0IF tiene que ser borrado por *software* dentro de la ISR antes de rehabilitar esta interrupción. Esta interrupción no puede despertar al PIC, ya que el temporizador está apagado durante el modo *Sleep*.

## 7.3 INTERRUPCIÓN RB

Un cambio de estado en cualquiera de los pines RB<7:4> genera una interrupción y hace que la bandera RBIF del registro INTCON tome un valor de 1. Esta interrupción puede habilitarse/deshabilitarse con el bit

RBIE del registro INTCON. Únicamente los pines configurados como entradas pueden producir esta interrupción. Los pines de entrada RB<7:4> se comparan con el estado anterior que tenían en la última lectura del puerto B. Si no hay coincidencia en todos los pines, se genera la interrupción.

Esta interrupción puede despertar al PIC. El usuario, dentro de la ISR, puede borrar la bandera de interrupción con cualquiera de los métodos siguientes:

- Lectura o escritura del registro PORTB. Esto concluye la condición de falta de coincidencia.
- Borrar la bandera RBIF.

Esta interrupción se recomienda para despertar al PIC en caso de presionar una tecla o en el caso de que el puerto B se emplee únicamente para la interrupción RB. La lectura continua (*Polling*) del puerto B no se recomienda mientras se usa la función de interrupción RB.

## 7.4 MANEJO DE INTERRUPCIONES EN mikroC™

Las interrupciones pueden manipularse fácilmente por medio de la palabra reservada **interrupt**. En mikroC™ se ha declarado de manera implícita la función **interrupt**, la cual no puede ser redeclarada. Su prototipo es:

```
void interrupt(void);
```

Lo único que el usuario tiene que hacer es escribir la definición de esta función (*rutina de servicio a la interrupción ISR*) para manejar interrupciones en la aplicación que esté desarrollando. mikroC™ se encarga de salvar y recuperar de la pila (*stack*) los registros W, STATUS, FSR y PCLATH.

Se pueden realizar llamadas a funciones desde la función **interrupt**. El compilador toma en cuenta los registros que se están empleando tanto en la función **interrupt** como en la función **main**, y salva únicamente los registros que se emplean en ambas funciones.

En caso de que haya múltiples interrupciones habilitadas, se debe detectar la fuente de la interrupción por medio de las banderas de interrupción (*flags*) y proceder a la ejecución del código apropiado.

## 7.5 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

El siguiente ejemplo fue resuelto en el Capítulo III (**LCD1.c**) empleando la técnica de *lectura continua (Polling)* de la entrada RA4; ahora se resolverá con la interrupción INT. Nótese el uso del pin RB0/INT para cumplir dos funciones: como salida de datos hacia el LCD y como entrada para la interrupción INT. El programa se encarga de mantener el valor original del registro TRISB.

**INT1.c:** Cada vez que presiona el pulsador conectado en RB0 se incrementa un contador que se visualiza en el centro de la segunda línea de la pantalla (figuras 7.2.1 y 7.2.2). Si la cuenta supera 100, el conteo se reinicia desde 0. En el centro de la primera línea se muestra la palabra “Conteo:”. *Conversions*  *LCD* .

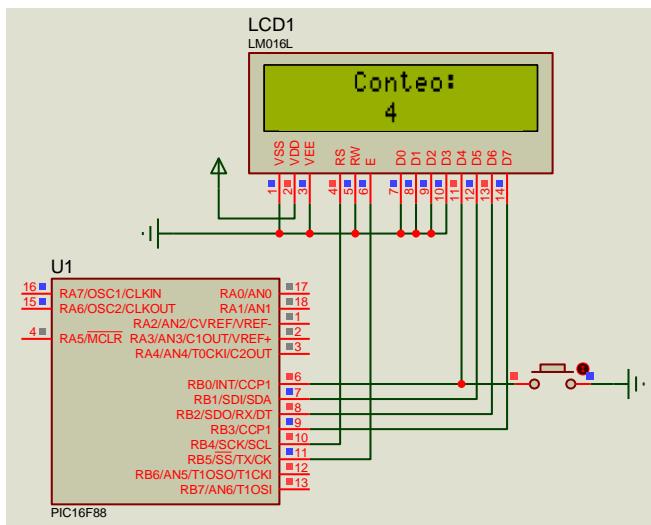
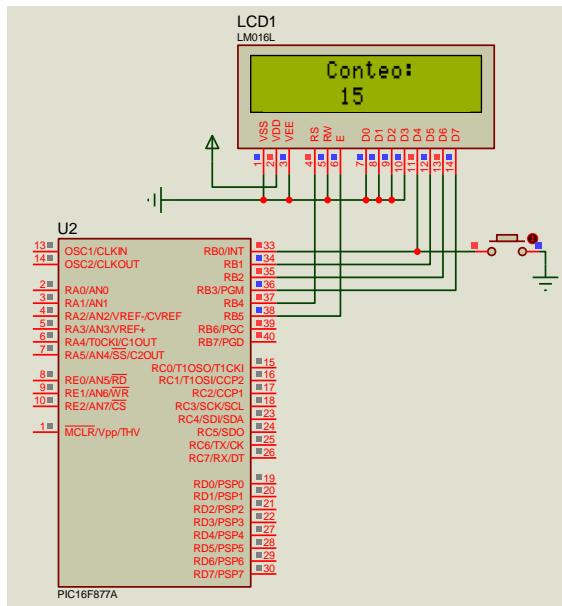


Figura 7.2.1 Circuito del problema INT1.c (PIC16F88 y 16F628A)

Figura 7.2.2 Circuito del problema INT1.c  
(PIC16F877A)

```


//INT1.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

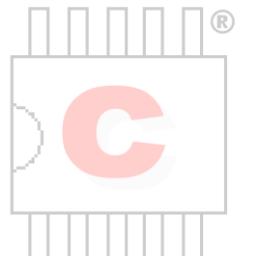
char contador=0, texto[4], respaldo;
www.programarpicenc.com
©Ing. Juan Ricardo Penagos Plazas

void main() {
    OSCCON=0x40; //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
    GIE_bit=1; //Interrupciones habilitadas.
    NOT_RBPU_bit=0; //Pull ups habilitados.
    INTEDG_bit=0; //INT por flanko descendente.

    Lcd_Init(); //Inicializa el LCD.
    Lcd_Cmd(_LCD_CLEAR); //Borra el display.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
    Lcd_Out(1,6,"Conteo:");
    ByteToStr(contador, texto);
    Lcd_Out(2,6,texto);

    respaldo=TRISB; //Guardar el estado de TRISB.
    TRISB0_bit=1; //RB0 como entrada.
    INTEDG_bit=1; //Interrupción INT habilitada.
    while (1)
    {
        asm SLEEP //Entra en modo SLEEP.
        asm NOP //Se despierta por INT, ejecuta NOP y
        //salta a "interrupt".
        ByteToStr(contador, texto); //Retorna de "interrupt" y continúa.
        TRISB=respaldo; //Restaurar TRISB.
        Lcd_Out(2,6,texto);
        respaldo=TRISB; //Guardar el estado de TRISB.
        TRISB0_bit=1; //RB0 como entrada.
    }
}

```

*microC®*

```

void interrupt(void)
{
    Delay_ms(20);
    if (RB0_bit==0) contador++; //Pulsador presionado.
    while (RB0_bit==0); //Esperar mientras siga presionado.
    if (contador >100) contador=0;
    INTF_bit=0;
}

```

En el subsiguiente ejemplo se deben tomar muy en cuenta estos detalles:

- El pin RB0/INT se emplea exclusivamente para la interrupción INT. No se puede emplear también para enviar datos al LCD, ya que mientras se mantenga presionado el pulsador el nivel lógico en este pin es 0, lo que interfiere con los datos de salida.
- Las interrupciones globales permanecen deshabilitadas (GIE=0). Por lo tanto, al despertarse el PIC no se produce el salto al vector de interrupción (dirección 0x0004).
- El bit D4 del LCD ahora se conecta al pin RB6 del PIC.

**INT2.c:** Mientras se presiona el pulsador conectado en RB0 se incrementa un contador cada 200 ms, el cual se visualiza en el centro de la segunda línea de la pantalla (figura 7.3). Si la cuenta supera 100, el conteo se reinicia desde 0. En el centro de la primera línea se muestra la palabra “Conteo:”. *Conversions*  *LCD* .

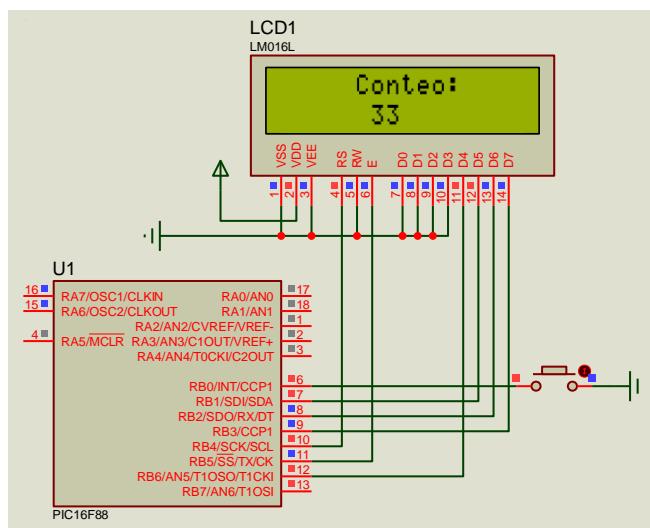


Figura 7.3 Circuito del problema INT2.c

```


//INT2.c
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB6_bit; //Cambiado.
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 TRISB6_bit; //Cambiado.
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Fin de declaración de variables de conexión.
char contador=0, texto[4];

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOF0==0); //Esperar mientras el oscilador está inestable.


```

```

ANSEL=0x00;           //Bits AN6:AN0 como E/S digital.
NOT_RBPU_bit=0;       //Pull ups habilitados.
INTEDG_bit=0;         //INT por flanco descendente.

Lcd_Init();           //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,6,"Conteo:");
ByteToStr(contador, texto);
Lcd_Out(2,6,texto);
INTE_bit=1;            //Interrupción INT habilitada.
while (1)
{
    asm SLEEP          //Entra en modo SLEEP.
    Delay_ms(20);      //Se despierta por INT y continúa.
    while (RB0_bit==0) //Mientras el pulsador siga presionado.
    {
        contador++;   //Contador++
        if (contador>100) contador=0;
        ByteToStr(contador, texto);
        Lcd_Out(2,6,texto);
        Delay_ms(200);
        INTF_bit=0;
    }
}
}

```

Debe observarse que debido a que la corriente de salida de cada pin del PIC está limitada a 25 mA, se pueden conectar LEDs sin resistencias en serie, sin peligro alguno, como se hace en el siguiente ejemplo.

**INT3.c:** Mientras se presiona el pulsador conectado en RB0 el LED conectado a RB1 (figura 7.4) parpadea con una cadencia de 500 ms ON y 200 ms OFF.

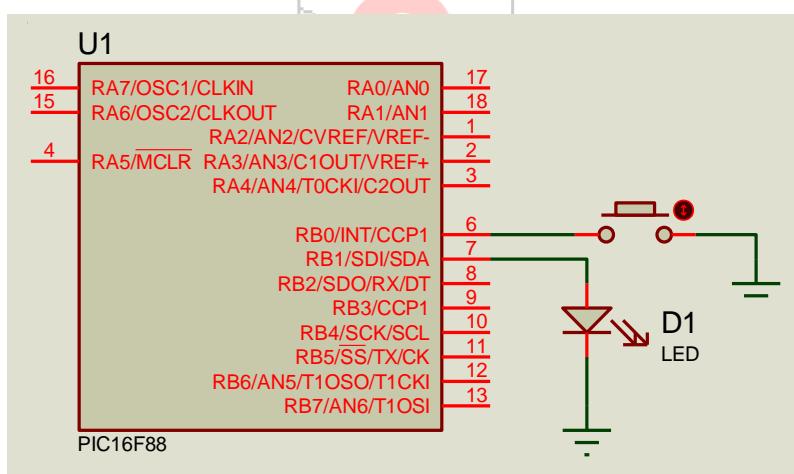


Figura 7.4 Circuito del problema INT3.c

```

//INT3.c
void main(){
    OSCCON=0x40;           //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00;             //Inicialización.
    ANSEL=0x00;              //Bits AN6:AN0 como E/S digital.
    TRISB1_bit=0;            //RB1 como salida.
    NOT_RBPU_bit=0;          //Pull ups habilitados.
    INTEDG_bit=0;            //INT por flanco descendente.
    INTE_bit=1;               //Interrupción INT habilitada.
    while (1)
    {
        asm SLEEP          //Entra en modo SLEEP.
        Delay_ms(20);      //Se despierta por INT y continúa.
        while (RB0_bit==0) //Mientras el pulsador siga presionado.
        {
            RB1_bit=1;      //RB1_bit=1;
            Delay_ms(500); //Delay_ms(500);
        }
    }
}

```

```

    RB1_bit=0;
    Delay_ms(200);
    INTF_bit=0;
}
}
}

```

**INT4.c:** Cada vez que presione el pulsador conectado en RB0 comutará el estado de un LED conectado en RB7. Al mismo tiempo, en el LCD se visualizará un mensaje desplazándose. *LCD*  4MHz .

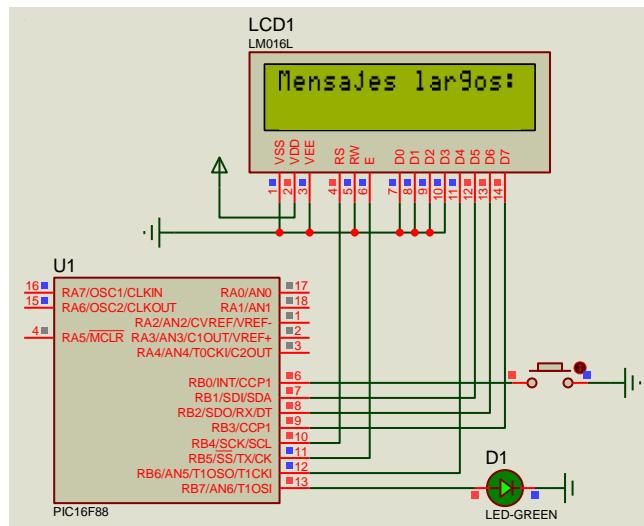


Figura 7.5.1 Circuito del problema INT4.c (PIC16F88 y 16F628A)

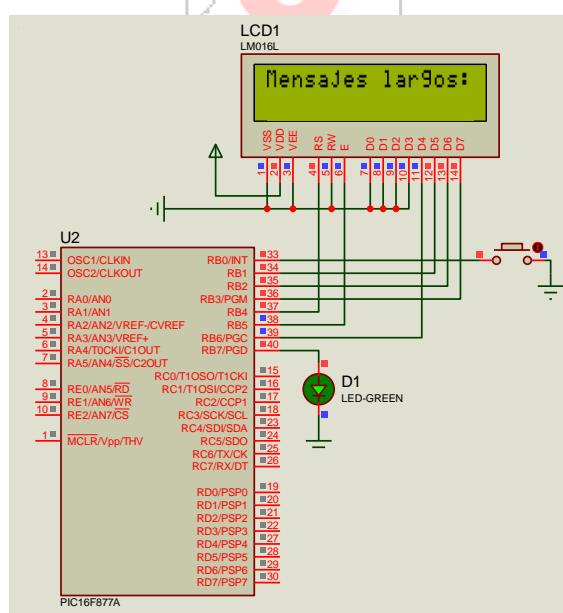


Figura 7.5.2 Circuito del problema INT4.c (PIC16F877A)

```


//INT4.c
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB6_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 TRISB6_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// Fin de declaración de variables de conexión.

char i;

void main(){
OSCCON=0x60; //Oscilador interno a 4MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
RB7_bit=0; //Estado inicial.
TRISB7_bit=0; //RB7 como salida.
GIE_bit=1; //Interrupciones habilitadas.
NOT_RBPU_bit=0; //Pull ups habilitados.
INTEDG_bit=0; //INT por flanco descendente.

Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

INTE_bit=1; //Interrupción INT habilitada.
while (1){
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_Out(1,1,"Mensajes largos: maximo de 40 caracteres.");
    Delay_ms(400);
    for (i=39;i>0;i--){
        Lcd_Cmd(_LCD_SHIFT_RIGHT); //Desplaza la pantalla (no el texto) a la
                                    //derecha.
        Delay_ms(250);
    }
    Lcd_Cmd(_LCD_CLEAR);
}
}

void interrupt(void){
Delay_ms(20);
if (RB0_bit==0) RB7_bit=~RB7_bit; //Pulsador presionado.
while (RB0_bit==0); //Esperar mientras siga presionado.
INTF_bit=0;
}
}

```



[www.programarpicenc.com](http://www.programarpicenc.com)

Los siguientes ejemplos muestran cómo emplear la interrupción RB.

**RB1.c:** Cada vez que presione un pulsador conectado en RB6 se incrementará un contador que será visualizado en el LCD (figura 7.6). *Conversions*  *LCD*  *4MHz*

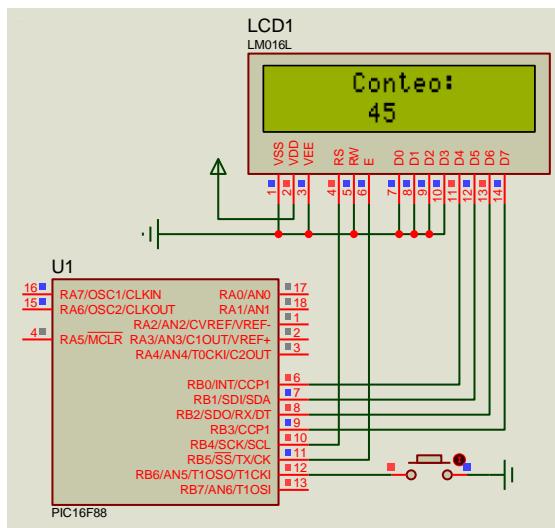


Figura 7.6 Circuito del problema RB1.c

```


//RB1.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador=0, texto[4];

void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
GIE_bit=1;              //Interrupciones habilitadas.
NOT_RBPU_bit=0;         //Pull ups habilitados.

Lcd_Init();             //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);   //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,6,"Conteo:");
ByteToStr(contador, texto);
Lcd_Out(2,6,texto);

RBIE_bit=1;              //Interrupción RB habilitada.
while (1){
asm SLEEP
asm NOP
//Se despierta por RB, ejecuta NOP y
//salta a "interrupt".
ByteToStr(contador, texto); //Retorna de "interrupt" y continúa.
Lcd_Out(2,6,texto);
}
}

void interrupt(void){
Delay_ms(20);
if (RB6_bit==0) contador++; //Pulsador presionado.
while (RB6_bit==0); //Esperar mientras siga presionado.
if (contador >100) contador=0;
RBIF_bit=0;
}
}

www.programarpicenc.com
Cpt. Juan Ricardo Penagos Plazas

```

**RB2.c:** Un LED conectado a la línea RB2 se enciende durante 5 s cuando pulse RB7 y durante 1 s cuando pulse RB6 (figura 7.7).

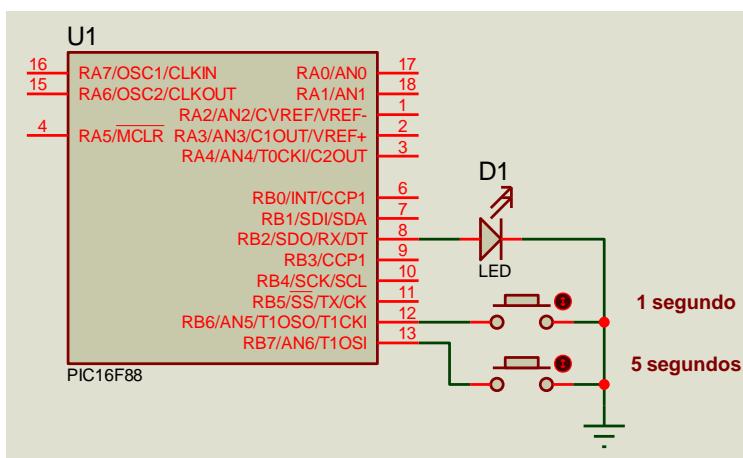


Figura 7.7 Circuito del problema RB2.c

```

 //RB2.c
void main()
{
    OSCCON=0x40;           //Oscilador interno a 1MHz.
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0;                //Inicialización.
    ANSEL=0x00;              //Bits AN6:AN0 como E/S digital.
    TRISB2_bit=0;            //RB2 como salida.
    GIE_bit=1;                //Interrupciones habilitadas.
    NOT_RBPU_bit=0;          //Pull ups habilitados.
    RBIE_bit=1;                //Interrupción RB habilitada.
    while (1){
        asm SLEEP             //Entra en modo SLEEP.
        asm NOP                 //Se despierta por RB, ejecuta NOP y
        //salta a "interrupt".
        //Retorna de "interrupt" y continúa.
    }
}

void interrupt(void)
{
    Delay_ms(20);           //Pulsador RB6 presionado.
    if (RB6_bit==0)
    {
        RB2_bit=1;
        Delay_ms(1000);
        RB2_bit=0;
    }
    while (RB6_bit==0);      //Esperar mientras siga presionado.

    if (RB7_bit==0)          //Pulsador RB7 presionado.
    {
        RB2_bit=1;
        Delay_ms(5000);
        RB2_bit=0;
    }
    while (RB7_bit==0);      //Esperar mientras siga presionado.
    RBIF_bit=0;
}
}

```



**RB3.c:** Dos pulsadores conectados en RB7 y RB6 producen una interrupción cada vez que se pulsan (figura 7.8). En el LCD aparece el nombre del pulsador activado: “RB7” o “RB6”. *LCD*

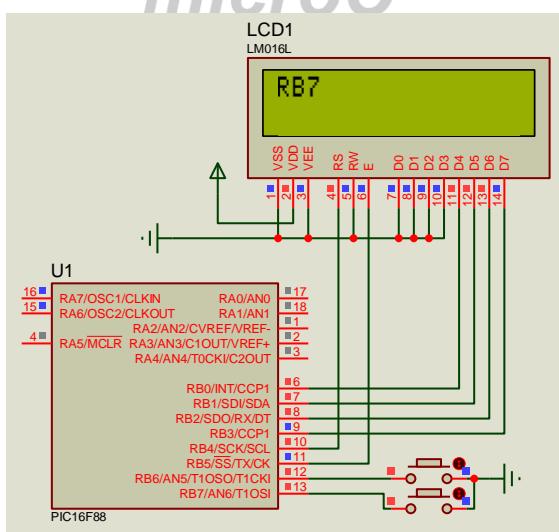


Figura 7.8 Circuito del problema RB3.c

```

 //RB3.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

void main(){
OSCCON=0x40;           //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
NOT_RBPU_bit=0;         //Pull ups habilitados.

Lcd_Init();            //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);   //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

RBIE_bit=1;             //Interrupción RB habilitada.
while (1)
{
asm SLEEP              //Entra en modo SLEEP.
Delay_ms(20);           //Se despierta por RB y continúa.

if (RB6_bit==0)          //Pulsador RB6 presionado.
Lcd_Out(1,1,"RB6");    //Esperar mientras siga presionado.

if (RB7_bit==0)          //Pulsador RB7 presionado.
Lcd_Out(1,1,"RB7");    //Esperar mientras siga presionado.
RBIF_bit=0;
}
}

```

Los siguientes ejemplos muestran cómo emplear la interrupción del Timer0.

**(PIC16F88)** Con una frecuencia de oscilador de 8MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 0,5us, por lo tanto el registro TMR0 debe ser cargado con un valor de  $256-100=156$  para contar 100 ciclos de instrucción ( $100 \times 0,5\text{us}=50\text{ us}$ ). Debe notarse que este valor es únicamente teórico, debido a que el tiempo de ejecución de las instrucciones hará que el semiperíodo sea mayor, por lo tanto el valor de carga de TMR0 debería ser corregido experimentalmente para obtener exactamente la frecuencia deseada. El prescaler tiene un valor 1:1 ya que se encuentra asignado al WDT. Estos datos se emplean en el siguiente ejemplo.

**(PIC16F628A y 16F877A)** Con una frecuencia de oscilador de 4MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 1us, por lo tanto el registro TMR0 debe ser cargado con un valor de  $256-50=206$  para contar 50 ciclos de instrucción ( $50 \times 1\text{us}=50\text{ us}$ ).

**Int\_Timer0\_1.c:** Por la línea RB3 se genera una onda cuadrada de 10 kHz (cada semiperíodo durará 50 us). 8MHz.

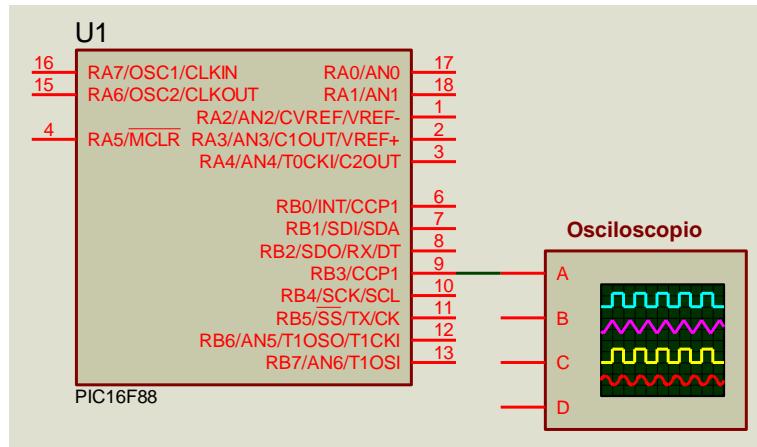


Figura 7.9 Circuito del problema Int\_Timer0\_1.c

```


//Int_Timer0_1.c
void main(){
    OSCCON=0x70;           //Oscilador interno a 8MHz (TCI=0,5 us).
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00;             //Inicialización.
    ANSEL=0x00;              //Pines AN<6:0> como E/S digital.
    TRISB3_bit=0;            //RB3 como salida.
    T0CS_bit=0;              //Timer0 como temporizador.
    GIE_bit=1;                //Interrupciones habilitadas.
    TMROIE_bit=1;             //Interrupción del Timer0 habilitada.
    }                         //Esperar.

    void interrupt(void)
    {
        TMRO=156;           //Valor inicial del TMRO.
        RB3_bit=~RB3_bit;     //Cambiar el estado del pin RB3.
        TMROIF_bit=0;          //Borra la bandera de interrupción.
    }
}

```

**(PIC16F88)** Con una frecuencia de oscilador de 1MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 4us, por lo tanto el registro TMRO debe ser cargado con un valor de  $256-244=12$  para contar 244 ciclos de prescaler (que tiene un valor de 1:256 para el siguiente ejemplo), lo que da un intervalo de temporización de  $244 \times 4 \times 256 \text{ us} = 249.856 \text{ us} = 0,250 \text{ s} (1/4 \text{ s})$ . Este valor se toma como base de tiempo para lograr la medición de intervalos largos, con la ayuda de un contador auxiliar, como se muestra a continuación.

©Ing. Juan Ricardo Penagos Plazas

**(PIC16F628A y 16F877A)** Con una frecuencia de oscilador de 4MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 1us, por lo tanto el registro TMRO debe ser cargado con un valor de  $256-244=12$  para contar 244 ciclos de prescaler (que tiene un valor de 1:256 para el siguiente ejemplo), lo que da un intervalo de temporización de  $244 \times 1 \times 256 \text{ us} = 62.464 \text{ us} = 0,0625 \text{ s} (1/16 \text{ s})$ .

**Int\_Timer0\_2.c:** Un LED conectado en RB1 se enciende repetitivamente durante 500 ms y se apaga durante otros 500 ms.

```


//Int_Timer0_2.c
char contador=0;

void main(){
    OSCCON=0x40;           //Oscilador interno a 1MHz (TCI=4 us).
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    PORTB=0x00;             //Inicialización.
    ANSEL=0x00;              //Pines AN<6:0> como E/S digital.
    TRISB1_bit=0;            //RB1 como salida.
    OPTION_REG=0b11010111;   //Timer0 como temporizador. Prescaler asignado
                            //al Timer0. Prescaler 1:256.
    GIE_bit=1;                //Interrupciones habilitadas.
    TMROIE_bit=1;             //Interrupción del Timer0 habilitada.
    }                         //Esperar.

    void interrupt(void){
        TMRO=12;           //Valor inicial del TMRO. Reinicia el prescaler.
        contador++;          //Contador que controla el encendido/apagado del LED.
        if (contador==2) {

```

```

RB1_bit=~RB1_bit; //Cambiar el estado del pin RB1.
contador=0;
}
TMR0IF_bit=0; //Borra la bandera de interrupción.
}

```

**Int\_Timer0\_3.c:** En el LCD se visualiza constantemente un mensaje largo que se desplaza. Al mismo tiempo un LED conectado en RB6 parpadea a razón de 500 ms ON y 500 ms OFF. **LCD**☒.

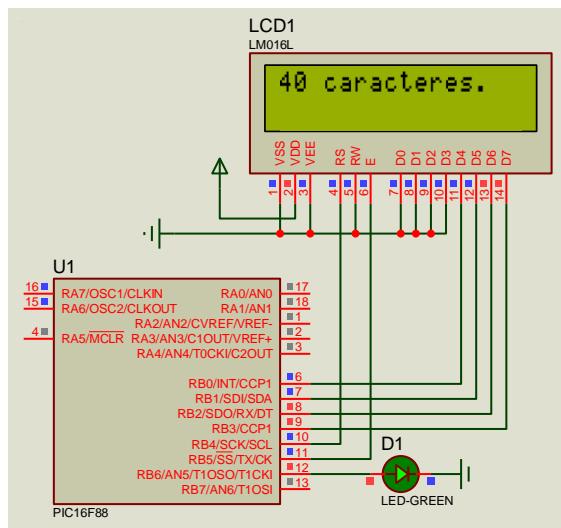


Figura 7.10 Circuito del problema Int\_Timer0\_3.c

```


//Int_Timer0_3.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.
char i, contador=0;

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz.
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00; //Inicialización.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
TRISB6_bit=0; //RB6 como salida.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

OPTION_REG=0b11010111; //Timer0 como temporizador. Prescaler asignado
//al Timer0. Prescaler 1:256.
GIE_bit=1; //Interrupciones habilitadas.
TMR0IE_bit=1; //Interrupción del Timer0 habilitada.

while (1){
Lcd_Cmd(_LCD_CLEAR);
Lcd_Out(1,1,"Mensajes largos: máximo de 40 caracteres.");
Delay_ms(400);
for (i=40;i>0;i--) {

```

microC®

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

```

    Lcd_Cmd(_LCD_SHIFT_RIGHT); //Desplaza la pantalla (no el texto) a la
                                //derecha.
    Delay_ms(250);           //Esperar 250 ms.
}
}

void interrupt(void){
    TMR0=12; //Valor inicial del TMR0. Reinicia el prescaler.
    contador++;
    if (contador==2){
        RB6_bit=~RB6_bit; //Cambiar el estado del pin RB6.
        contador=0;
    }
    TMROIF_bit=0; //Borra la bandera de interrupción.
}
}

```

**(PIC16F88)** Con una frecuencia de oscilador de 8MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 0,5us, por lo tanto el registro TMR0 debería ser cargado con un valor de 156 para contar 100 ciclos de instrucción, equivalentes a 50 us (prescaler 1:1). De esta forma se genera una interrupción debida al Timer0 cada 50 us, lo que se pudiera tomar como base de tiempo. Debido a que el tiempo de ejecución de las instrucciones y los saltos hará que el semiperíodo sea mayor, el valor de carga de TMR0 se ha corregido experimentalmente para obtener las frecuencias deseadas con mayor exactitud. En el ejemplo siguiente, el valor de carga de TMR0 es de 178.

**(PIC16F628A y 16F877A)** Con una frecuencia de oscilador de 4MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 1us, por lo tanto el registro TMR0 debería ser cargado con un valor de 206 para contar 50 ciclos de instrucción, equivalentes a 50 us (prescaler 1:1). De esta forma se genera una interrupción debida al Timer0 cada 50 us, lo que se pudiera tomar como base de tiempo. Debido a que el tiempo de ejecución de las instrucciones y los saltos hará que el semiperíodo sea mayor, el valor de carga de TMR0 se ha corregido experimentalmente para obtener las frecuencias deseadas con mayor exactitud. En el ejemplo siguiente, el valor de carga de TMR0 es de 228.

**Int\_Timer0\_4.c:** Por la línea RB6 se genera una onda cuadrada. La frecuencia de la señal de salida cambia mediante activación del pulsador conectado al pin RB7 (tabla 7.1). El LCD muestra la frecuencia generada (figura 7.11).  $LCD \checkmark. 8MHz \checkmark$ .

Pulsación	Frecuencia	Semiperíodo	Factor
(Inicialmente)	10 kHz	50 us = 1x50 us	1
Primera	5 kHz	100 us = 2x50 us	2
Segunda	2 kHz	250 us = 5x50 us	5
Tercera	1 kHz	500 us = 10x50 us	10
Cuarta	500 Hz	1.000 us = 20x50 us	20
Quinta	200 Hz	2.500 us = 50x50 us	50
Sexta	100 Hz	5.000 us = 100x50 us	100
Séptima	50 Hz	10.000 us = 200x50 us	200

Tabla 7.1 Frecuencias de salida del problema Int\_Timer0\_4.c

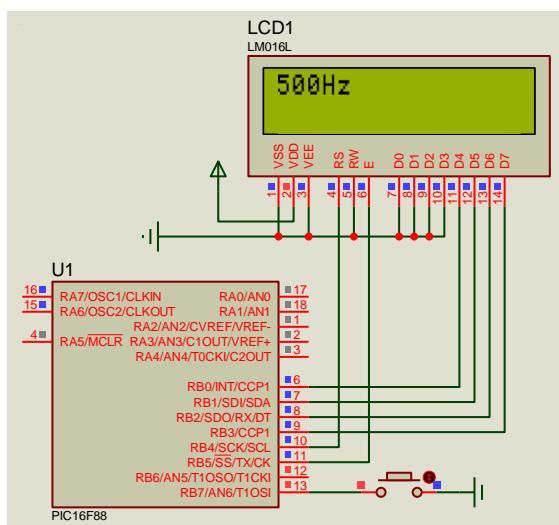


Figura 7.11 Circuito del problema Int\_Timer0\_4.c

```


//Int_Timer0_4.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador=0, contador1=1, factor;

void main(){
OSCCON=0x70;           //Oscilador interno a 8MHz (TCI=0,5 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00;
ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
TRISB6_bit=0;           //RB6 como salida.
NOT_RBPU_bit=0;         //Pull ups habilitados.

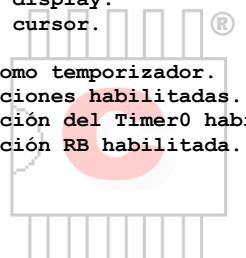
Lcd_Init();              //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);    //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

T0CS_bit=0;               //Timer0 como temporizador.
GIE_bit=1;                //Interrupciones habilitadas.
TMR0IE_bit=1;             //Interrupción del Timer0 habilitada.
RBIE_bit=1;                //Interrupción RB habilitada.

while (1){
switch (contador){
case 0:
Lcd_Out(1,1,"10kHz");
factor=1;
break;
case 1:
Lcd_Out(1,1," 5kHz");
factor=2;
break;
case 2:
Lcd_Out(1,1," 2kHz");
factor=5;
break;
case 3:
Lcd_Out(1,1," 1kHz");
factor=10;
break;
case 4:
Lcd_Out(1,1,"500Hz");
factor=20;
break;
case 5:
Lcd_Out(1,1,"200Hz");
factor=50;
break;
case 6:
Lcd_Out(1,1,"100Hz");
factor=100;
break;
case 7:
Lcd_Out(1,1," 50Hz");
factor=200;
}
}

void interrupt(void){
if (RBIF_bit==1){        //Causa de la interrupción: RB.
Delay_ms(20);
}
}
}

```



**microC®**

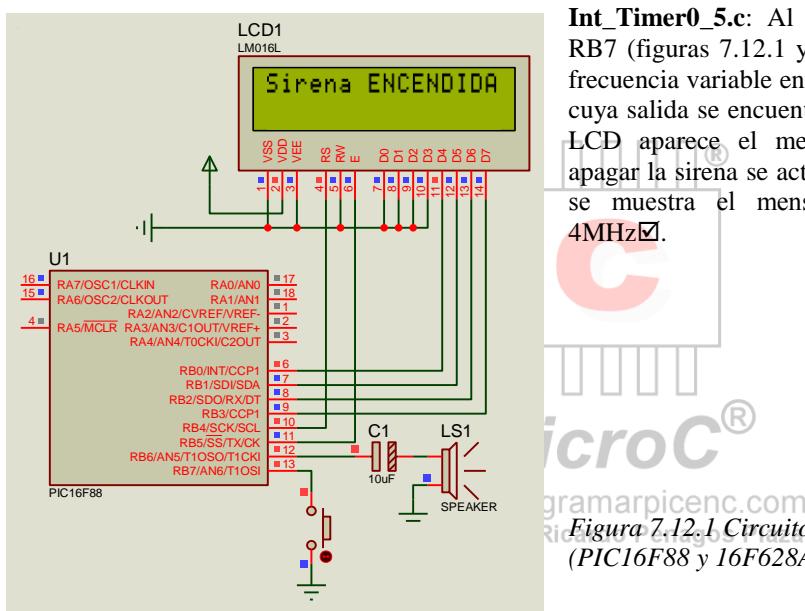
www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

```

if (RB7_bit==0) contador++; //Pulsador presionado.
while (RB7_bit==0); //Esperar mientras siga presionado.
if (contador==8) contador=0;
RBIF_bit=0; //Borra la bandera de interrupción RB.
}
//Causa de la interrupción: Timer0.
TMR0=178; //Valor inicial del TMR0.
if (contador1==factor){
RB6_bit=~RB6_bit; //Cambiar el estado del pin RB6.
contador1=1;
}
else
contador1++;
TMR0IF_bit=0; //Borra la bandera de interrupción.
}

```

**(PIC16F88, 16F628A y 16F877A)** Con una frecuencia de oscilador de 4MHz el ciclo de instrucción ( $T_{CI}$ ) tiene una duración de 1 us, por lo tanto el registro TMR0 debe ser cargado con un valor de  $256-15=241$  para contar 15 ciclos de prescaler (que tiene un valor de 1:8 para el siguiente ejemplo), lo que da un semiperíodo de  $15 \times 1 \times 8 \text{ us}=120 \text{ us}$ , correspondiente a una frecuencia máxima aproximada de 4 kHz. Para una frecuencia mínima aproximada de 300 Hz, el valor de carga del TMR0 debe ser  $256-200=56$ , para contar 200 ciclos de prescaler (semiperíodo= $200 \times 1 \times 8 \text{ us}=1.600 \text{ us}$ ).



**Int\_Timer0\_5.c:** Al presionar un pulsador conectado en RB7 (figuras 7.12.1 y 7.12.2) se activa una sirena con una frecuencia variable entre 4 kHz y 300 Hz aproximadamente, cuya salida se encuentra en el pin RB6; al tiempo que en el LCD aparece el mensaje “Sirena ENCENDIDA”. Para apagar la sirena se actúa en el mismo pulsador y en el LCD se muestra el mensaje “Sirena APAGADA”. *LCD*  $\square$ . 4MHz  $\square$ .

Figura 7.12.1 Circuito del problema Int\_Timer0\_5.c  
(PIC16F88 y 16F628A)

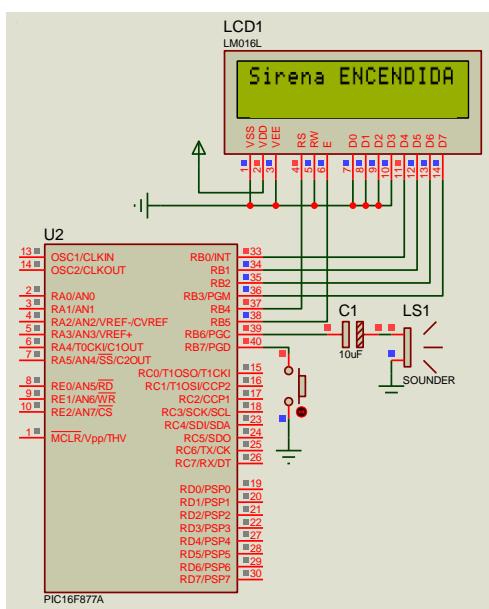


Figura 7.12.2 Circuito del problema Int\_Timer0\_5.c  
(PIC16F877A)

```


//Int_Timer0_5.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char carga=56;
bit variable;

void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00;
ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
TRISB6_bit=0;           //RB6 como salida.
Lcd_Init();              //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);    //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

OPTION_REG=0b01010010;   //Pull ups habilitados.Timer0 como temporizador.
                         //Prescaler asignado al Timer0. Prescaler 1:8.

GIE_bit=1;               //Interrupciones habilitadas.
TMR0IE_bit=1;            //Interrupción del Timer0 habilitada.
RBIE_bit=1;               //Interrupción RB habilitada.

while (1){
if (variable==0)
Lcd_Out(1,1,"Sirena APAGADA");
while (variable==1){
Lcd_Out(1,1,"Sirena ENCENDIDA");
Delay_ms(5);           //Frecuencia inicial=300 Hz.
carga++;                //Disminuye el semiperíodo.
if (carga>241) break;   //Frecuencia final=4 kHz.
}
while (variable==1){
Lcd_Out(1,1,"Sirena ENCENDIDA");
Delay_ms(5);           //Frecuencia inicial=4 kHz.
carga -= 2;              //Aumenta el semiperíodo.
if (carga<56) break;    //Frecuencia final=300 Hz.
}
}
}

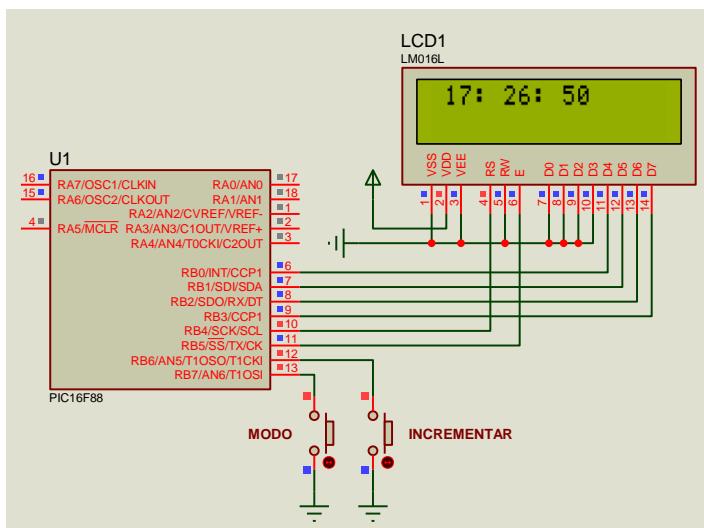
void interrupt(void){
if (RBIF_bit==1){        //Causa de la interrupción: RB.
Delay_ms(20);
if (RB7_bit==0) variable=~variable; //Pulsador presionado.
while (RB7_bit==0); //Esperar mientras siga presionado.
RBIF_bit=0;           //Borra la bandera de interrupción RB.
}

if (variable==1){        //Causa de la interrupción: Timer0.
TMR0=carga;           //Valor inicial del TMR0.Reinicia el prescaler.
RB6_bit=~RB6_bit;
}
TMR0IF_bit=0;           //Borra la bandera de interrupción.
}
}

```

**(PIC16F88, 16F628A y 16F877A)** Para generar una interrupción debida al Timer0 cada 50ms, se utilizarán los siguientes valores: N=195 (Q=61),  $T_{Cl}=1\text{us}$  y un prescaler de 1:256, con lo que se tiene  $T=195 \times 1 \times 256 = 49.920\text{us}$  (50ms).

**Int\_Timer0\_6.c:** Programa para un reloj digital de 24 horas (figuras 7.13.1 y 7.13.2). Se visualiza en formato 16:47:39. La base de tiempo está determinada por el Timer0 que genera una interrupción cada 50 ms. Para configurar la hora se emplean dos pulsadores: “MODO” conectado en RB7 e “INCREMENTAR” conectado en RB6, de la siguiente forma:



- Al pulsar “MODO”, los minutos se ponen intermitentes y se pueden ajustar desde 0 a 59 con el pulsador “INCREMENTAR”.
  - Al pulsar “MODO” por segunda vez, pasa al ajuste de las horas desde 0 a 23 de forma similar.
  - Al pulsar “MODO” por tercera vez, finaliza la configuración de la hora y pasa a la visualización normal.
- Conversions*  *LCD*  *4MHz*

Figura 7.13.1 Circuito del problema Int\_Timer0\_6.c (PIC16F88 y 16F628A)

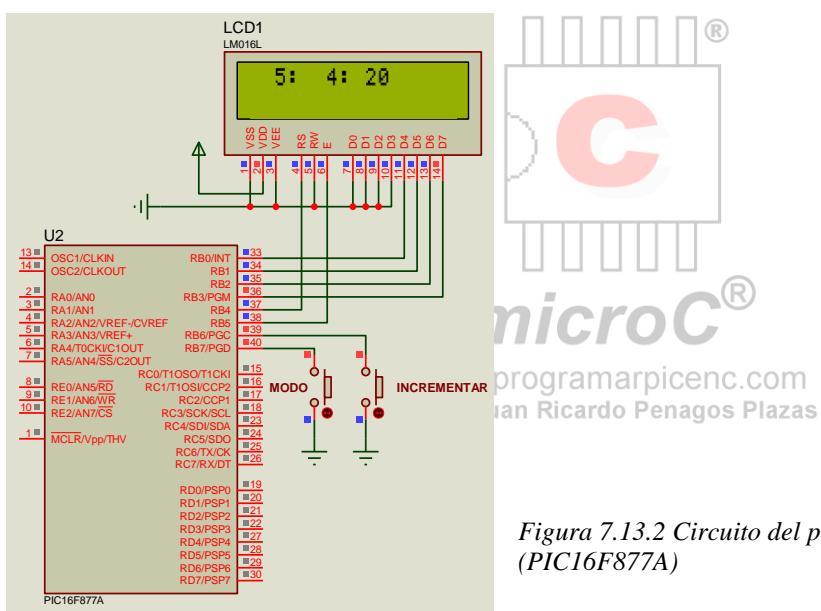


Figura 7.13.2 Circuito del problema Int\_Timer0\_6.c (PIC16F877A)

```

 //Int_Timer0_6.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.

char contador=0,segundos=0,minutos=0,horas=0,texto[4],modo=0;

```

```

void main(){
    OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
    Lcd_Init();              //Inicializa el LCD.
    Lcd_Cmd(_LCD_CLEAR);   //Borra el display.
    Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
    Lcd_Out(1,8,":");
    Lcd_Out(1,4,":");
    OPTION_REG=0b01010111; //Pull ups habilitados.Timer0 como temporizador.
                           //Prescaler asignado al Timer0. Prescaler 1:256.

    TMR0=61;                //Valor inicial del TMR0. Reinicia el prescaler.
    GIE_bit=1;               //Interrupciones habilitadas.
    TMR0IE_bit=1;            //Interrupción del Timer0 habilitada.
    RBIE_bit=1;               //Interrupción RB habilitada.
    while (1){
        if (segundos==60){
            minutos++;
            segundos=0;
        }
        ByteToStr(segundos, texto);
        Lcd_Out(1,9,texto); //Escribe los segundos.

        if (minutos==60){
            horas++;
            minutos=0;
        }
        ByteToStr(minutos, texto);
        Lcd_Out(1,5,texto); //Escribe los minutos.
        if (modo==1){
            Lcd_Out(1,5,texto);
            Delay_ms(500);
            Lcd_Out(1,5," ");
            Delay_ms(500);
        }

        if (horas==24) horas=0;
        ByteToStr(horas, texto);
        Lcd_Out(1,1,texto); //Escribe las horas.
        if (modo==2){
            Lcd_Out(1,1,texto);
            Delay_ms(500);
            Lcd_Out(1,1," ");
            Delay_ms(500);
        }
    }

    void interrupt(void){
        if (RBIF_bit==1){ //Causa de la interrupción: RB.
            Delay_ms(20);
            if (RB7_bit==0) modo++; //Pulsador presionado.
            while (RB7_bit==0); //Esperar mientras siga presionado.
            if (modo==3) modo=0;

            if (modo==1 && RB6_bit==0){ //Pulsador presionado.
                minutos++;
                if (minutos==60) minutos=0;
            }

            if (modo==2 && RB6_bit==0){ //Pulsador presionado.
                horas++;
                if (horas==24) horas=0;
            }
            while (RB6_bit==0); //Esperar mientras siga presionado.
            RBIF_bit=0; //Borra la bandera de interrupción RB.
        }
        TMR0=61; //Valor inicial del TMR0. Reinicia el prescaler.
        contador++; //Cada 49.920 us (50 ms).
        if (contador==20){
            segundos++;
            contador=0;
        }
        TMR0IF_bit=0; //Borra la bandera de interrupción.
    }
}

```



[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO VIII: TECLADO MATRICIAL

En este capítulo se hará el estudio de los teclados matriciales (figura 8.1), que son los más utilizados en el desarrollo de proyectos con PICs y que tienen su aplicación en el ingreso de datos de manera manual por parte del usuario, en aquellos casos en que el empleo de pulsadores simples no es lo más apropiado, ya sea por la presentación final del producto o por la restricción del número de líneas de entrada del PIC.

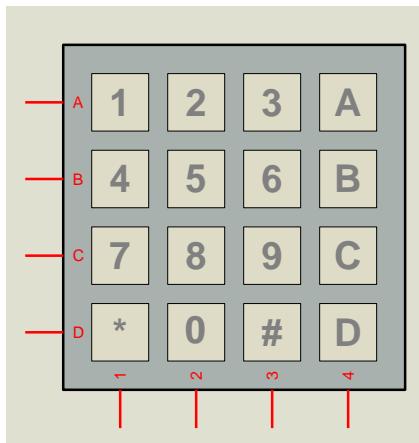


Figura 8.1 Teclado matricial hexadecimal

El teclado matricial está constituido por una matriz de pulsadores dispuestos en filas (A,B,C,D) y columnas (1,2,3,4), con la intención de reducir el número de pines necesarios para su conexión. Las 16 teclas necesitan sólo 8 pines, en lugar de los 16 que se requerirían para 16 teclas independientes. Cuando se presiona una tecla, se conectan internamente la fila y columna correspondientes; por ejemplo, al presionar la tecla “7” se conectan la fila C y la columna 1. Si no hay ninguna tecla presionada, las filas están desconectadas de las columnas.

### 8.1 FUNCIONES DE mikroC™ PARA TECLADO

Función	Descripción
Keypad_Init( )	Inicializa un puerto para el trabajo con el teclado
Keypad_Key_Press( )	Lee una tecla cuando es presionada
Keypad_Key_Click( )	Lee una tecla cuando es presionada y liberada

En la tabla 8.1 se describen las funciones que permiten el trabajo con teclados 4x4 y que se incluyen en la librería *Keypad4x4*.

Tabla 8.1 Funciones para trabajo con teclados matriciales 4x4

Para poder utilizar estas funciones se tiene que declarar previamente una variable que especifica el puerto que se empleará para la conexión del teclado, como se observa en los ejemplos de programación. La conexión a los PICs 16F88, 16F628A y 16F877A se muestra en las figuras 8.2.1, 8.2.2 y 8.2.3 donde se ha empleado el puerto B: las columnas se conectan al nibble bajo, mientras que las filas se conectan al nibble alto del mismo puerto.

©Ing. Juan Ricardo Penagos Plazas

**(PIC16F628A)** Debido a que el pin RA4 trabaja como drenaje abierto cuando se configura como salida, se hace necesario conectar una resistencia de *pull-up* como se observa en la figura 8.2.2.

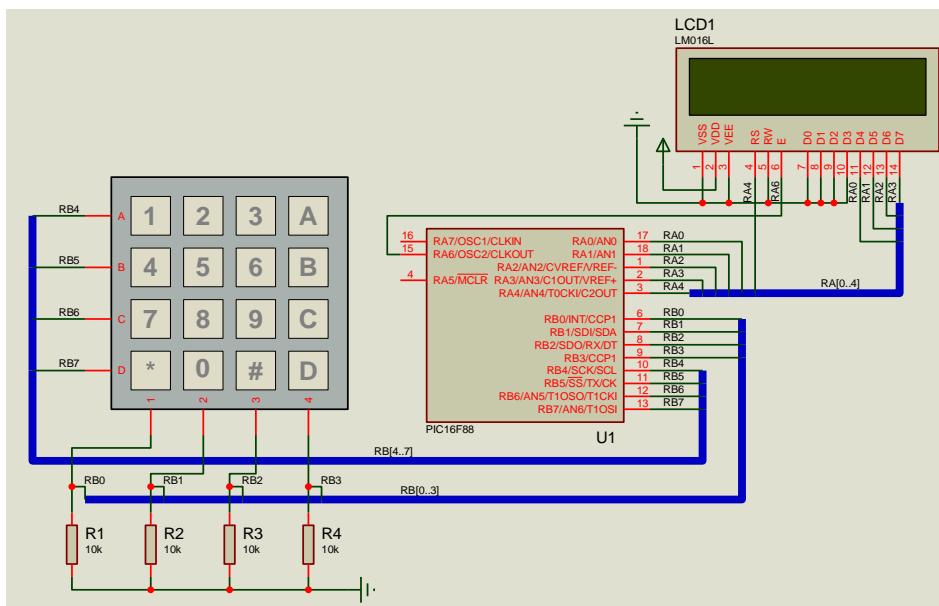


Figura 8.2.1  
Conexión de un teclado matricial al PIC16F88 (16F628A)

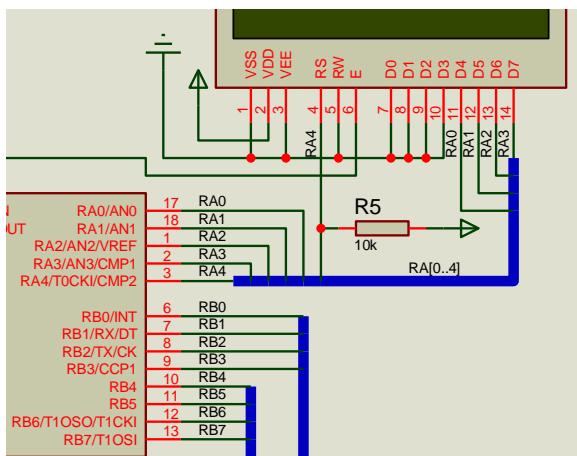


Figura 8.2.2 Detalle de la resistencia de pull-up en el pin RA4 (PIC16F628A)

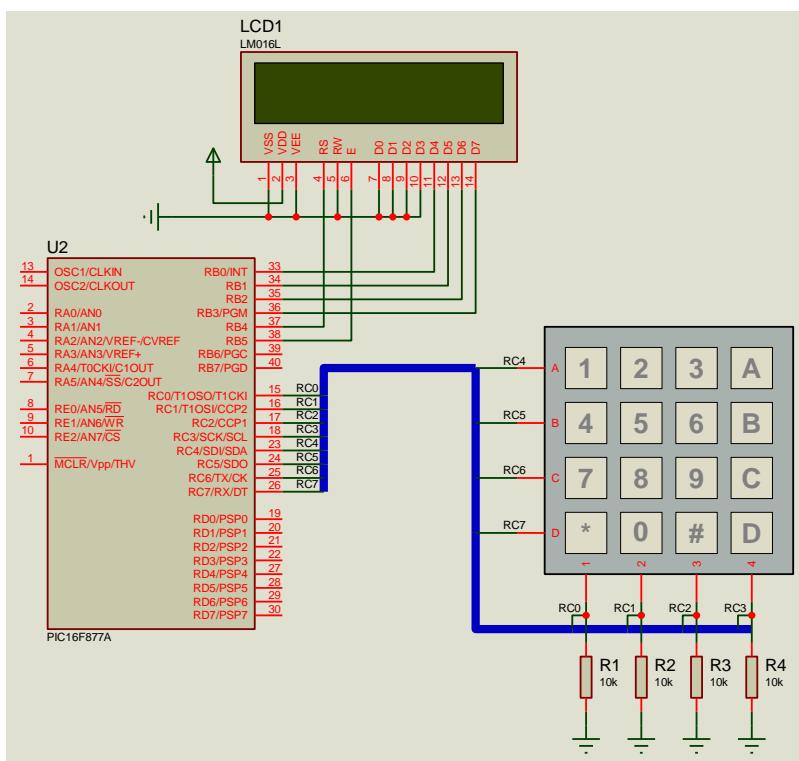


Figura 8.2.3 Conexión de un teclado matricial al PIC16F877A

## 8.2 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos se pueden probar en el circuito de la figura 8.2.1 y corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro. Debe notarse que el código que se obtiene al presionar una tecla es un número entero entre 1 y 16, de izquierda a derecha y de arriba abajo, comenzando por la fila A. Este número se emplea para

asociarle el símbolo respectivo impreso en el teclado (por medio de los códigos ASCII), para su uso posterior en el programa. Esto se detalla en el siguiente ejemplo.

**Teclado\_1.c:** Lo que se va escribiendo por el teclado es visualizado en la primera línea del LCD. Cuando llega al final se borra todo y comienza de nuevo. **LCD** **Keypad4x4**.

```


// Teclado_1.c
// Variable necesaria para la conexión del teclado.
char keypadPort at PORTB;
// Declaración de las 12 variables necesarias para la conexión
// del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;

sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
```

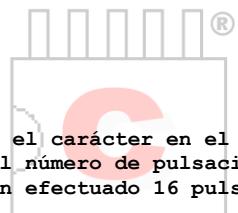
```

sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.
char kp, contador=0;

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz (TCI=4 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Keypad_Init(); //Inicializa el teclado.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

while (1){
kp=0;
do //Espera por una tecla.
kp=Keypad_Key_Click(); //Lee el número de la tecla y lo guarda en kp.
while (!kp);
switch (kp){
case 1: kp = 49; break; //49 es el código ASCII del número 1.
case 2: kp = 50; break; //50 es el código ASCII del número 2.
case 3: kp = 51; break; //51 es el código ASCII del número 3.
case 4: kp = 65; break; // A
case 5: kp = 52; break; // 4
case 6: kp = 53; break; // 5
case 7: kp = 54; break; // 6
case 8: kp = 66; break; // B
case 9: kp = 55; break; // 7
case 10: kp = 56; break; // 8
case 11: kp = 57; break; // 9
case 12: kp = 67; break; // C
case 13: kp = 42; break; // *
case 14: kp = 48; break; // 0
case 15: kp = 35; break; // #
case 16: kp = 68; break; // D
}
Lcd_Ch(chr(kp); //Presenta el carácter en el LCD.
contador++; //Cuenta el número de pulsaciones.
if (contador==16){ //Si se han efectuado 16 pulsaciones.
contador=0;
Delay_1sec(); //Espera 1 s.
Lcd_Cmd(_LCD_CLEAR); //Borra la pantalla y retorna el cursor al
//origen.
}
}
}
}
}


```



**microC®**

**Teclado\_2.c:** Los valores decimales que se escriben por el teclado aparecen en el LCD. Si se pulsa cualquier otra tecla que no sea número lo interpreta como tecla de borrado de la pantalla. *LCD*  *Keypad4x4* .

```


//Teclado_2.c
// Variable necesaria para la conexión del teclado.
char keypadPort at PORTB;
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;
sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.
char kp, contador=0;
void main(){
OSCCON=0x40; //Oscilador interno a 1MHz (TCI=4 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Keypad_Init(); //Inicializa el teclado.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
while (1){

```

```

kp=0;
do                                //Espera por una tecla.
    kp=Keypad_Key_Click(); //Lee el número de la tecla y lo guarda en kp.
    while (!kp);
    switch (kp){                  //Reemplaza el número por el símbolo impreso.
        case 1: kp = 49; break; //49 es el código ASCII del número 1.
        case 2: kp = 50; break; //50 es el código ASCII del número 2.
        case 3: kp = 51; break; //51 es el código ASCII del número 3.
        case 4: kp = 65; break; // A
        case 5: kp = 52; break; // 4
        case 6: kp = 53; break; // 5
        case 7: kp = 54; break; // 6
        case 8: kp = 66; break; // B
        case 9: kp = 55; break; // 7
        case 10: kp = 56; break; // 8
        case 11: kp = 57; break; // 9
        case 12: kp = 67; break; // C
        case 13: kp = 42; break; // *
        case 14: kp = 48; break; // 0
        case 15: kp = 35; break; // #
        case 16: kp = 68; break; // D
    }
    if (kp==65||kp==66||kp==67||kp==42||kp==35||kp==68)
        Lcd_Cmd(_LCD_CLEAR); //Borra la pantalla y retorna el cursor al
                               //origen.
    else{
        Lcd_Ch chr_Cp(kp); //Presenta el carácter en el LCD.
        contador++; //Cuenta el número de pulsaciones.
        if (contador==16){
            contador=0;
            Delay_1sec(); //Espera 1 s.
            Lcd_Cmd(_LCD_CLEAR); //Borra la pantalla y retorna el cursor al
                               //origen.
        }
    }
}
}

```

En el siguiente ejemplo se debe rescatar el valor numérico impreso en las teclas y no su código ASCII, ya que este último se emplea con el propósito de visualización en el LCD, mientras que lo que se pretende es el ingreso de un número de varias cifras a través del teclado; así, por ejemplo, la tecla impresa con el número ‘7’ debe ser leída con el valor de 7 y no con el valor de 55 (código ASCII). Esto se realiza de forma semejante al ejemplo anterior por medio de una instrucción de selección múltiple (**switch**).

**Teclado\_3.c:** Programa para una cerradura electrónica cuya salida (conectada en RA7) se activa durante 3 segundos cuando una clave de 4 dígitos (3589) introducida por el teclado sea correcta. Al tercer intento fallido la puerta queda bloqueada permanentemente y se tendrá que reiniciar el PIC. Si se pulsa cualquier otra tecla que no sea número lo interpreta como tecla de borrado. *Conversions*  *LCD*  *Keypad4x4*

```

 //Teclado_3.c
// Variable necesaria para la conexión del teclado.
char keypadPort at PORTB;
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;
sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.
unsigned umil, centenas, decenas, codigo, clave=3589;
char kp, contador=0, intentos=0, i;

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz (TCI=4 us).
//while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
TRISA7_bit=0; //RA7 como salida. Inicialmente vale 0.
}

```

```

Keypad_Init();           //Inicializa el teclado.
Lcd_Init();             //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);   //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

while (1){
    Lcd_Out(1,1,"Clave:4 digitos?");
    kp=0;
    do                      //Espera por una tecla.
        kp=Keypad_Key_Click(); //Lee el número de la tecla y lo guarda en kp.
    while (!kp);
    switch (kp){
        case 1: kp = 1; break; //La tecla número 1 corresponde al número 1.
        case 2: kp = 2; break; //La tecla número 2 corresponde al número 2.
        case 3: kp = 3; break; //La tecla número 3 corresponde al número 3.
        case 4: kp = 65; break; //65 es el código ASCII de la 'A'.
        case 5: kp = 4; break; //La tecla número 5 corresponde al número 4.
        case 6: kp = 5; break; //La tecla número 6 corresponde al número 5.
        case 7: kp = 6; break; //La tecla número 7 corresponde al número 6.
        case 8: kp = 66; break; //66 es el código ASCII de la 'B'.
        case 9: kp = 7; break; //La tecla número 9 corresponde al número 7.
        case 10: kp = 8; break; //La tecla número 10 corresponde al número 8.
        case 11: kp = 9; break; //La tecla número 11 corresponde al número 9.
        case 12: kp = 67; break; //67 es el código ASCII de la 'C'.
        case 13: kp = 42; break; //42 es el código ASCII del '*'.
        case 14: kp = 0; break; //La tecla número 14 corresponde al número 0.
        case 15: kp = 35; break; //65 es el código ASCII de '#'.
        case 16: kp = 68; break; //68 es el código ASCII de la 'D'.
    }
    if (kp==65||kp==66||kp==67||kp==42||kp==35||kp==68){
        Lcd_Cmd(_LCD_CLEAR); //Borra la pantalla y retorna el cursor al
        contador=0;          //origen.
    }
    else{
        contador++;          //Cuenta el número de pulsaciones.
        switch (contador){
            case 1:Lcd_Ch(2,1,'*');
                umil=kp;
                umil<<12; //umil=0buuuu 0000 0000 0000
                break;
            case 2:Lcd_Ch(2,2,'*');
                centenas=kp;
                centenas<<8; //centenas=0b0000 cccc 0000 0000
                break;
            case 3:Lcd_Ch(2,3,'*');
                decenas=kp;
                decenas<<4; //decenas=0b0000 0000 dddd 0000
                break;
            case 4:contador=0; //Si se han efectuado 4 pulsaciones.
                Lcd_Ch(2,4,'*');
                umil|=centenas; //Recupera las centenas.
                umil|=decenas; //Recupera las decenas.
                umil|=kp; //Recupera las unidades.
                codigo=Bcd2Dec16(umil); //Obtiene el equivalente decimal del
                //número umil=0xucdu.

                if (codigo==clave){
                    intentos=0;
                    RA7_bit=1; //Activa la cerradura.
                    for (i=1;i<=3;i++)
                        //Espera 3 segundos.
                        Delay_1sec();
                    RA7_bit=0; //Desactiva la cerradura.
                }
                else{
                    intentos++;
                    if (intentos==3){
                        intentos=0;
                        Lcd_Out(1,1,"3 intentos.      ");
                        Lcd_Out(2,1,"Puerta bloqueada");
                        while(1);
                    }
                    Delay_1sec(); //Espera 1 s.
                    Lcd_Cmd(_LCD_CLEAR); //Borra la pantalla y retorna el cursor
                    //al origen.
                }
            }
        }
    }
}

```



**microC®**

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO IX: PERIFÉRICOS

Los periféricos son los subsistemas que le añaden gran poder y versatilidad a un microcontrolador ya que, al estar incluidos dentro de éste, simplifican enormemente el proceso de diseño, tanto en hardware como en software, de una determinada aplicación. Entre los más importantes están los módulos PWM (Modulación de Ancho de Pulso), los convertidores analógico/digital (Convertidor A/D), los módulos de comunicación serial SSP y USART (SCI) y los comparadores.

### 9.1 MODULACIÓN DE ANCHO DE PULSO (PWM)

Este es uno de los tres posibles modos de operación del módulo CCP de los PICs 16F88, 16F628A y 16F877A, y se describe a continuación debido a su gran importancia en el campo de la automatización.

Una señal PWM es una forma de onda digital binaria de una determinada frecuencia y ciclo de trabajo (*duty cycle*) variable. Un ejemplo típico de aplicación es el control de potencia (figura 9.1). Si se considera que el nivel 0 representa OFF y el nivel 1 representa ON, la potencia que consume la carga será directamente proporcional a la duración del pulso.

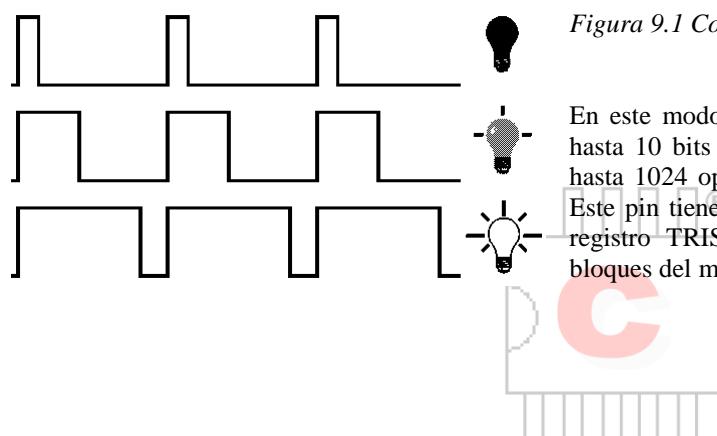


Figura 9.1 Control de potencia PWM

En este modo, el pin CCP1 produce una salida PWM de hasta 10 bits de resolución, lo que significa que se tienen hasta 1024 opciones de configuración del ciclo de trabajo. Este pin tiene que configurarse como salida por medio del registro TRISB. La figura 9.2 muestra un diagrama de bloques del módulo CCP operando en modo PWM.

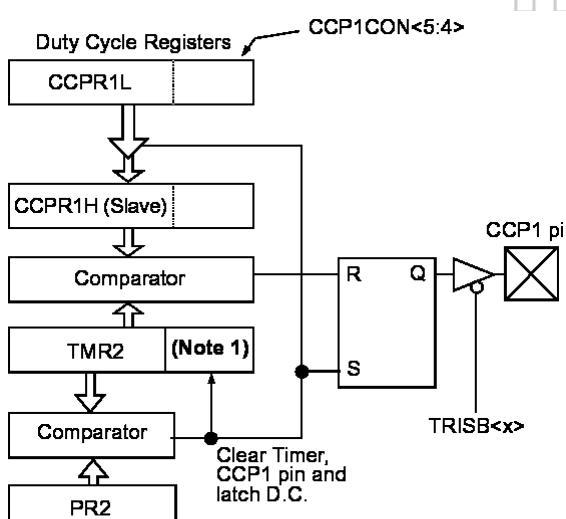


Figura 9.2 Diagrama de bloques PWM (16F88, 16F628A y 16F877A)  
Ing. Juan Ricardo Penagos Plazas

Una señal PWM (figura 9.3) se caracteriza por una base de tiempo (período) y un tiempo durante el cual la salida tiene un nivel alto (ciclo de trabajo). La frecuencia es el inverso del período.

**Note 1:** 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time base.

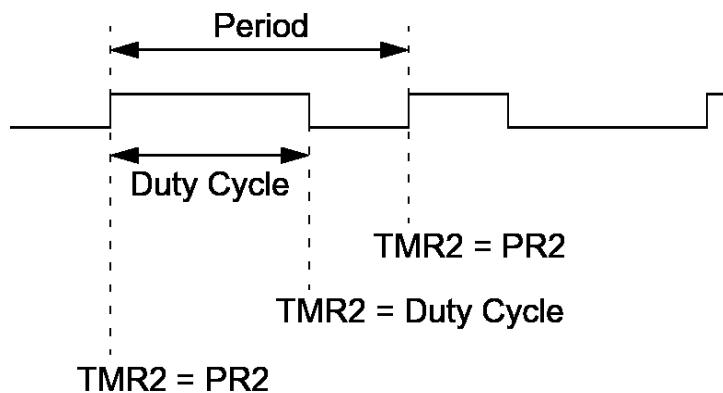


Figura 9.3 Salida PWM

**TMR2 = PR2**

### 9.1.1 Período PWM

El período se especifica escribiendo en el registro PR2 y se puede calcular con la siguiente fórmula:

$$\text{PWM Period} = [(PR2) + 1] \cdot 4 \cdot TOSC \cdot \\ (\text{TMR2 Prescale Value})$$

Cuando TMR2 es igual a PR2, se producen los tres siguientes eventos en el próximo ciclo de incremento:

- El registro TMR2 es borrado.
- El pin CCP1 se pone en 1 (excepto si el ciclo de trabajo es 0%).
- El ciclo de trabajo PWM es movido desde el registro CCPR1L al registro CCPR1H.

### 9.1.2 Ciclo de trabajo PWM

El ciclo de trabajo se especifica escribiendo en el registro CCPR1L (los 8 MSbs) y en los bits CCP1CON<5:4> (los 2 LSbs). Este valor de 10 bits se representa como CCPR1L: CCP1CON<5:4>. La siguiente fórmula permite el cálculo del ciclo de trabajo:

$$\text{PWM Duty Cycle} = (CCPR1L:CCP1CON<5:4>) \cdot \\ TOSC \cdot (\text{TMR2 Prescale Value})$$

En los bits CCPR1L: CCP1CON<5:4> se puede escribir en cualquier momento, pero el ciclo de trabajo no es movido hacia CCPR1H hasta que se produce una coincidencia entre TMR2 y PR2 (es decir, se completa el período). En el modo PWM, el registro CCPR1H es de solo lectura.

El registro CCPR1H y un depósito interno de 2 bits se emplean como doble almacenamiento del ciclo de trabajo. Cuando estos 10 bits son iguales al registro TMR2+2bits del prescaler, el pin CCP1 se pone en 0.

Para una determinada frecuencia de la señal PWM, hay una cantidad limitada de ciclos de trabajo posibles. Esta cantidad representa la resolución y se mide en bits. Por ejemplo, una resolución de 10 bits permite 1024 ciclos de trabajo diferentes, mientras que una resolución de 8 bits permite 256 ciclos de trabajo.

La máxima resolución PWM (bits) para una determinada frecuencia PWM está dada por la siguiente fórmula:

$$\text{Resolution} = \frac{\log\left(\frac{Fosc}{FPWM \times \text{TMR2 Prescaler}}\right)}{\log(2)} \text{ bits}$$

### 9.1.3 Funciones de mikroC™ para PWM

Función	Descripción
PWM1_Init(frecuencia)	Inicializa el módulo PWM con un ciclo de trabajo igual a 0. La frecuencia PWM se expresa en Hz (tomar en cuenta las frecuencias permitidas de acuerdo a la frecuencia del oscilador $F_{OSC}$ ).
PWM1_Set_Duty(ciclo de trabajo)	Establece el ciclo de trabajo desde 0 hasta 255 (resolución de 8 bits). Donde 0 es 0%, 127 es 50% y 255 es 100%. Se pueden calcular otros valores del ciclo de trabajo con la fórmula (Porcentaje*255)/100.
PWM1_Start( )	Inicia la generación de la señal PWM.
PWM1_Stop( )	Detiene la generación de la señal PWM.

Tabla 9.1 Funciones para PWM

### 9.1.4 Ejemplos de programación PWM

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

Con una frecuencia de oscilador ( $F_{OSC}$ ) de 4MHz se tiene un período ( $T_{OSC}$ ) de 0,25 us; si el prescaler del Timer2 tiene un valor de 1, entonces se pueden calcular los períodos mínimo y máximo PWM y las frecuencias máxima y mínima permitidas, respectivamente. El período mínimo se obtiene cuando el registro PR2 tiene un valor de 0, por lo tanto:

$$T_{PWMm\acute{a}n} = (0+1) \times 4 \times 0,25 \times 1 = 1 \text{ us}$$

Y la frecuencia máxima será:  $f_{PWMm\acute{a}x} = 1 \text{ MHz}$

El período máximo se obtiene cuando el registro PR2 tiene un valor de 255:

$$T_{PWMm\acute{a}x} = (255+1) \times 4 \times 0,25 \times 1 = 256 \text{ us}$$

Y la frecuencia mínima será:  $f_{PWMm\acute{a}n} = 3.906 \text{ Hz}$

En el siguiente ejemplo se trabaja con una frecuencia PWM de 5kHz.

**PWM\_1.c:** Realizar un control de velocidad de un motor de corriente continua de 12 V en cinco pasos que se pueden seleccionar por medio de un pulsador conectado en RB7. Inicialmente el motor se encuentra detenido, al pulsar la primera vez la velocidad será del 25%, la segunda el 50%, la tercera el 75% y la cuarta el 100%. Si se vuelve a presionar, el motor se detiene. La velocidad actual se muestra en el LCD (figuras 9.4.1 y 9.4.2). **LCD**  **Button**  **PWM**  **4MHz**

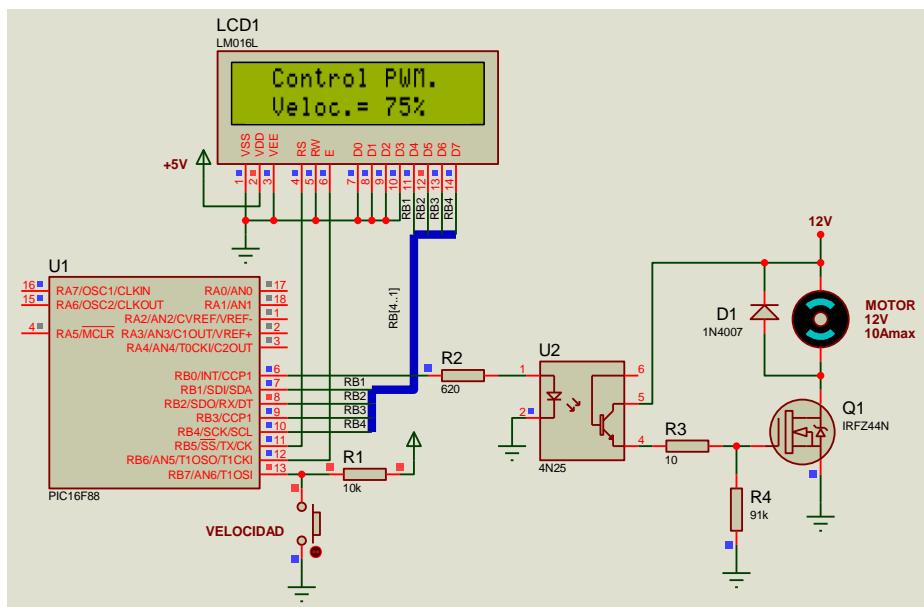


Figura 9.4.1 Circuito del problema PWM\_1.c (PIC16F88 y 16F628A)

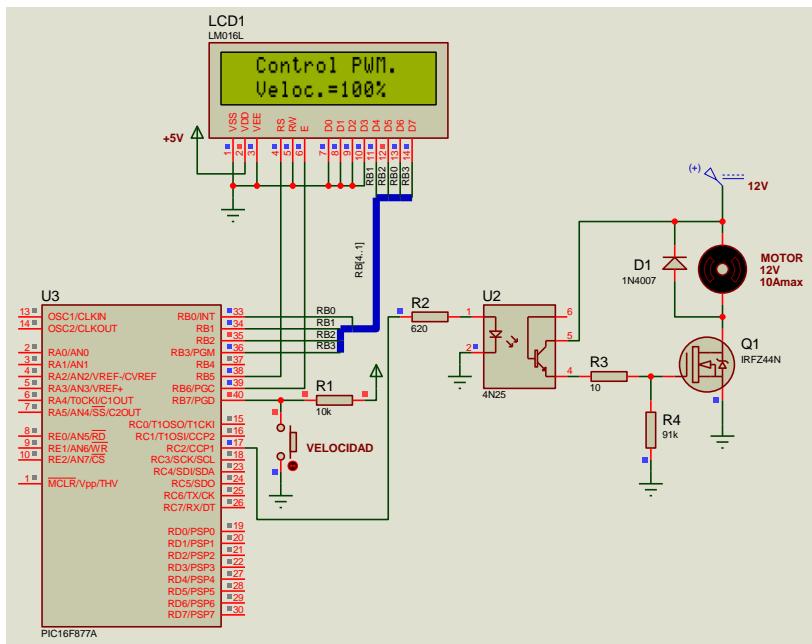


Figura 9.4.2 Circuito del problema PWM\_1.c (PIC16F877A)

```


//PWM_1.c
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RB5_bit;
sbit LCD_EN at RB6_bit;
sbit LCD_D4 at RB1_bit;
sbit LCD_D5 at RB2_bit;
sbit LCD_D6 at RB3_bit;
sbit LCD_D7 at RB4_bit;

sbit LCD_RS_Direction at TRISB5_bit;
sbit LCD_EN_Direction at TRISB6_bit;
sbit LCD_D4_Direction at TRISB1_bit;
sbit LCD_D5_Direction at TRISB2_bit;
sbit LCD_D6_Direction at TRISB3_bit;
sbit LCD_D7_Direction at TRISB4_bit;
// Fin de declaración de variables de conexión.
char contador=0,estado=1;

void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00;             //Inicialización.
ANSEL=0x00;              //Bits AN6:AN0 como E/S digital.
TRISB0_bit=0;            //RB0 como salida.
Lcd_Init();              //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);    //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,3,"Control PWM.");
PWM1_Init(5000);         //Frecuencia PWM.
PWM1_Start();
while (1){
  if (Button(&PORTB,7,1,0)) estado=0; //Si se pulsa.
  if (estado==0 && Button(&PORTB,7,1,1)){ //Si se pulsa y se libera.
    contador++;
    if (contador>4) contador=0;
    estado=1;
  }
  switch (contador){
  case 0:
    Lcd_Out(2,3,"Veloc.= 0%");
    PWM1_Set_Duty(0);
    break;
  case 1:
    Lcd_Out(2,3,"Veloc.= 25%");
    PWM1_Set_Duty(64);
    break;
  case 2:
    Lcd_Out(2,3,"Veloc.= 50%");
    PWM1_Set_Duty(128);
    break;
  case 3:
    Lcd_Out(2,3,"Veloc.= 75%");
    PWM1_Set_Duty(192);
    break;
  case 4:
    Lcd_Out(2,3,"Veloc.= 100%");
    PWM1_Set_Duty(255);
    break;
  }
}
}

```



**microC®**

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

```

Lcd_Out(2,3,"Veloc.= 50%");  

PWM1_Set_Duty(127);  

break;  

case 3:  

Lcd_Out(2,3,"Veloc.= 75%");  

PWM1_Set_Duty(191);  

break;  

case 4:  

Lcd_Out(2,3,"Veloc.=100%");  

PWM1_Set_Duty(255);  

}  

}
}
}

```

## 9.2 CONVERTIDOR A/D DE 7 CANALES (PIC16F88 Y 16F877A)

El convertidor Analógico/Digital (A/D) tiene 7 entradas en el PIC16F88 y 8 entradas en el PIC16F877A. La señal analógica de entrada carga un capacitor de muestreo y retención durante el período de adquisición. La salida de este capacitor es la entrada al convertidor. El convertidor genera un resultado digital de este nivel analógico por medio de aproximaciones sucesivas. La conversión de una señal analógica de entrada da como resultado un número de 10 bits. La figura 9.5 muestra la función de transferencia que permite determinar el número resultante de la conversión en función del voltaje analógico de entrada ( $1LSb=5/1024=4,88[mV]$ ). Por ejemplo, la salida será 002h para un intervalo del voltaje de entrada entre 7,32 mV (1,5LSb) y 12,2 mV (2,5LSb). El máximo voltaje analógico de entrada es 1.023,5 LSb=4,998V (5V aprox.).

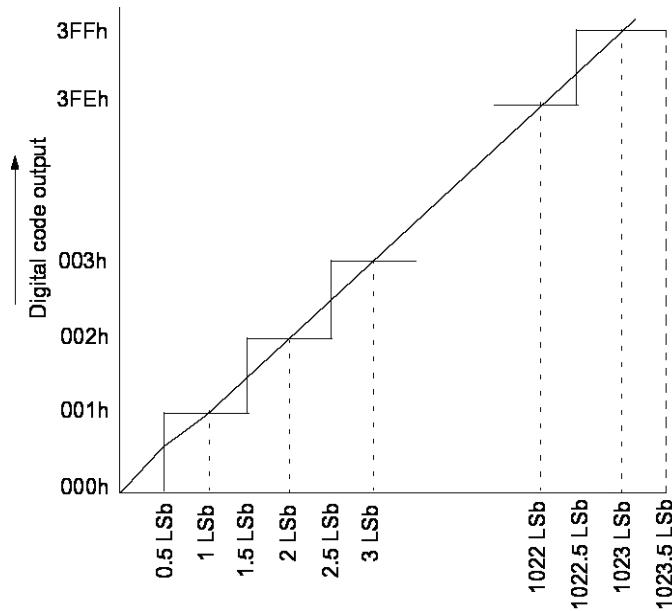


Figura 9.5 Función de transferencia del conversor A/D

La figura 9.6 muestra la secuencia de conversión y los términos empleados. El período de adquisición (aproximadamente 20 us como mínimo) es el tiempo durante el cual el capacitor de muestreo y retención está conectado al nivel externo de voltaje de la señal analógica. A continuación está el período de conversión de  $12T_{AD}$  ( $T_{AD}$  = tiempo de conversión por cada bit), que comienza cuando el bit GO se pone en 1. La suma de estos dos tiempos es el período de muestreo. Existe un período mínimo de adquisición para asegurar que el capacitor de muestreo y retención se cargue a un nivel apropiado para realizar la conversión con la precisión deseada.

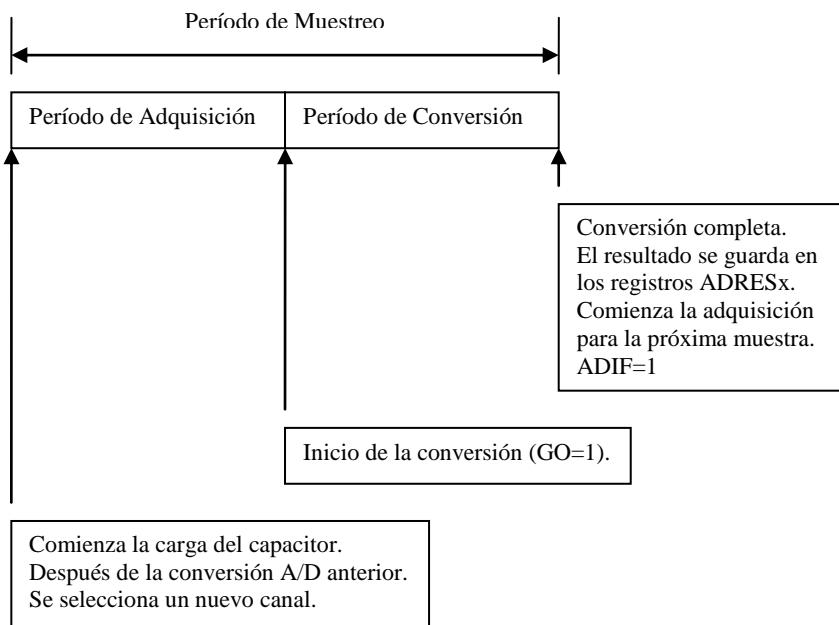
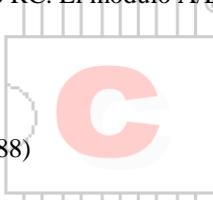


Figura 9.6 Secuencia de conversión A/D

Este módulo tiene la característica particular de poder operar en modo *sleep*, para lo cual el reloj de conversión debe provenir del oscilador interno RC. El módulo A/D tiene los siguientes registros:

- ADRESH
- ADRESL
- ADCON0
- ADCON1
- ANSEL (únicamente para el PIC16F88)



Los registros ADRESH y ADRESL contienen los 10 bits resultantes de la conversión. El registro ADCON0 controla la operación del módulo A/D. Los registros ADCON1 y ANSEL (figura 9.7) configuran el funcionamiento de los pines del conversor.

◆ Por defecto los pines AN<7:0> del PIC16F877A están configurados como entradas analógicas, por lo tanto no es necesario programar el registro ADCON1 cuando se va a emplear el conversor A/D.

#### ANSEL: ANALOG SELECT REGISTER (ADDRESS 9Bh) PIC16F88 DEVICES ONLY

U-0	R/W-1						
	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
bit 7							bit 0

bit 7      **Unimplemented:** Read as '0'

bit 6-0      **ANS<6:0>:** Analog Input Select bits

Bits select input function on corresponding AN<6:0> pins.

1 = Analog I/O<sup>(1,2)</sup>

0 = Digital I/O

**Note 1:** Setting a pin to an analog input disables the digital input buffer. The corresponding TRIS bit should be set to input mode when using pins as analog inputs. Only AN2 is an analog I/O, all other ANx pins are analog inputs.

Figura 9.7 Bits del registro ANSEL (PIC16F88)

Cuando se completa la conversión A/D, el resultado se guarda en el par de registros ADRESx, el bit GO/#DONE del registro ADCON0 se pone en 0 y la bandera de interrupción A/D (bit ADIF del registro PIR1) se pone en 1.

Después de la configuración del módulo A/D, el paso siguiente es la *adquisición* del canal seleccionado, antes del comienzo de la conversión. Los canales analógicos deben tener los bits correspondientes del registro

TRISx con un valor de 1 (configurados como entradas). Después de que ha transcurrido el tiempo de adquisición se puede iniciar la conversión.

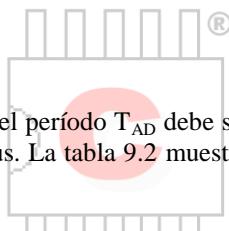
Los pasos a seguir para realizar una conversión A/D son los siguientes:

1. Configurar el módulo A/D.
2. Configurar la interrupción A/D (opcional).
3. Esperar a que transcurra el tiempo de adquisición (20 us).
4. Iniciar la conversión.
5. Esperar el final de la conversión.
6. Leer el resultado de los registros ADRESx.
7. Para la próxima conversión, ir al paso 1 o 2 dependiendo de lo requerido. Se debe esperar un mínimo de  $2T_{AD}$  antes de comenzar la siguiente adquisición.

### 9.2.1 Selección del reloj de conversión

El tiempo de conversión por cada bit se define como  $T_{AD}$ . La conversión A/D requiere un tiempo de  $12T_{AD}$  para un resultado de 10 bits. La fuente del reloj de conversión se puede seleccionar por software por medio de los bits ADCS2 del registro ADCON1 y ADCS<1:0> del registro ADCON0. Las siete posibles opciones para  $T_{AD}$  son las siguientes:

- $2T_{OSC}$
- $4T_{OSC}$
- $8T_{OSC}$
- $16T_{OSC}$
- $32T_{OSC}$
- $64T_{OSC}$
- Oscilador interno RC (2-6 us)



Para que la conversión sea correcta, el período  $T_{AD}$  debe seleccionarse lo más pequeño que sea posible, pero no inferior a 1,6 us y no superior a 6,4 us. La tabla 9.2 muestra las frecuencias máximas del oscilador para los diferentes valores de  $T_{AD}$  permitidos.

AD Clock Source (TAD)			Maximum Device Frequency
Operation	ADCS<2>	ADCS<1:0>	Max.
2 Tosc	0	00	1.25 MHz
4 Tosc	1	00	2.5 MHz
8 Tosc	0	01	5 MHz
16 Tosc	1	01	10 MHz
32 Tosc	0	10	20 MHz
64 Tosc	1	10	20 MHz
RC <sup>{1,2,3}</sup>	x	11	(Note 1)

**Note 1:** The RC source has a typical TAD time of 4  $\mu$ s, but can vary between 2-6  $\mu$ s.

**2:** When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for Sleep operation.

Tabla 9.2  $T_{AD}$  vs. Frecuencias máximas de operación

### 9.2.2 Registros de resultado

El par de registros ADRESx es el lugar donde se almacenan los 10 bits resultantes al final de la conversión. Estos dos registros tienen un ancho total de 16 bits. Se tiene la posibilidad de alinear a la izquierda o la derecha los 10 bits del resultado dentro de este espacio de 16 bits. El bit ADFM del registro ADCON1 controla la alineación. En la figura 9.8 se muestran las dos posibilidades de alineación. Los bits restantes toman un valor de 0.

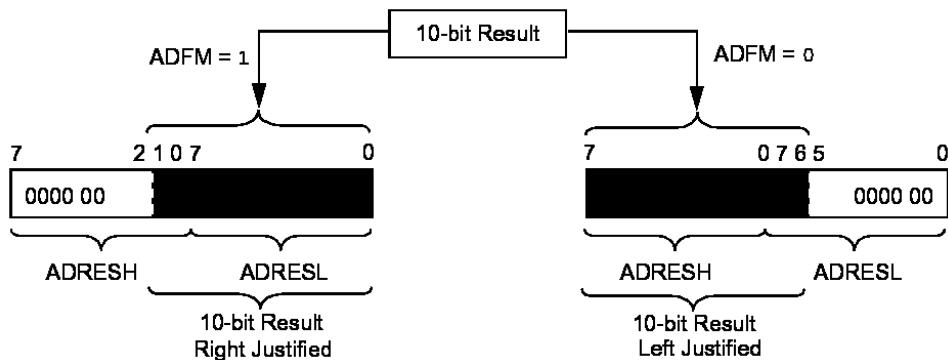


Figura 9.8 Alineación del resultado de la conversión A/D

### 9.2.3 Funciones de mikroC™ para conversión A/D

Función	Descripción
ADC_Read(canal)	Inicializa el módulo A/D para operar con el oscilador interno RC. Retorna un número de 10 bits como resultado de la conversión.

Tabla 9.3 Funciones para conversión A/D

### 9.2.4 Ejemplos de programación de conversión A/D

Estos ejemplos corresponden al PIC16F88. El código fuente para el PIC16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

En este ejemplo debe tomarse en cuenta que el máximo voltaje de entrada al conversor A/D es 5V, por lo tanto, si se desea incrementar el intervalo de medición se debe añadir un divisor de voltaje. Para el problema planteado, el divisor de voltaje da como resultado  $V_T=V_{IN}/3$ , por lo tanto  $V_T=3V_{IN}$  ( $V_T$ =voltaje medido,  $V_{IN}$ =voltaje de entrada al conversor A/D).

**AD\_1.c:** Voltímetro digital desde 0 a 15 V<sub>CD</sub>. El resultado se muestra en el LCD (figuras 9.9.1 y 9.9.2). **ADC**✓. **C\_String**✓. **Conversions**✓. **LCD**✓.

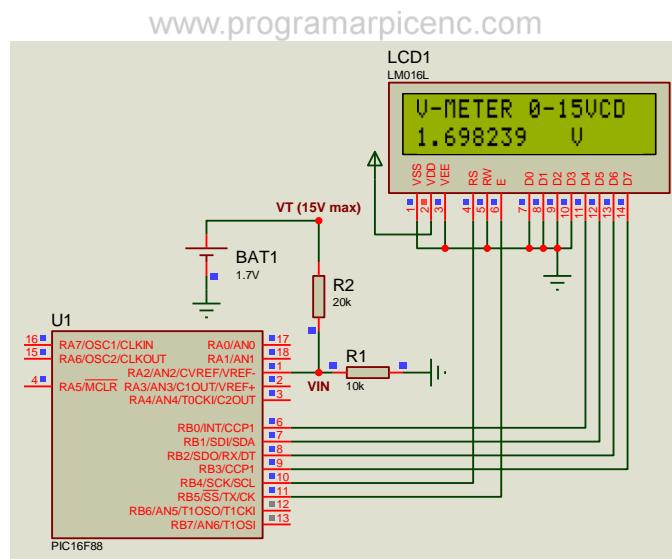


Figura 9.9.1 Circuito del problema AD\_1.c (PIC16F88)

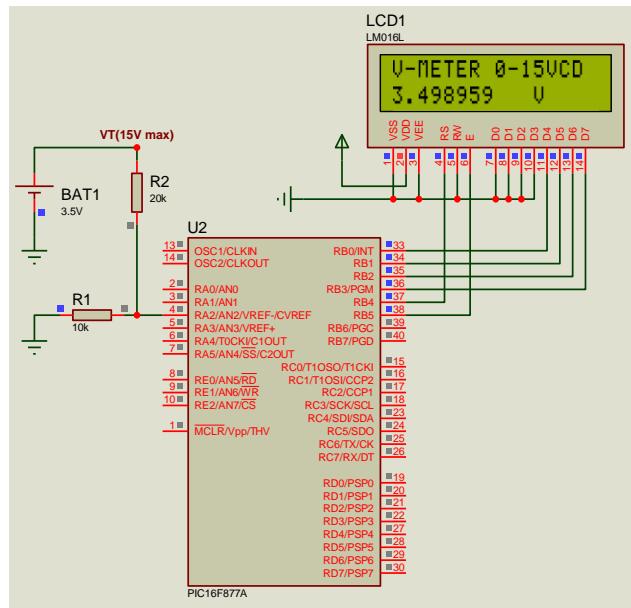


Figura 9.9.2 Circuito del problema AD\_1.c (PIC16F877A)

```

//AD_1.c
//Declaración de las 12 variables necesarias para la conexión
//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;
// Fin de declaración de variables de conexión.
int temp_res;
float voltaje;
char txt[15];

void main(){
OSCCON=0x40; //Oscilador interno a 1MHz (TCl=4 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTA=0x00; //Inicialización.
ANSEL=0x04; //Bit AN2 como entrada analógica.
TRISA=0x04; //RA2 como entrada.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,1,"V-METER 0-15VCD");
Lcd_Out(2,12,"V");
do{
    temp_res=ADC_Read(2); //Obtiene 10 bits de la conversión del canal 2.
    voltaje=(float)(temp_res*0.00488); //Transforma el número de 10 bits al
    //voltaje correspondiente.
    voltaje*=3.0; //VT=3VIN.
    FloatToStr(voltaje,txt);
    Lcd_Out(2,1,txt);
} while(1);
}

```

## CAPÍTULO X: COMUNICACIÓN CON EL ORDENADOR

La comunicación del PIC con el ordenador es de gran importancia y para esto se ha incorporado un módulo con las características apropiadas para el intercambio de información. Este módulo es conocido como USART (*Universal Synchronous Asynchronous Receiver Transmitter*) y se estudia en este capítulo. El módulo USART opera de acuerdo al estándar RS232 que también es muy utilizado en las computadoras personales.

Este es uno de los dos módulos serie E/S, también llamado Interfaz de Comunicación Serie SCI. El módulo USART puede configurarse como un sistema asincrónico full-dúplex que puede comunicarse con dispositivos periféricos, tales como terminales CRT y computadoras personales, o puede configurarse como un sistema sincrónico half-dúplex que puede comunicarse con otros dispositivos tales como conversores A/D y D/A, EEPROMs serie, etc.

Los modos de operación disponibles son los siguientes:

- Asincrónico (full-dúplex)
- Sincrónico-Maestro (half-dúplex)
- Sincrónico-Esclavo (half-dúplex)

### 10.1 ESTÁNDAR RS232

Este es un protocolo de comunicación serie ampliamente difundido en las computadoras personales y empleado por los puertos COM del ordenador. El acceso físico a estos puertos es a través de un conector DB-25 o DB-9, machos y hembras. La norma RS232 se estableció para comunicar un ordenador con un modem, por lo que en los conectores DB-25 aparecen muchos pines que en otras aplicaciones no se utilizan, y en las que es más común emplear el conector DB-9. Cada uno de los terminales del conector RS232 tiene una función especificada por la norma. Unos pines transmiten y reciben datos, mientras que otros permiten el control de la comunicación. En la tabla 10.1 se observan estos detalles.

PIN	SEÑAL
1	<i>Data Carrier Detect (DCD)</i>
2	<i>Received Data (RxD)</i>
3	<i>Transmitted Data (TxD)</i>
4	<i>Data Terminal Ready (DTR)</i>
5	<i>Signal Ground (SG)</i>
6	<i>Data Set Ready (DSR)</i>
7	<i>Request to Send (RTS)</i>
8	<i>Clear to Send (CTS)</i>
9	<i>Ring Indicator (RI)</i>

Tabla 10.1 Funciones de los pines del conector DB-9 con estándar RS232

Un dato a tener muy en cuenta es la velocidad de transmisión, que es la cantidad de información enviada por la línea de transmisión en la unidad de tiempo. Se mide en Baudios y es proporcional a los Bits/segundo (bps). Las velocidades de transmisión normalizadas para los puertos COM son: 75, 150, 300, 600, 1200, 2400, 4800, 9600, etc. Baudios.

Otra cuestión fundamental se refiere a los niveles de voltaje de la norma:

- Los datos se transmiten con lógica negativa, es decir, un voltaje positivo representa 0, mientras que un voltaje negativo representa 1.
- El 0L se encuentra entre +3 y +15V.
- El 1L se encuentra entre -3 y -15V.
- Los voltajes más usados son +12V para el 0L y -12V para el 1L.
- Cuando un puerto no está transmitiendo mantiene el terminal de transmisión en 1L (-12V).

### 10.2 CIRCUITO INTEGRADO MAX232

Este IC es un estándar en la industria y se emplea como interfaz entre los niveles TTL y RS232 y requiere únicamente una fuente de +5V para su operación. Para generar los niveles de +12V y -12V necesita 4 capacitores de 1,0 uF. Dispone de dos entradas TTL con salida RS232, así como dos entradas RS232 con salida TTL (figuras 10.1 y 10.2). Puede realizar la transferencia de datos a una velocidad máxima de 120 kbps.

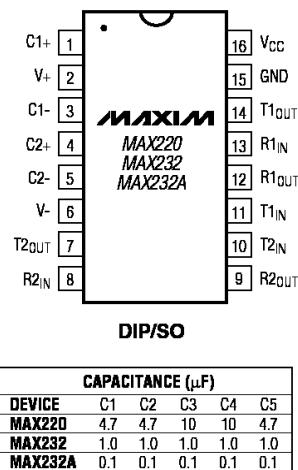


Figura 10.1 Distribución de terminales y valores de los capacitores

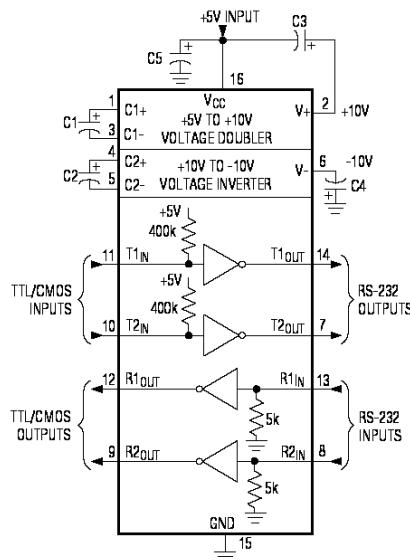


Figura 10.2 Circuito típico de operación (note la polaridad de los capacitores)

### 10.3 CONEXIÓN DEL PIC AL ORDENADOR

Para conectar los PICs 16F88, 16F628A o 16F877A a la computadora personal únicamente se requieren dos líneas, una para el envío de datos (Tx) y otra para la recepción de datos (Rx), como se muestra en las figuras 10.3, 10.4 y 10.5 (no se muestra la polarización de los PICs). Como se explicó, además es necesario tener un dispositivo que realice la conversión de los niveles de voltaje TTL a los niveles RS232 y viceversa (esta función es desempeñada por el MAX232).

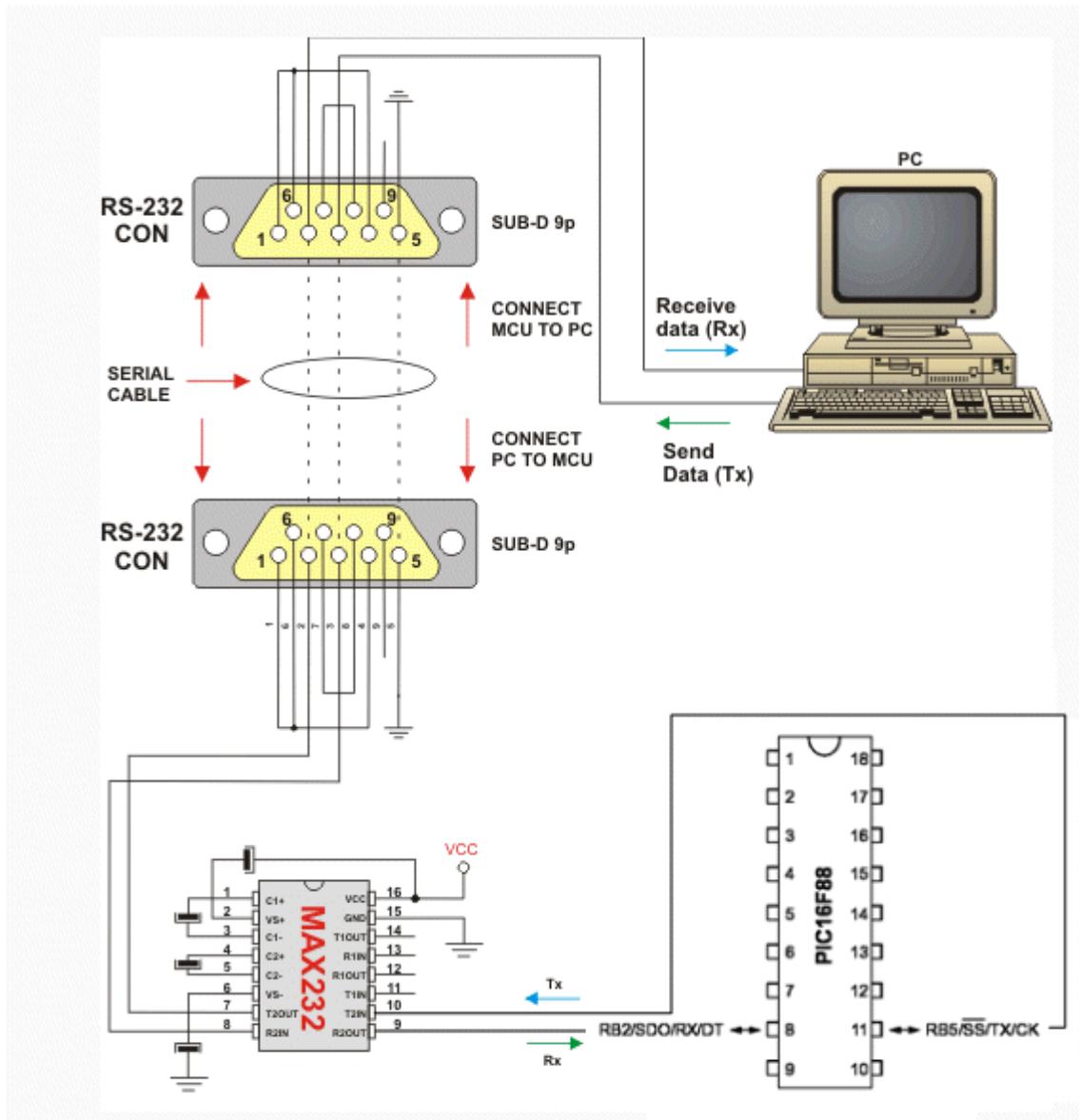


Figura 10.3 Conexión del PIC16F88 al ordenador

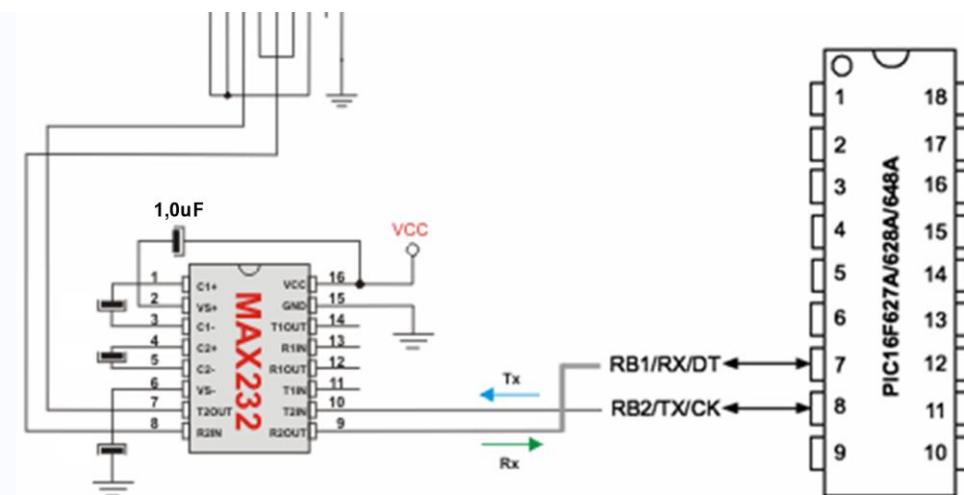


Figura 10.4 Detalle de la conexión del PIC16F628A al ordenador

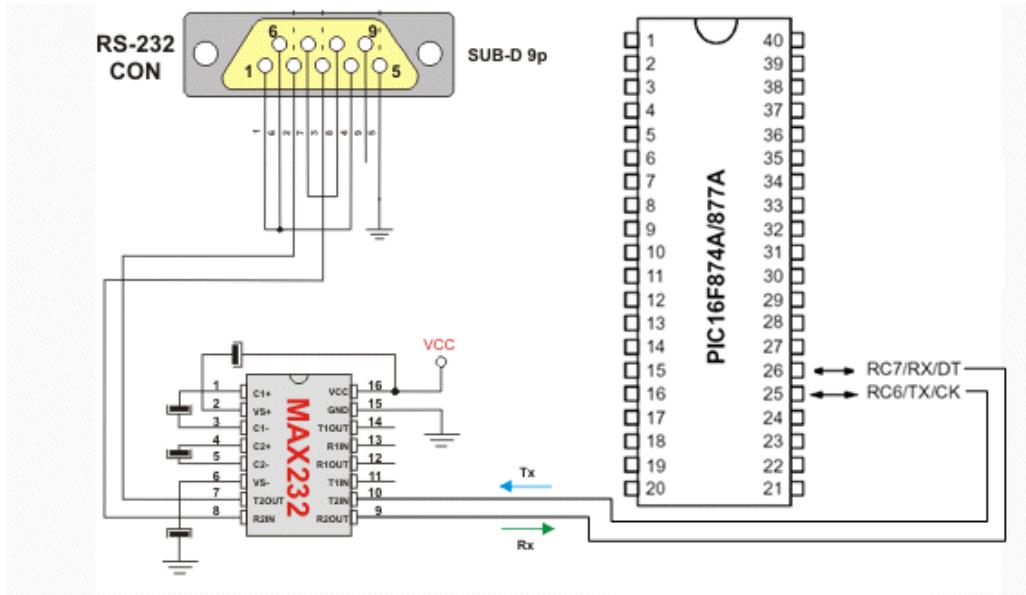


Figura 10.5 Detalle de la conexión del PIC16F877A al ordenador

Para el control de la comunicación se necesita, por una parte, programar el microcontrolador y, por otro lado, un programa que se ejecute en el ordenador y que realice la gestión correspondiente; para esto mikroC™ dispone de la herramienta *USART Terminal* del menú *Tools*, que permite la comunicación a través del puerto COM empleando el protocolo RS232 (figura 10.6).

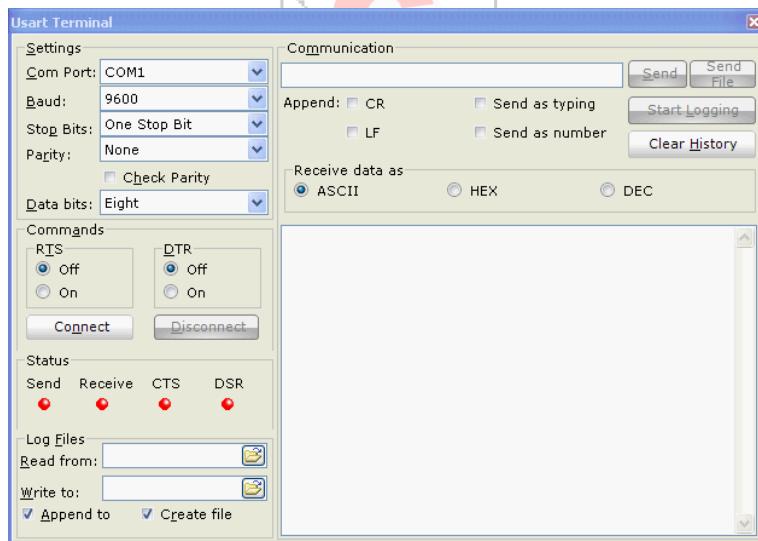


Figura 10.6 Ventana de usuario de la herramienta USART Terminal

## 10.4 FUNCIONES DE mikroC™ PARA EL MÓDULO USART

El módulo USART se incluye en los PICs 16F88, 16F628A y 16F877A (algunos PICs, como el 16F84A, no tienen este módulo). La biblioteca *UART* de mikroC™ proporciona las herramientas para hacer que el empleo de este módulo en modo asincrónico (full-dúplex) sea tan sencillo como nunca lo había sido hasta ahora. Esto permite la comunicación con otros dispositivos a través del protocolo RS232. En la tabla 10.2 se pueden ver las funciones incorporadas con sus características más sobresalientes.

Función	Descripción
UART1_Init(velocidad de transmisión)	Inicializa el módulo USART con la velocidad de transmisión especificada. La velocidad de transmisión está limitada por la frecuencia del oscilador FOSC.
UART1_Data_Ready()	Permite detectar si hay un dato de entrada listo para su lectura.
UART1_Tx_Idle()	Permite detectar si el registrador de desplazamiento de transmisión se encuentra vacío o no.
UART1_Read()	Recibe un byte. Primero emplear la función UART1_Data_Ready para comprobar si el dato está listo.
UART1_Read_Text(salida, delimitador, intentos)	Lee caracteres de entrada hasta que se detecta una secuencia de finalización. Los caracteres se almacenan en “salida”, la secuencia de finalización se almacena en “delimitador”. Si el delimitador no se encuentra, el procedimiento finaliza. El parámetro “intentos” define el número de caracteres recibidos en el que se espera llegue el delimitador. Si “intentos” es igual a 255, la función intentará detectar continuamente el delimitador.
UART1_Write(data)	Transmite un byte (dato) a través del módulo USART.
UART1_Write_Text(texto)	Envía el texto a través del módulo USART.

Tabla 10.2 Funciones para el módulo USART

## 10.5 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.

**USART\_01.c:** Lo que escriba por el teclado del ordenador se visualizará en el módulo LCD y será enviado de vuelta al ordenador, por lo que se visualizará también en el monitor. Cuando se hayan ingresado 16 caracteres el LCD se borrará y empezará nuevamente en la primera fila y primera columna (figuras 10.7.1, 10.7.2 y 10.7.3). **LCD** **UART** **4MHz**.

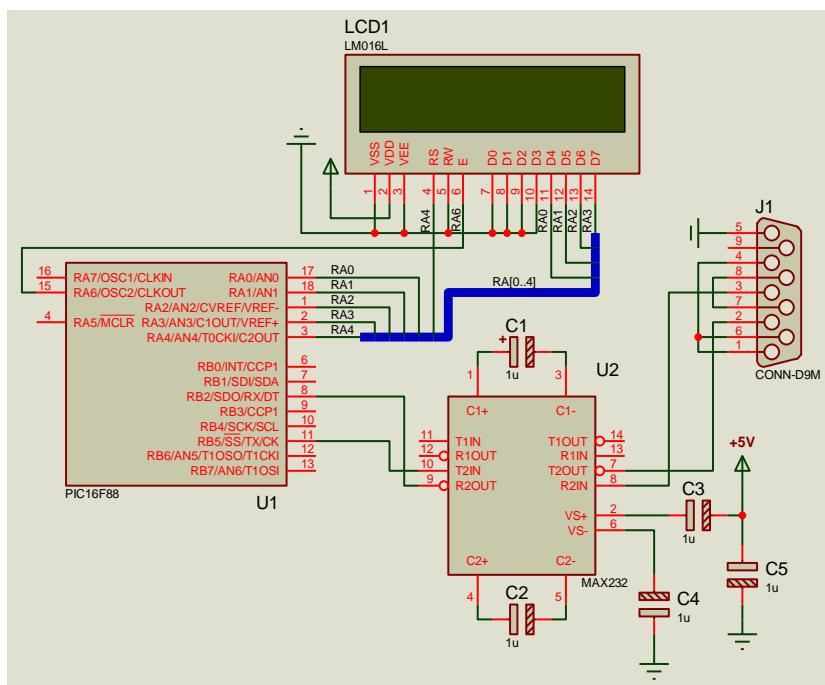


Figura 10.7.1 Circuito del problema USART\_01.c (PIC16F88)

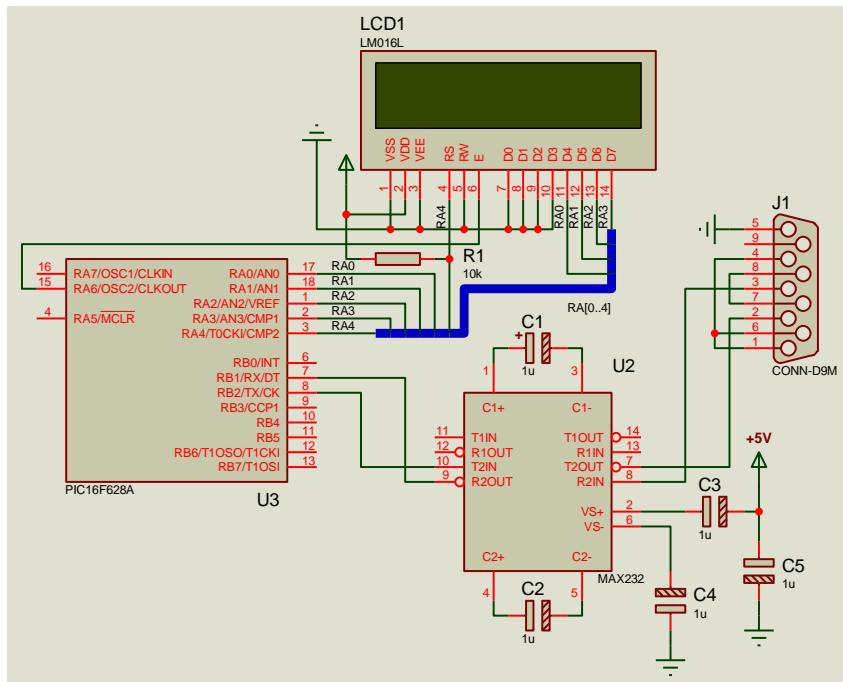


Figura 10.7.2 Circuito del problema USART\_01.c (PIC16F628A)

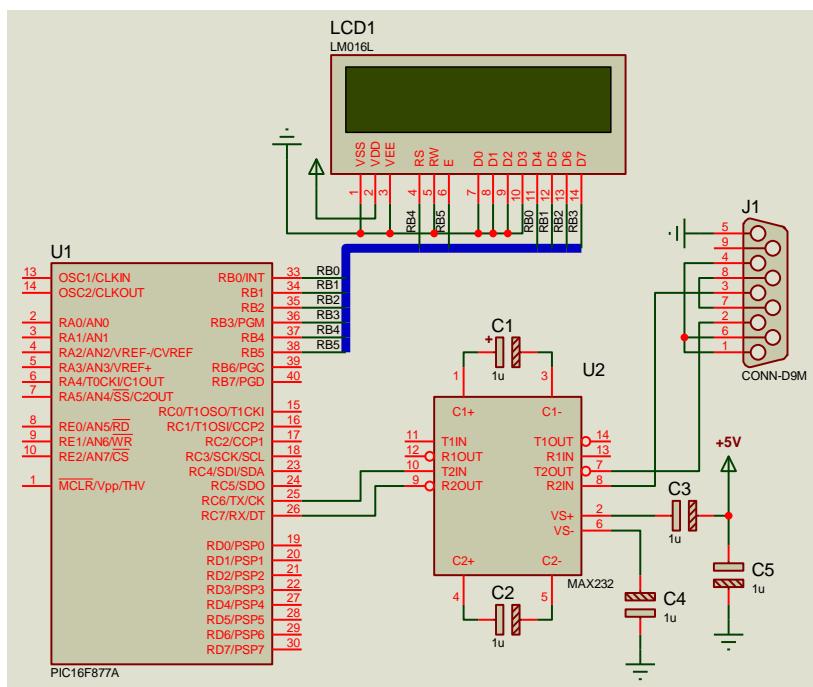


Figura 10.7.3 Circuito del problema USART\_01.c (PIC16F877A)

```


//USART_01.c
//Declaración de las 12 variables necesarias para la conexión
//del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;
sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.

```

```

char uart_rd, contador=0;
void main() {
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
Lcd_Init();              //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);    //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

UART1_Init(9600);       //Inicializa el USART a 9600 bps.
Delay_ms(100);           //Espera a que el USART se estabilice.
UART1_Write_Text("Hola."); //Envía este texto a través del USART.

while (1){
if (UART1_Data_Ready()){ //Si se recibe un dato,
uart_rd = UART1_Read(); //lee el dato recibido,
Lcd_Ch(chr_cp(uart_rd)); //lo presenta en el LCD,
UART1_Write(uart_rd); //y lo envía a través del USART.

contador++;
if(contador==16){ //Detecta el ingreso de 16 caracteres,
contador=0; //reinicia el contador,
Delay_1sec(); //espera 1 segundo,
Lcd_Cmd(_LCD_CLEAR); //y borra el LCD.
}
}
}
}

```

**USART\_02.c:** Envía un mensaje grabado en la memoria de programa del PIC al ordenador y se muestra en el monitor (figuras 10.8.1, 10.8.2 y 10.8.3). *UART*  4MHz .

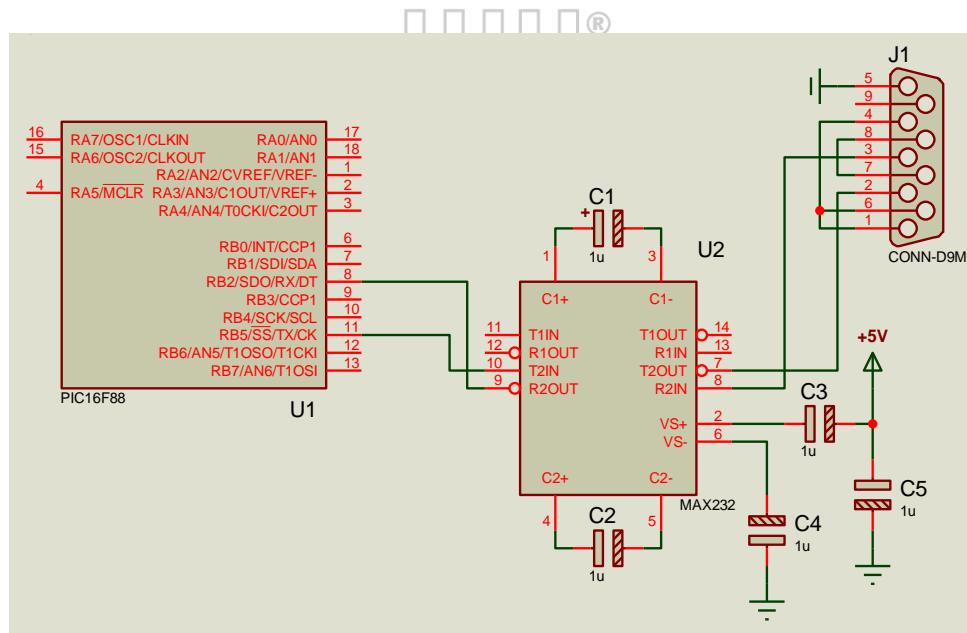


Figura 10.8.1 Circuito del problema USART\_02.c (PIC16F88)

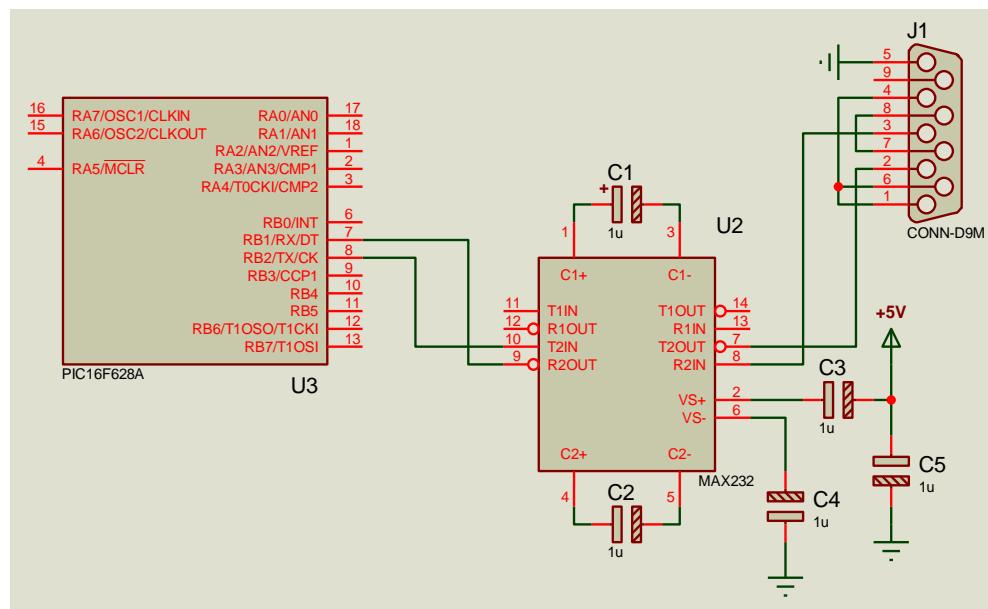


Figura 10.8.2 Circuito del problema USART\_02.c (PIC16F628A)

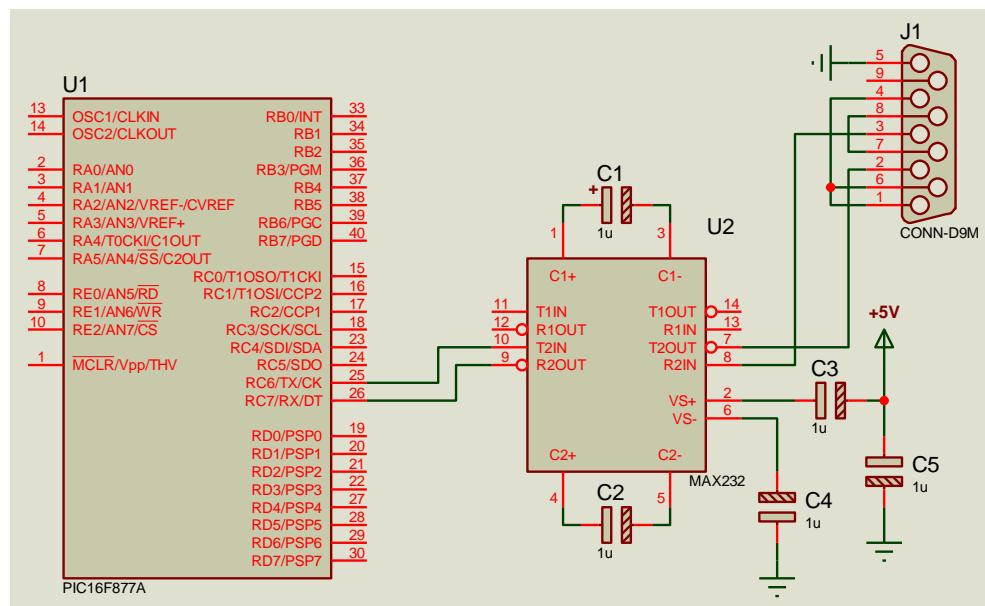


Figura 10.8.3 Circuito del problema USART\_02.c (PIC16F877A)

```


//USART_02.c
void main(){
    OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    UART1_Init(9600);      //Inicializa el USART a 9600 bps.
    Delay_ms(100);          //Espera a que el USART se estabilice.
    UART1_Write_Text("Hola. Este es uno de mis primeros ejemplos con RS232.");
    UART1_Write(13);         //Mueve el cursor a la siguiente línea.
                           //13 es el código ASCII de CR (retorno de carro).
    UART1_Write_Text("Con las funciones de mikroC es muy facil.");
    UART1_Write(13);
    UART1_Write_Text("Así mi aprendizaje es muy eficiente.");
}


```

En el siguiente ejemplo se emplean los *pull-ups* del puerto B en conjunto con 4 interruptores para obtener los niveles lógicos de entrada, además nótese como se emplea el código ASCII de control BS (*backspace*) para borrar el monitor y así poder actualizar los valores.

**USART\_03.c:** Sistema de monitoreo de los estados de los pines RB<7,6,4,3> (PIC16F88) o RB<7:4> (PIC16F628A y 16F877A) configurados como entradas. Se lee el estado de estos pines y se envía al ordenador, actualizando la lectura cada 500ms (figuras 10.9.1, 10.9.2 y 10.9.3). *UART*  4MHz .

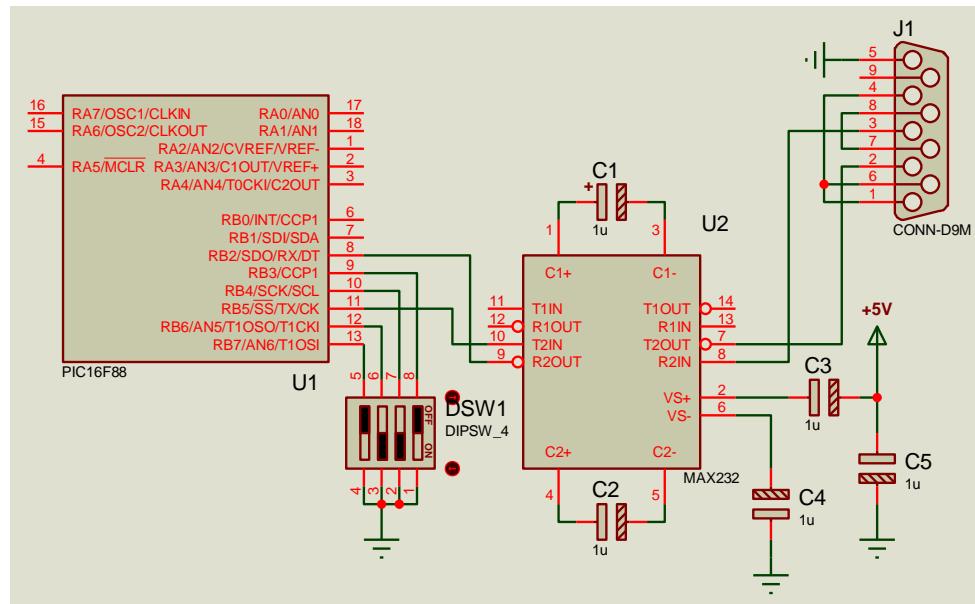


Figura 10.9.1 Circuito del problema USART\_03.c (PIC16F88)

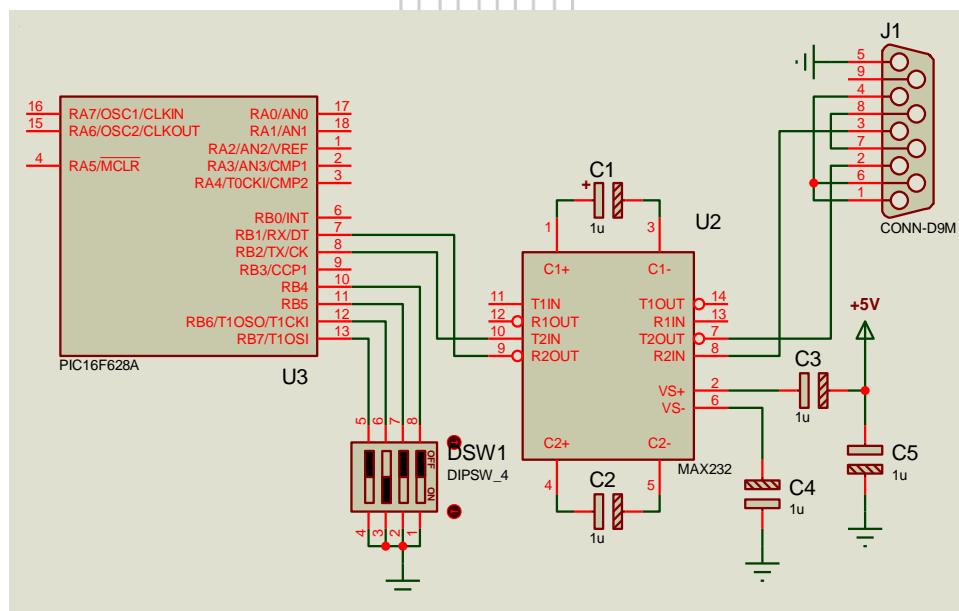


Figura 10.9.2 Circuito del problema USART\_03.c (PIC16F628A)

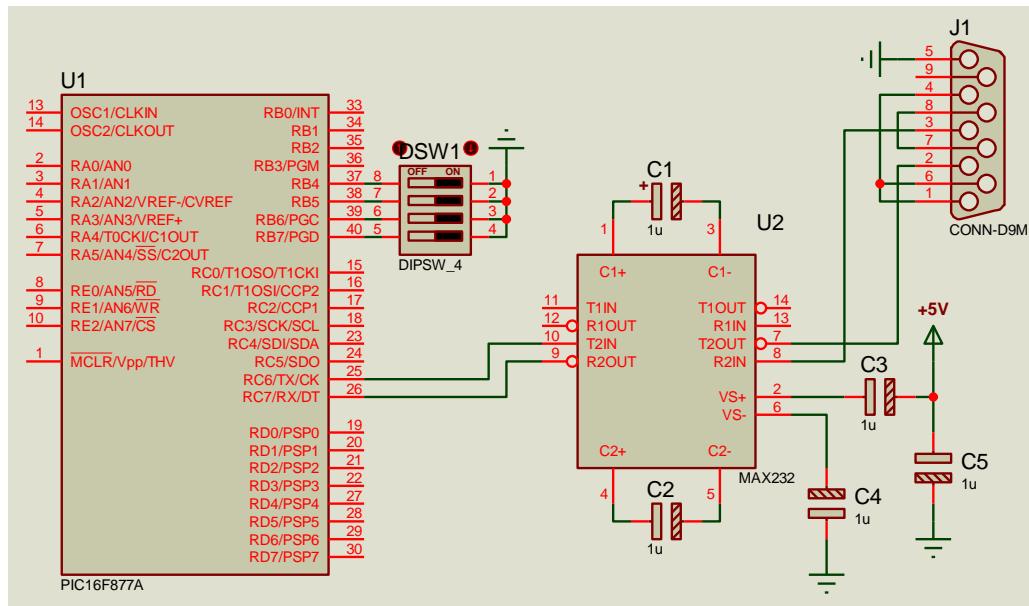


Figura 10.9.3 Circuito del problema USART\_03.c (PIC16F877A)

```

 //USART_03.c
void main() {
    OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
    while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
    ANSEL=0x00;             //Bits AN6:AN0 como E/S digital.
    TRISB7_bit=1;TRISB6_bit=1; //Pines RB<7:6> como entradas.
    TRISB4_bit=1;TRISB3_bit=1; //Pines RB<4:3> como entradas.
    NOT_RBPU_bit=0;         //Pull-ups habilitados.
    UART1_Init(9600);       //Inicializa el USART a 9600 bps.
    Delay_ms(100);          //Espera a que el USART se estabilice.

    UART1_Write_Text("Monitoreo de los bits RB<7,6,4,3>.");
    UART1_Write(13);         //Mueve el cursor a la siguiente línea.

    while (1){
        if (RB7_bit==0)
            UART1_Write_Text("0");
        else
            UART1_Write_Text("1");

        if (RB6_bit==0)
            UART1_Write_Text("0");
        else
            UART1_Write_Text("1");

        if (RB4_bit==0)
            UART1_Write_Text("0");
        else
            UART1_Write_Text("1");

        if (RB3_bit==0)
            UART1_Write_Text("0");
        else
            UART1_Write_Text("1");
        Delay_ms(500);           //Esperar 500 ms.
        UART1_Write(8);          //Retornar el cursor.
        UART1_Write(8);          //8 es el código ASCII de BS (backspace).
        UART1_Write(8);
        UART1_Write(8);
    }
}

```

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## CAPÍTULO XI: BUS I<sup>2</sup>C

El bus I<sup>2</sup>C (*Inter-Integrated Circuit*) o Interconexión de Circuitos Integrados es un interfaz serie de dos conductores. La especificación original, o modo Estándar, es para velocidades de transmisión de hasta 100kbps. También se ha desarrollado una especificación mejorada, o modo Rápido (400 kbps). Los dispositivos con modo Estándar o Rápido pueden operar en el mismo bus, si el bus opera a la velocidad del dispositivo más lento. El interfaz I<sup>2</sup>C emplea un protocolo para asegurar la transferencia confiable de los datos. Cuando se transmiten datos, uno de los dispositivos es el maestro, el cual inicia la transferencia en el bus y genera las señales de reloj para permitir esa transferencia, mientras que los otros dispositivos se comportan como esclavos. La tabla 11.1 define algunos de los términos I<sup>2</sup>C. En el protocolo I<sup>2</sup>C cada dispositivo tiene una dirección. Cuando el maestro desea iniciar la transferencia de datos, primero transmite la dirección del dispositivo con el cual desea comunicarse. Todos los dispositivos están atentos para determinar si es su dirección. Dentro de la dirección, un bit especifica si el maestro desea leer o escribir al esclavo. El maestro y el esclavo siempre están operando en modos complementarios (transmisor/receptor) durante una transferencia de datos. Es decir, pueden operar en cualquiera de los siguientes modos:

- Maestro-transmisor y Esclavo-receptor
- Esclavo-Transmisor y Maestro-receptor

En los dos casos el maestro genera la señal de reloj. Se emplean resistores externos de *pull-up* para asegurar un nivel alto cuando ninguno de los dispositivos lleva las líneas a un nivel bajo. Un valor de 4k7 es satisfactorio para la mayoría de las aplicaciones.

Término	Descripción
Transmisor	El dispositivo que envía datos al bus.
Receptor	El dispositivo que recibe datos del bus.
Maestro	El dispositivo que inicia la transferencia, genera el reloj y finaliza la transferencia.
Esclavo	El dispositivo seleccionado por el maestro a través de una dirección.

Tabla 11.1 Terminología del bus I<sup>2</sup>C

### 11.1 CARACTERÍSTICAS DEL BUS I<sup>2</sup>C

Se ha definido el siguiente **protocolo de bus**:

- La transferencia de datos se puede iniciar únicamente cuando el bus no está ocupado.
- Durante la transferencia de datos, la línea de datos tiene que permanecer estable siempre que la línea de reloj esté en nivel ALTO. Si se producen cambios en la línea de datos mientras la línea de reloj está en ALTO, será interpretado como una condición de START o STOP.

De acuerdo a esto se han definido las siguientes condiciones (figura 11.1):

#### 1. Bus desocupado (A)

Las líneas de reloj y datos permanecen en ALTO.

#### 2. Inicio de transferencia de datos. Condición de START (B)

Una transición ALTO-BAJO en la línea SDA mientras la línea de reloj (SCL) está en ALTO determina la condición de START. Todos los comandos deben estar precedidos por una condición de START.

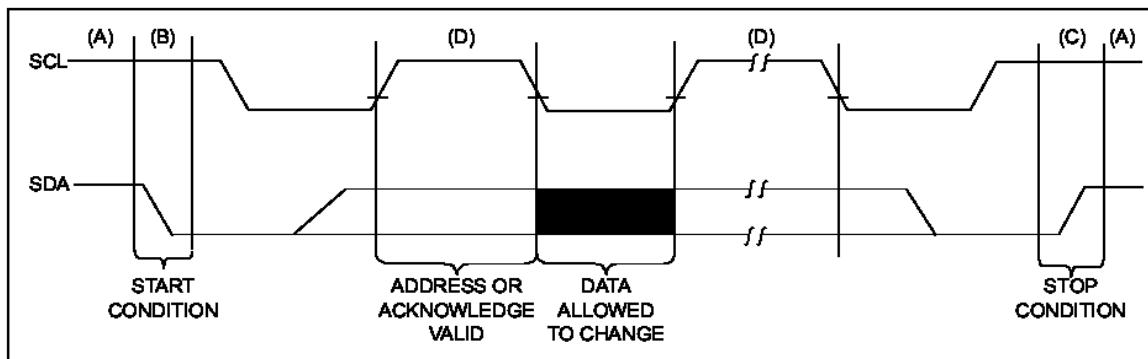
#### 3. Fin de transferencia de datos. Condición de STOP (C)

Una transición BAJO-ALTO en la línea SDA mientras la línea de reloj (SCL) está en ALTO determina la condición de STOP. Todas las operaciones deben finalizar con una condición de STOP.

#### 4. Dato válido (D)

El estado de la línea de datos representa datos válidos cuando, después de una condición de START, la línea de datos permanece estable durante el tiempo de nivel ALTO de la señal de reloj. Los datos en la línea tienen que cambiarse durante el tiempo de nivel BAJO de la señal de reloj. Hay un bit de datos por cada pulso del reloj.

Cada transferencia de datos comienza con una condición de START y finaliza con una condición de STOP. El número de bytes de datos transferido está determinado por el dispositivo maestro.

Figura 11.1 Secuencia de transferencia de datos en el bus I<sup>2</sup>C

### 5. Reconocimiento ACK

Cada dispositivo receptor, cuando es direccionado, está obligado a generar una señal de reconocimiento ACK después de la recepción de cada byte. El dispositivo maestro tiene que generar un pulso de reloj adicional que está asociado con este bit de reconocimiento (figura 11.2). El dispositivo que genera el reconocimiento tiene que colocar en BAJO la línea SDA durante el pulso de reloj correspondiente de tal manera que la línea SDA tenga un nivel BAJO estable durante el tiempo de nivel ALTO del ciclo de reloj. Durante las lecturas, el maestro tiene que informar al esclavo la finalización de los datos recibidos al NO generar un bit ACK en el último byte. En este caso, el esclavo dejará la línea de datos en ALTO para habilitar al maestro para que genere la condición de STOP.

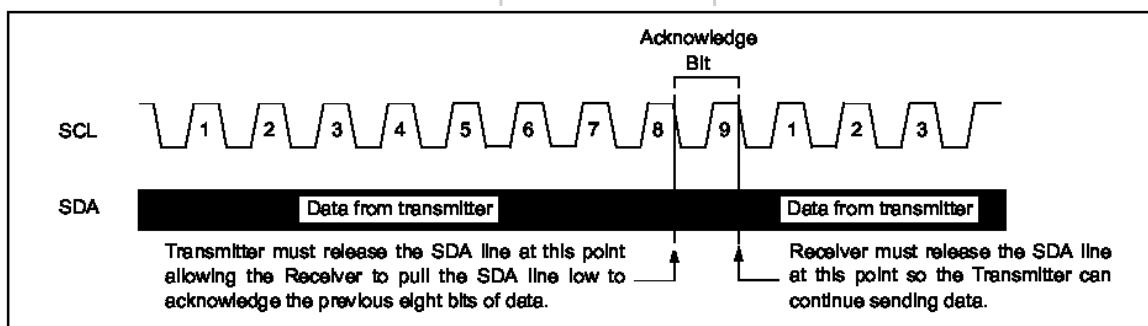
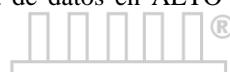


Figura 11.2 Temporización de reconocimiento ACK

El transmisor tiene que liberar la línea SDA al final del dato transmitido para permitir que el receptor lleve la línea a nivel BAJO como reconocimiento de los ocho bits de datos previos. El receptor tiene que liberar la línea al final del pulso de reloj para que el transmisor continúe enviando datos.

## 11.2 FUNCIONES DE mikroC™ PARA EL BUS I<sup>2</sup>C

La biblioteca *Software\_I2C* contiene las funciones necesarias para establecer la comunicación con dispositivos I<sup>2</sup>C. Estas funciones pueden emplearse con cualquier PIC (aunque no disponga de módulos I<sup>2</sup>C). Las funciones incorporadas permiten que el usuario emplee un PIC como maestro (el modo multi-maestro no está disponible). La tabla 11.2 muestra las funciones que pueden ser utilizadas para la comunicación I<sup>2</sup>C.

◆ Las interrupciones tienen que estar deshabilitadas cuando se emplea la biblioteca *Software\_I2C*. Los pines empleados para la comunicación deben estar conectados a los resistores de *pull-up*.

Función	Descripción
Soft_I2C_Init( )	Configura el módulo I <sup>2</sup> C por software.
Soft_I2C_Start( )	Determina si el bus I <sup>2</sup> C está libre y envía una señal de START.
Soft_I2C_Read(ack)	Lee un byte del esclavo y responde con una señal <i>NO ACKNOWLEDGE</i> (si ack= =0) o <i>ACKNOWLEDGE</i> ( si ack≠0).
Soft_I2C_Write(dato)	Envía un byte (dato) por el bus I <sup>2</sup> C.
Soft_I2C_Stop( )	Envía una señal de STOP.
Soft_I2C_Break( )	Todas las funciones de la biblioteca <i>Software_I2C</i> pueden bloquear el flujo del programa. Al ejecutar esta función desde la función de interrupción (interrupt) se producirá el desbloqueo del programa (este mecanismo es similar al WDT). Aún así, las interrupciones tienen que estar deshabilitadas.

Tabla 11.2 Funciones para el bus I<sup>2</sup>C

### 11.3 MEMORIA EEPROM EN BUS I<sup>2</sup>C

En algunos proyectos es necesario almacenar gran cantidad de datos en memoria EEPROM, sin embargo, la memoria disponible para los microcontroladores es muy limitada. Este problema se puede solucionar fácilmente con las memorias EEPROM serie, manteniendo la simplicidad del hardware, debido a que un solo chip de 8 pines puede almacenar gran cantidad de información. Se ha seleccionado la EEPROM serie 24LC256 (figura 11.3 y tabla 11.3) de Microchip, que es muy popular, con una capacidad de memoria de 32kbyte.

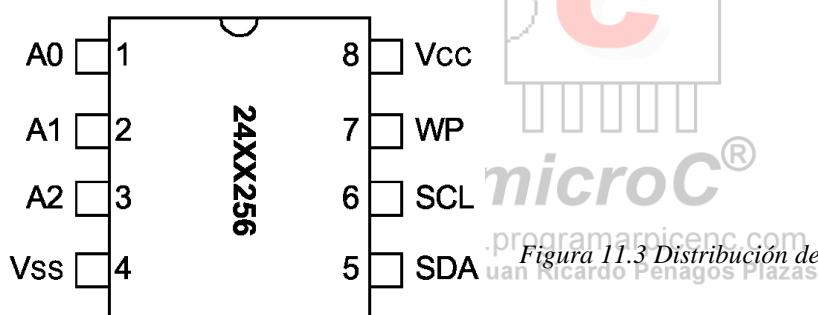


Figura 11.3 Distribución de terminales de la memoria 24LC256

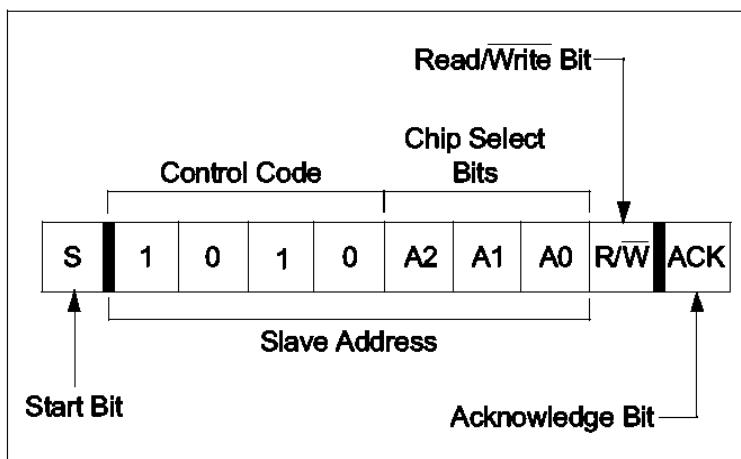
Pin	Descripción
A0, A1 y A2	Líneas de selección.
V <sub>SS</sub>	Referencia (negativo).
SDA	Línea de datos I <sup>2</sup> C.
SCL	Línea de reloj I <sup>2</sup> C.
WP	Protección contra escritura.
V <sub>CC</sub>	Fuente de alimentación (2,5 a 5,5V).

Tabla 11.3 Funciones de los pines de la memoria 24LC256

◆ La EEPROM para bus I<sup>2</sup>C debe considerarse como un ejemplo fundamental, y puede ser tomado como referencia para la comunicación del PIC con otros dispositivos que acepten el protocolo I<sup>2</sup>C: termómetro (DS1624), reloj calendario (DS1307), conversor AD/DA (PCF8591), etc.

#### 11.3.1 Direccionamiento como esclavo

El primer byte que se recibe del dispositivo maestro, después del START, es el Byte de Control (figura 11.4). El Byte de Control consiste de un código de control de 4 bits (**1010**<sub>2</sub> para la memoria 24LC256, **1001**<sub>2</sub> para el termómetro DS1624, **1101**<sub>2</sub> para el reloj DS1307, **1001**<sub>2</sub> para el conversor AD/DA PCF8591) para las operaciones de lectura y escritura. Los tres siguientes bits son los bits de selección (A<2:0>). Estos tres bits permiten el uso de hasta ocho memorias 24LC256 en el mismo bus y se emplean para seleccionar el dispositivo al que se tendrá acceso. Los bits A<2:0> del Byte de Control deben corresponder a los niveles lógicos en los pines de la memoria para que haya una respuesta. El último bit del Byte de Control define la operación que será realizada. Cuando es igual a 1 la operación es de lectura, cuando es igual a 0 la operación es de escritura. Los



siguientes dos bytes recibidos definen la dirección del primer byte de datos (figura 11.5). Debido a que únicamente se emplean los bits A<14:0>, el estado del bit A15 no importa. Los bits de dirección más significativos se transfieren primero y a continuación los bits menos significativos.

Figura 11.4 Formato del Byte de Control

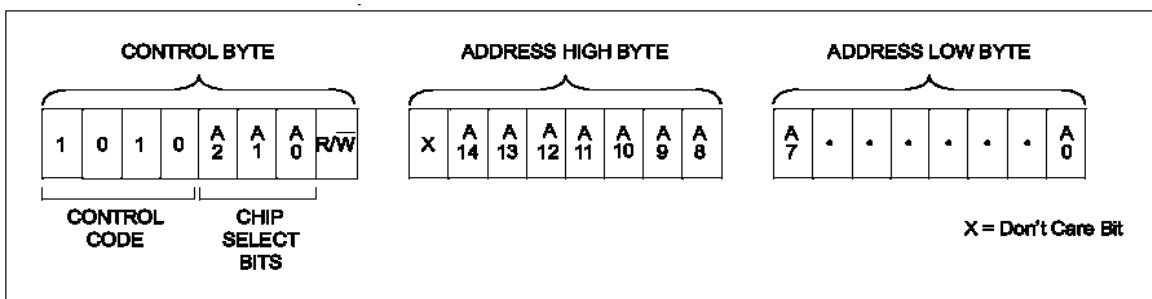


Figura 11.5 Secuencia de los bytes de Control y Dirección

Luego de la condición de START, la memoria monitorea la línea SDA para detectar el identificador de dispositivo que ha sido transmitido. Una vez que ha recibido el código 1010<sub>2</sub> y los bits de selección apropiados, el dispositivo esclavo envía una señal ACK (acknowledge o reconocimiento) a la línea SDA. Dependiendo del estado del bit R/#W, la memoria seleccionará una operación de lectura o escritura.

### 11.3.2 Operaciones de escritura

#### Escritura de un byte

Luego de recibir el Byte de Control configurado para la operación de escritura, el esclavo genera una señal ACK y a continuación el maestro transmite el primer byte de dirección, el esclavo genera otro ACK y el maestro transmite el segundo byte de dirección. Estos dos bytes son guardados en el puntero de direcciones de la memoria 24LC256. Despues de recibir otro ACK el maestro transmitirá el byte de datos que será almacenado en la localidad de memoria especificada previamente. La memoria genera otro ACK y el maestro genera una señal de STOP (figura 11.6). Aquí se inicia el ciclo interno de escritura, y, durante este tiempo (5ms como máximo), la memoria no generará señales ACK. El puntero de direcciones se incrementa automáticamente después de la operación de escritura de un byte de datos.

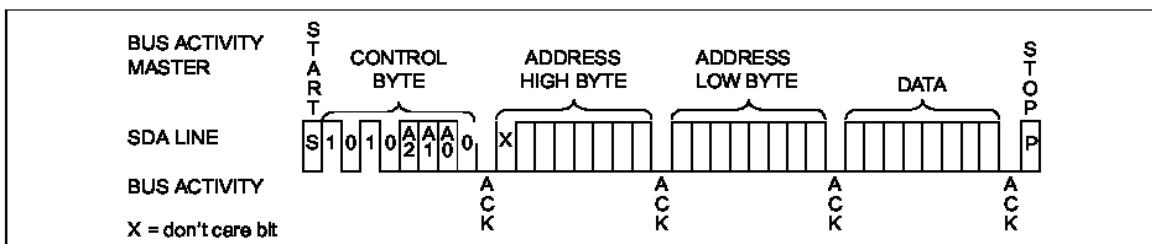


Figura 11.6 Escritura de un byte

#### Escritura de una página

El Byte de Control, la dirección y el primer byte de datos son transmitidos hacia la memoria 24LC256 en la misma forma en la que se escribe un byte. Pero en lugar de generar un STOP, el maestro transmite hasta un

máximo de 63 bytes adicionales, los cuales son almacenados temporalmente dentro del chip y serán escritos en la memoria después de que el maestro transmita la señal de STOP (figura 11.7). El puntero de direcciones se incrementa automáticamente después de recibir cada byte. Si el maestro desea transmitir más de 64 bytes se debe repetir la operación desde el principio. Una vez que la memoria recibe la condición de STOP, comienza el ciclo interno de escritura (5ms como máximo).

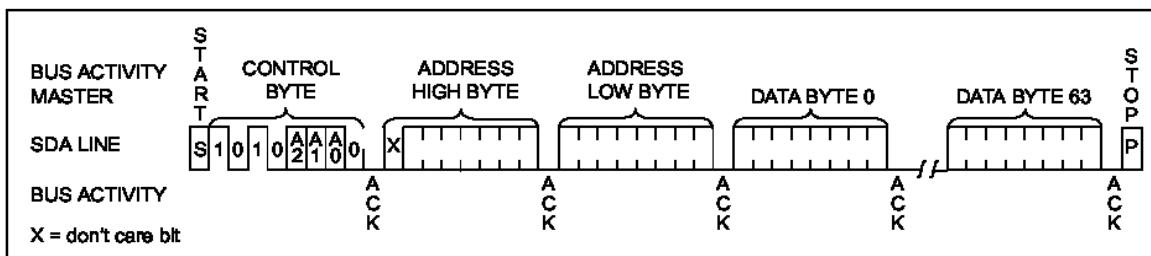


Figura 11.7 Escritura de una página

### 11.3.3 Operaciones de lectura

#### Lectura en la dirección actual

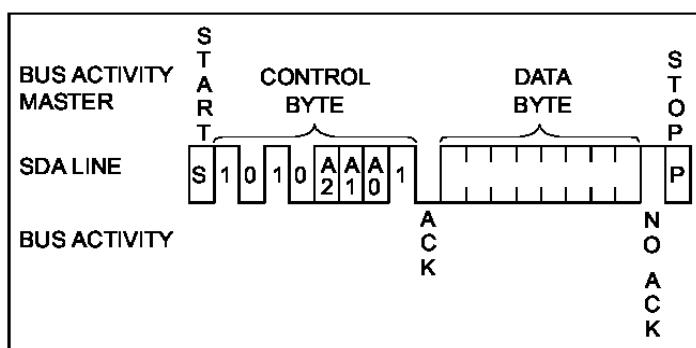


Figura 11.8 Lectura en la dirección actual

La memoria 24LC256 contiene un contador de direcciones que mantiene la última dirección de acceso, incrementada en 1. Por lo tanto, si la lectura anterior fue en la dirección n, la dirección de lectura actual sería n+1. Luego de recibir el Byte de Control, la memoria transmite una señal ACK y a continuación el byte de datos. El maestro transmitirá una señal NO ACK y generará una condición de STOP y con esto la memoria concluye el envío de datos (figura 11.8).

#### Lectura aleatoria

Este tipo de lectura permite que el maestro tenga acceso a cualquier dirección de la memoria. Primero debe definirse la dirección de acceso. Esto se hace enviando la dirección al chip de memoria como parte de una operación de escritura. Despues de enviar la dirección, el maestro genera un START luego del ACK proveniente de la memoria. Esto concluye la operación de escritura. Luego el maestro envía un nuevo byte de control pero en modo de lectura. La memoria enviará un ACK y a continuación el byte de datos. El maestro genera un NO ACK y una condición de STOP, lo que hace que la memoria concluya el envío de datos (figura 11.9). El contador de direcciones se incrementará automáticamente en 1.

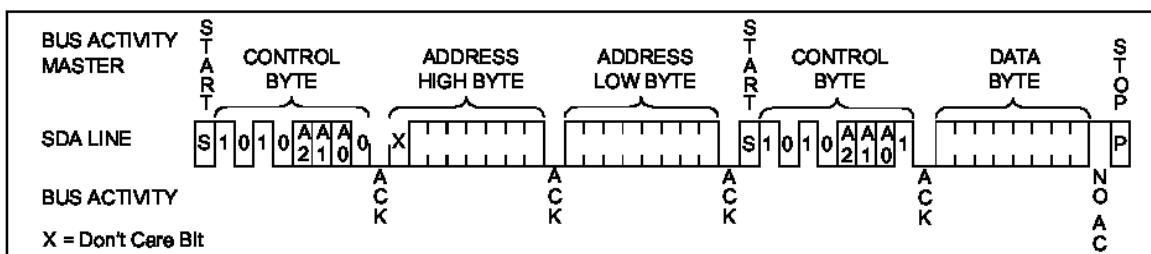


Figura 11.9 Lectura aleatoria

#### Lectura secuencial

La lectura secuencial se inicia de la misma forma que la lectura aleatoria, excepto que después de que la memoria envía el primer byte de datos, el maestro envía un ACK sin añadir la condición de STOP. Este ACK induce a la memoria a transmitir el byte ubicado inmediatamente después del byte anterior (figura 11.10). Al final del último byte, el maestro genera un NO ACK y una condición de STOP. Para realizar lecturas

secuenciales, la memoria 24LC256 contiene un contador de direcciones que se incrementa en 1 luego de cada operación. De esta forma es posible leer en serie toda la memoria en una sola operación.

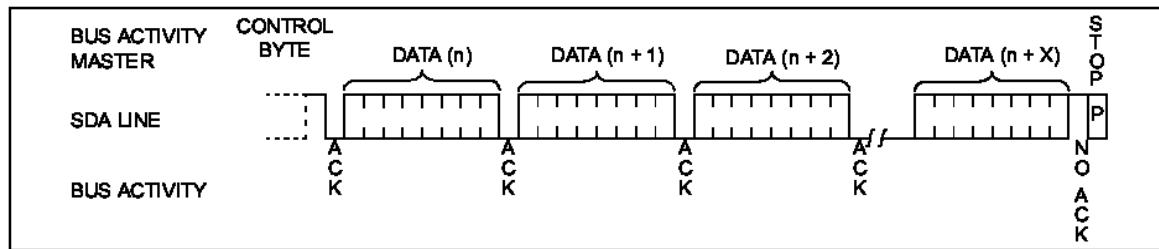
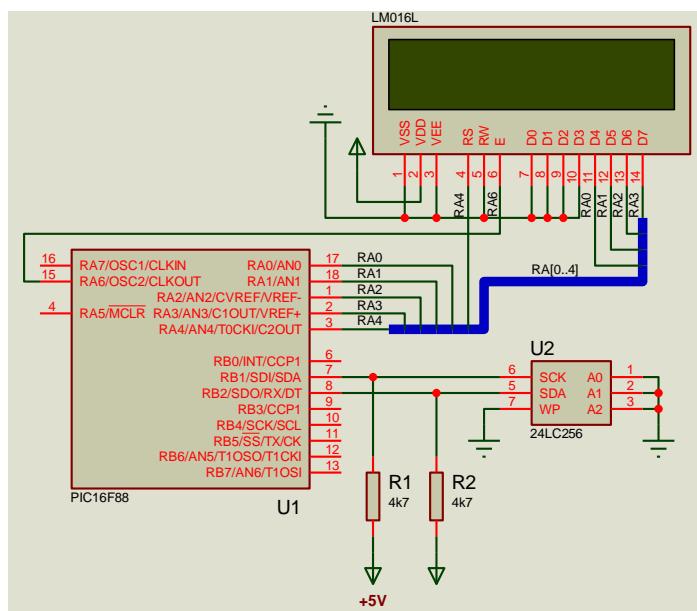


Figura 11.10 Lectura secuencial

## 11.4 EJEMPLOS DE PROGRAMACIÓN

Estos ejemplos corresponden al PIC16F88. El código fuente para los PICs 16F628A y 16F877A se encuentra en las carpetas correspondientes que acompañan a este libro.



En lenguaje C los *strings* o cadenas de caracteres terminan en el carácter NULO (0x00), lo cual se aprovecha en los siguientes ejemplos para determinar el final de la transferencia, tanto en operaciones de escritura como de lectura.

**I2C\_01.c:** Programa que escribe un mensaje de hasta 16 caracteres en la memoria 24LC256 a partir de la dirección 0x3B00. El mensaje se encuentra inicialmente en la memoria de programa del PIC. Luego el PIC lee el mensaje grabado anteriormente en la memoria y lo visualiza en el módulo LCD (figuras 11.11.1, 11.11.2 y 11.11.3). Software\_I2C✓. LCD✓. 4MHz✓.

Figura 11.11.1 Circuito del problema I2C\_01.c (PIC16F88)

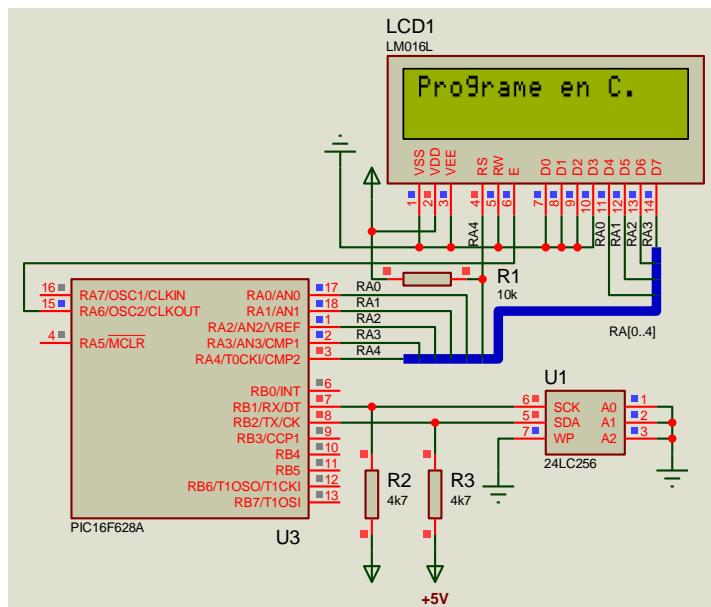


Figura 11.11.2 Circuito del problema I2C\_01.c (PIC16F628A)

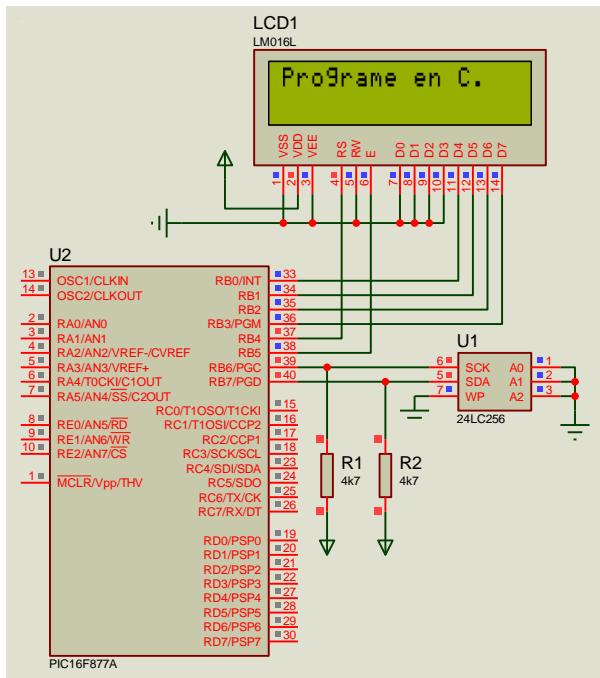


Figura 11.11.3 Circuito del problema I2C\_01.c  
(PIC16F877A)

```

 //I2C_01.c
//Variables de conexión I2C
sbit Soft_I2C_Scl at RB1_bit;
sbit Soft_I2C_Sda at RB2_bit;
sbit Soft_I2C_Scl_Direction at TRISB1_bit;
sbit Soft_I2C_Sda_Direction at TRISB2_bit;

//Variables de conexión del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;

sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit; Juan Ricardo Penagos Plazas
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.

char dato, texto[]={ "Programe en C.", i=0;

void main(){
OSCCON=0x60; //Oscilador interno a 4MHz (TCI=1 us).
//while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
Soft_I2C_Init(); //Inicializa la comunicación I2C.
Lcd_Init(); //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR); //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

//Escritura de una página.
Soft_I2C_Start(); //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x3B); //MSByte de dirección.
Soft_I2C_Write(0x00); //LSByte de dirección.
while (texto[i] != 0x00){
    Soft_I2C_Write(texto[i]); //Datos.
    i++;
}
Soft_I2C_Write(0x00); //Dato (carácter NULO).
Soft_I2C_Stop(); //Envía una señal de STOP.
Delay_ms(5); //Ciclo interno de escritura.
}

```



```

//Lectura secuencial.
Soft_I2C_Start();           //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x3B);       //MSByte de dirección.
Soft_I2C_Write(0x00);       //LSByte de dirección.
Soft_I2C_Start();
Soft_I2C_Write(0b10100001); //Byte de Control. Operación de lectura.
do{
    dato=Soft_I2C_Read(1);   //Lee un dato y responde con ACK.
    Lcd_ChR_Cp(dato);      //Envía el carácter al LCD.
}
while (dato != 0x00);
dato=Soft_I2C_Read(0);      //Lee el último dato (NULO) y responde con NO ACK.
Soft_I2C_Stop();            //Envía una señal de STOP.
}

```

El ejemplo siguiente muestra como se pueden gestionar varios mensajes largos almacenados en la memoria para su presentación en el LCD. Por supuesto, para muchos mensajes lo más conveniente es definir funciones tanto para la escritura como para la lectura, de otra manera el código fuente sería excesivamente extenso.

**I2C\_02.c:** Programa que escribe dos mensajes largos (hasta 40 caracteres por mensaje) en la memoria 24LC256 a partir de las direcciones 0x0100 y 0x0200. Los mensajes se encuentran inicialmente en la memoria de programa del PIC. Luego el PIC lee los mensajes grabados anteriormente en la memoria y los visualiza en el módulo LCD (circuitos de las figuras 11.11.1, 11.11.2 y 11.11.3). *Software\_I2C*  *LCD*  *4MHz*



```

//I2C_02.c
//Variables de conexión I2C
sbit Soft_I2C_Scl at RB1_bit;
sbit Soft_I2C_Sda at RB2_bit;
sbit Soft_I2C_Scl_Direction at TRISB1_bit;
sbit Soft_I2C_Sda_Direction at TRISB2_bit;

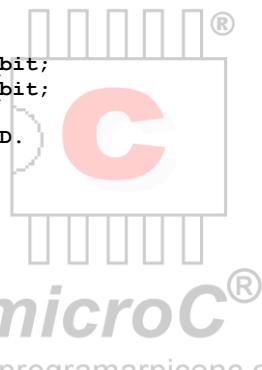
//Variables de conexión del módulo LCD.
sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;

sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit; Juan Ricardo Penagos Plazas
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.

char dato,contador,i;
char texto1[]="Estudie programacion en lenguaje C.";
char texto2[]="Programe sus microcontroladores en C.";
void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
//while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
ANSEL=0x00;              //Bits AN6:AN0 como E/S digital.
Soft_I2C_Init();          //Inicializa la comunicación I2C.
Lcd_Init();               //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);     //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.

//Escritura de una página (1er mensaje).
Soft_I2C_Start();         //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x01);      //MSByte de dirección.
Soft_I2C_Write(0x00);      //LSByte de dirección.
i=0;                      //Valor inicial del índice.
while (texto1[i] != 0x00){
    Soft_I2C_Write(texto1[i]); //Dato.
    i++;
}
Soft_I2C_Write(0x00);      //Dato (carácter NULO).
Soft_I2C_Stop();           //Envía una señal de STOP.
Delay_ms(5);               //Ciclo interno de escritura.
}

```



microC®

[www.programarpicenc.com](http://www.programarpicenc.com)

```

//Escritura de una página (2do mensaje).
Soft_I2C_Start();           //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x02);        //MSByte de dirección.
Soft_I2C_Write(0x00);        //LSByte de dirección.
i=0;                         //Valor inicial del índice.
while (texto2[i] != 0x00){
    Soft_I2C_Write(texto2[i]); //Dato.
    i++;
}
Soft_I2C_Write(0x00);         //Dato (carácter NULO).
Soft_I2C_Stop();             //Envía una señal de STOP.
Delay_ms(5);                //Ciclo interno de escritura.

//Lectura secuencial (1er mensaje).
Soft_I2C_Start();           //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x01);        //MSByte de dirección.
Soft_I2C_Write(0x00);        //LSByte de dirección.
Soft_I2C_Start();
Soft_I2C_Write(0b10100001); //Byte de Control. Operación de lectura.
contador=0;                  //Valor inicial del contador.
do{
    dato=Soft_I2C_Read(1);   //Lee un dato y responde con ACK.
    Lcd_ChR_CP(dato);       //Envía el carácter al LCD.
    contador++;
    if (contador>16)
        Lcd_Cmd(_LCD_SHIFT_RIGHT); //Desplaza la pantalla a la derecha.
    Delay_ms(150);
}
while (dato != 0x00);
dato=Soft_I2C_Read(0);        //Lee el último dato (NULO) y responde con NO ACK.
Soft_I2C_Stop();             //Envía una señal de STOP.

Delay_ms(500);               //Esperar antes de borrar la pantalla.
Lcd_Cmd(_LCD_CLEAR);         //Borra el display.

//Lectura secuencial (2do mensaje).
Soft_I2C_Start();           //Envía una señal de START.
Soft_I2C_Write(0b10100000); //Byte de Control. Operación de escritura.
Soft_I2C_Write(0x02);        //MSByte de dirección.
Soft_I2C_Write(0x00);        //LSByte de dirección.
Soft_I2C_Start();
Soft_I2C_Write(0b10100001); //Byte de Control. Operación de lectura.
contador=0;                  //Valor inicial del contador.
do{
    dato=Soft_I2C_Read(1);   //Lee un dato y responde con ACK.
    Lcd_ChR_CP(dato);       //Envía el carácter al LCD.
    contador++;
    if (contador>16)
        Lcd_Cmd(_LCD_SHIFT_RIGHT); //Desplaza la pantalla a la derecha.
    Delay_ms(150);
}
while (dato != 0x00);
dato=Soft_I2C_Read(0);        //Lee el último dato (NULO) y responde con NO ACK.
Soft_I2C_Stop();             //Envía una señal de STOP.

Delay_ms(500);               //Esperar antes de borrar la pantalla.
Lcd_Cmd(_LCD_CLEAR);         //Borra el display.
}

```

## CAPÍTULO XII: MOTORES DC Y PASO A PASO (PAP)

Las aplicaciones que tienen los motores en el área de la automatización son muy amplias, van desde los juguetes hasta la robótica industrial, pasando por la medicina, las aplicaciones militares, la investigación espacial y submarina, los electrodomésticos, las computadoras, los dispositivos de entretenimiento, los simuladores, las máquinas herramientas, los automóviles, etc. En este capítulo se estudia en detalle el control de sentido de giro, velocidad y posición angular de los motores DC convencionales y los motores paso a paso (PAP) o *stepper motor*.

### 12.1 CONTROLADOR L293B

El circuito integrado L293B (tabla 12.1 y figuras 12.1, 12.2, 12.3 y 12.4) se ha diseñado con el propósito de realizar el control de los motores de manera óptima y económica. Está conformado por cuatro amplificadores *push-pull* capaces de entregar una corriente de salida de 1A por canal. Cada canal está controlado por entradas compatibles con los niveles TTL y cada par de amplificadores (un puente completo) está equipado con una entrada de habilitación, que puede apagar los cuatro transistores de salida. Tiene una entrada de alimentación independiente para la lógica, de manera que se puede polarizar con bajos voltajes para reducir la disipación de potencia. Los cuatro pines centrales se emplean para conducir el calor generado hacia el circuito impreso. Sus características sobresalientes son las siguientes:

Símbolo	Significado	Valor máximo
V <sub>s</sub>	Fuente de alimentación (motores)	36 V
V <sub>ss</sub>	Fuente de alimentación de la lógica	36 V
V <sub>i</sub>	Voltaje de entrada	7 V
V <sub>inh</sub>	Voltaje de habilitación	7 V
I <sub>out</sub>	Corriente pico de salida	2 A
P <sub>tot</sub>	Disipación de potencia	5 W

Tabla 12.1 Valores máximos absolutos del driver L293B

- Corriente de salida de 1A por canal.
- Corriente pico de salida 2A por canal (no repetitiva).
- Pines de Habilitación.
- Alta inmunidad al ruido.
- Fuentes de alimentación separadas.
- Protección contra exceso de temperatura.

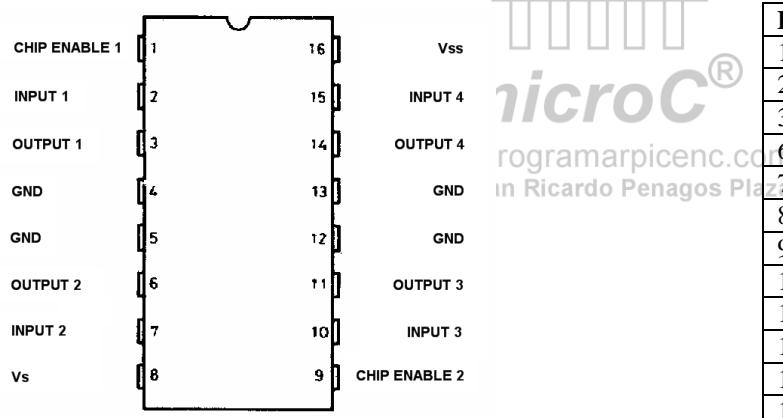


Figura 12.1 Distribución de terminales del driver L293B

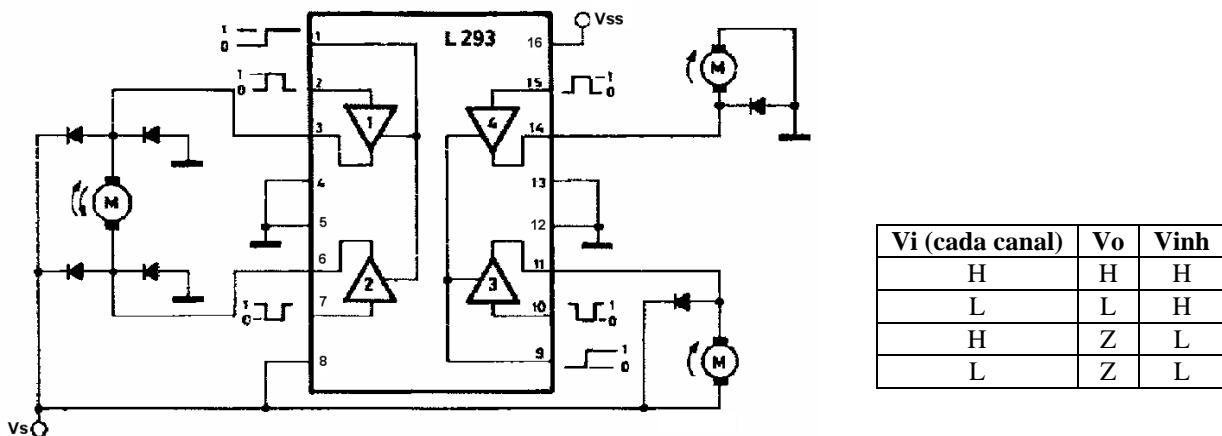


Figura 12.2 Diagrama de bloques del driver L293B y tabla de verdad ( $Z$ = Alta impedancia de salida). Se muestran diferentes tipos de conexión de motores DC.

Observe con cuidado la tabla de verdad de la figura 12.2 y note que si el voltaje de entrada de habilitación Vinh tiene un nivel ALTO el voltaje de salida Vo tendrá el mismo nivel (ALTO o BAJO), aunque NO el mismo valor, del nivel de entrada Vi. Algo que debe tenerse muy en cuenta es que los valores del voltaje de entrada Vi no son los mismos valores del voltaje de salida Vo, ya que Vi corresponde a valores TTL mientras que Vo es el voltaje de alimentación de los motores Vs.

Por otro lado, si Vinh tiene un valor BAJO, el pin de salida se pone en estado de alta impedancia (sin importar el valor del voltaje de entrada Vi).

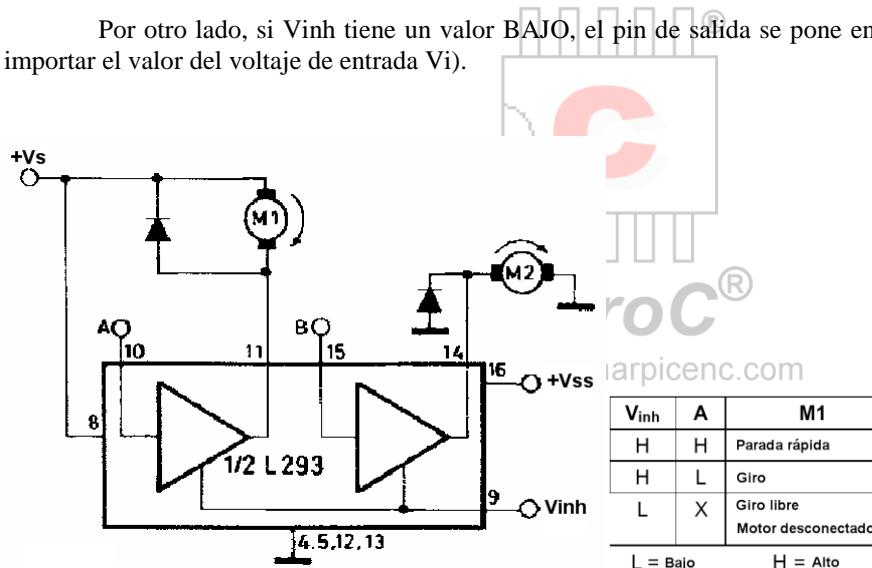


Figura 12.3 Control de motores DC (con conexión al positivo y al negativo de la fuente). Tabla de verdad.

La tabla de verdad de la figura 12.3 muestra la posibilidad de controlar dos motores en el mismo sentido de giro, con la diferencia de que M1 girará si la entrada A tiene un nivel BAJO, mientras que M2 girará si la entrada B tiene un nivel ALTO.

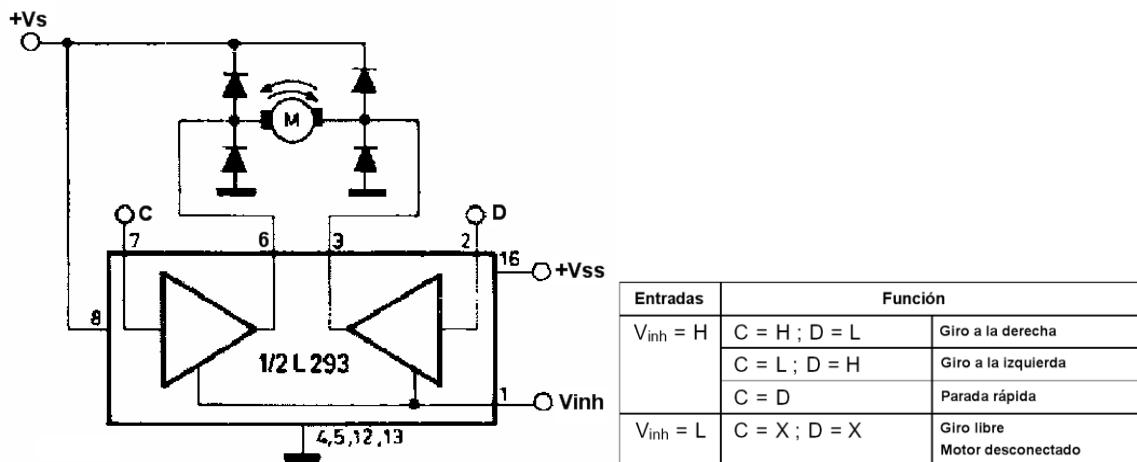
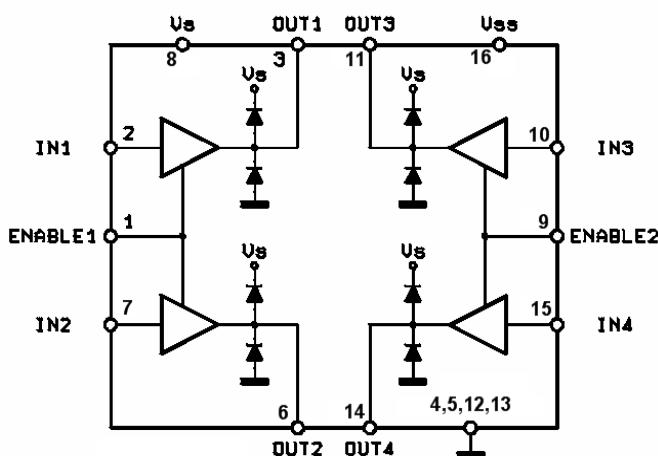
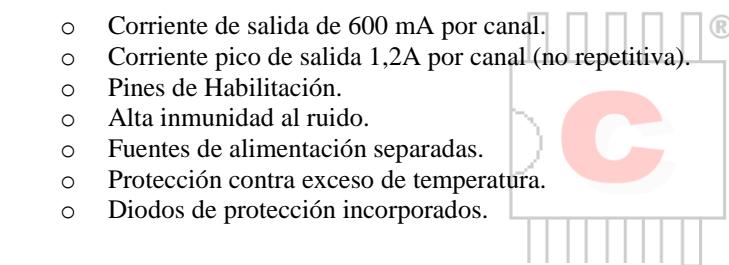


Figura 12.4 Control de giro en ambos sentidos de un motor DC

## 12.2 CONTROLADOR L293D

El *driver* L293D (figura 12.5) es similar al L293B, se diferencia fundamentalmente en su máxima corriente de salida y en la incorporación de los diodos de protección en cada uno de los cuatro amplificadores. Sus características principales son las siguientes:

- Corriente de salida de 600 mA por canal.
- Corriente pico de salida 1,2A por canal (no repetitiva).
- Pines de Habilitación.
- Alta inmunidad al ruido.
- Fuentes de alimentación separadas.
- Protección contra exceso de temperatura.
- Diodos de protección incorporados.



Está diseñado para recibir niveles TTL y alimentar cargas inductivas (relés, motores DC y PAP) y transistores de potencia de commutación. Este dispositivo se puede usar en aplicaciones de commutación hasta los 5 kHz. Está encapsulado en formato DIP16 y sus cuatro pines centrales se han conectado juntos y se emplean como disipadores de calor.

Figura 12.5 Diagrama de bloques del driver L293D

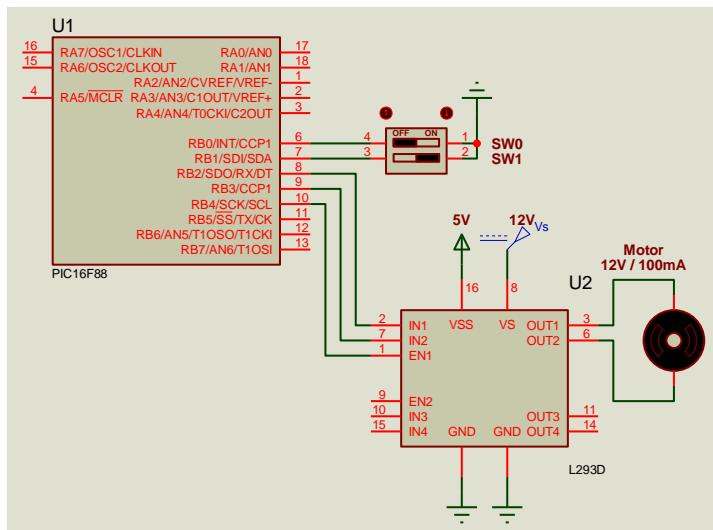
## 12.3 CONEXIÓN DEL DRIVER L293D AL PIC

El control de giro un motor DC por medio del *driver* L293D se detalla en el siguiente ejemplo. También puede emplearse el L293B tomando en cuenta que se deben añadir los diodos de protección (pueden ser del tipo 1N4007) como se indica en la figura 12.4.

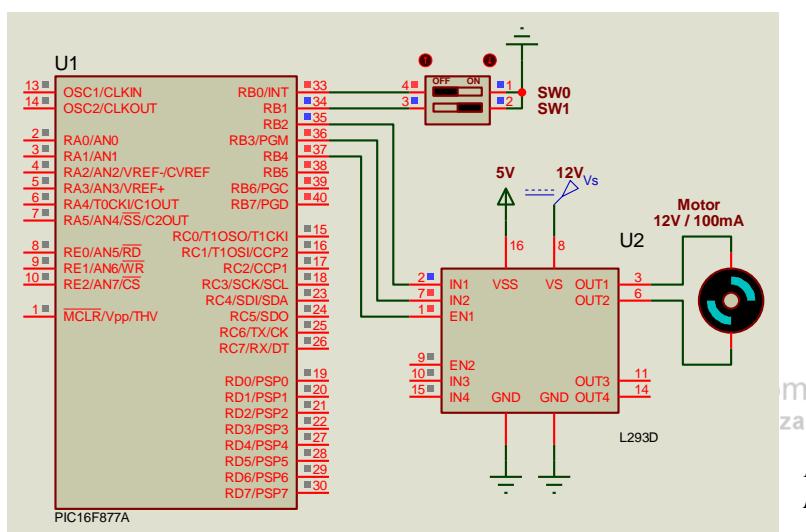
**MotorDC\_01.c:** Conexión típica de un motor DC al PIC a través del *driver* L293D (figuras 12.6.1 y 12.6.2). El giro del motor está determinado por el estado de los pines RB0 y RB1 de acuerdo a la tabla 12.2. El pin RB0 determina el encendido o apagado del motor, mientras que RB1 controla el sentido de giro. 4MHz .

<b>RB0</b>	<b>RB1</b>	<b>Motor</b>
0	X	Desconectado
1	0	Giro a la derecha
1	1	Giro a la izquierda

Tabla 12.2 Tabla de verdad del problema MotorDC\_01.c



*Figura 12.6.1 Circuito del problema MotorDC\_01.c (También PIC16F628A)*



*Figura 12.6.2 Circuito del problema MotorDC\_01.c (PJC16F877A)*

```
 //MotorDC_01.c
void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00;             //Inicialización.
NOT_RBPU_bit=0;         //Habilitar las pull-up.
TRISB=0b11100011;       //RB<4:2> como salidas.
while (1){
  if (RB0_bit==0) RB4_bit=0;           //Motor desconectado.
  if (RB0_bit==1){
    if (RB1_bit==0) PORTB=0b00011000; //Giro a la derecha.
    if (RB1_bit==1) PORTB=0b00010100; //Giro a la izquierda.
  }
}
}
```

Ahora se verá como realizar el control de giro y velocidad de un motor DC empleando PWM.

**(P1C16F88)** Debe modificarse el bit de configuración **CCP1\_Mux** para asignar la salida PWM al pin

RB3

**(PIC16F628A)** Es necesario conectar una resistencia de *pull-up* en el pin RA4.

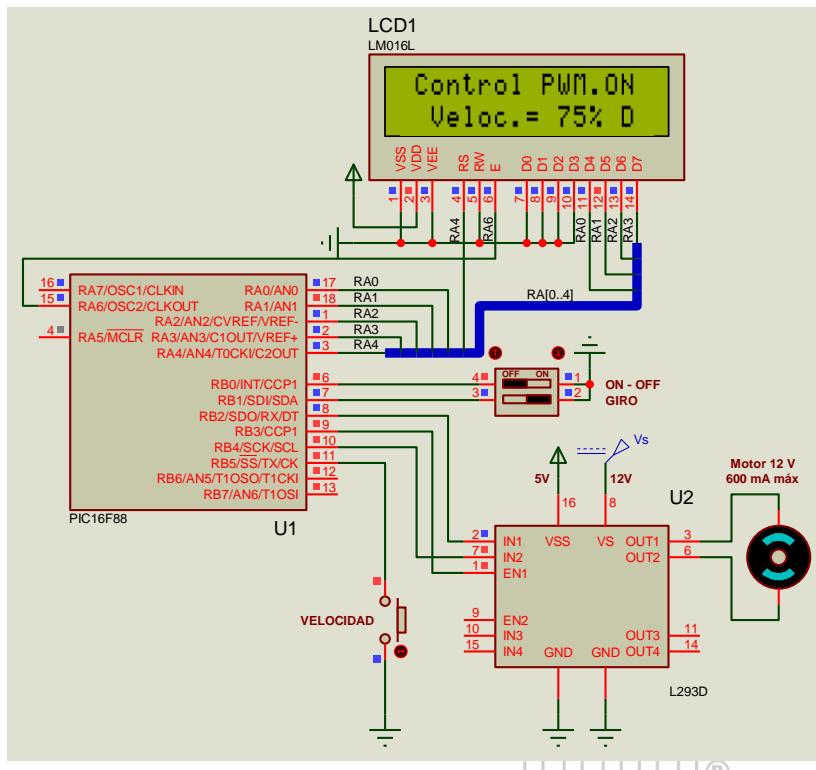


Figura 12.7.1 Circuito del problema MotorDC\_02.c (I=giro a la Izquierda; D=giro a la Derecha) (También PIC16F628A)

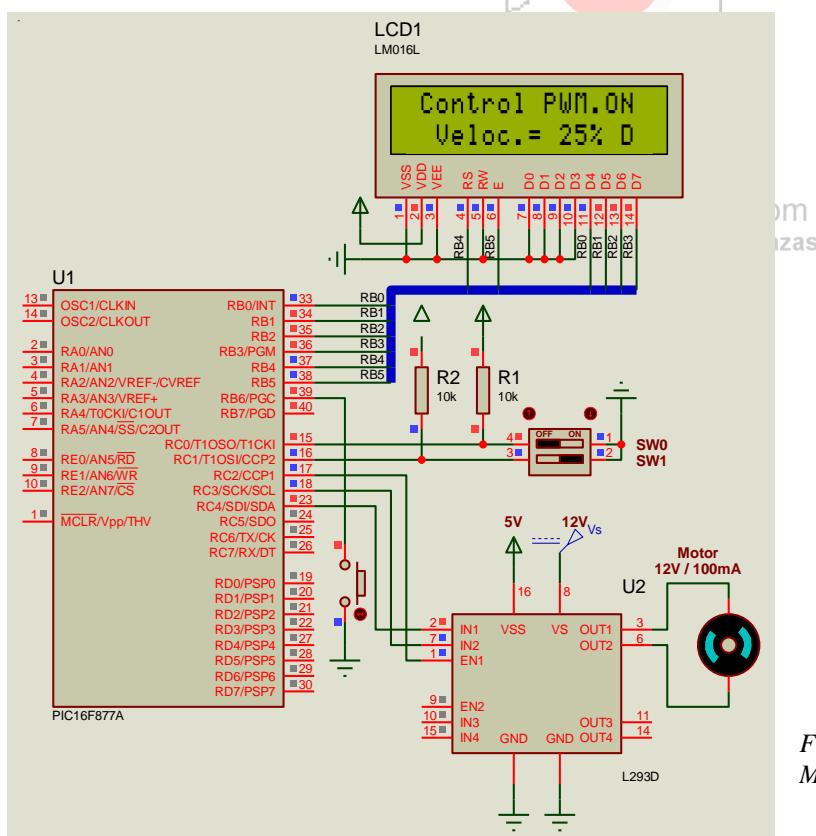


Figura 12.7.2 Circuito del problema MotorDC\_02.c (PIC16F877A)

/MotorDC\_02.c  
 //Variables de conexión del módulo LCD.

**MotorDC\_02.c:** Control de giro y velocidad de un motor DC de 12 V (figuras 12.7.1 y 12.7.2). El giro del motor está determinado por el estado de los pines RB0 y RB1 de acuerdo a la tabla 12.2. El pin RB0 determina el encendido o apagado del motor, mientras que RB1 controla el sentido de giro. La velocidad se puede seleccionar por medio de un pulsador conectado en RB5. Inicialmente el motor se encuentra detenido, al pulsar la primera vez la velocidad será del 25%, la segunda el 50%, la tercera el 75% y la cuarta el 100%. Si se vuelve a presionar, el motor se detiene. La velocidad actual y el sentido de giro se muestran en el LCD. *LCD*  *Button*  *PWM*  *4MHz* .

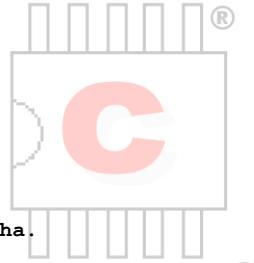
```

sbit LCD_RS at RA4_bit;
sbit LCD_EN at RA6_bit;
sbit LCD_D4 at RA0_bit;
sbit LCD_D5 at RA1_bit;
sbit LCD_D6 at RA2_bit;
sbit LCD_D7 at RA3_bit;
sbit LCD_RS_Direction at TRISA4_bit;
sbit LCD_EN_Direction at TRISA6_bit;
sbit LCD_D4_Direction at TRISA0_bit;
sbit LCD_D5_Direction at TRISA1_bit;
sbit LCD_D6_Direction at TRISA2_bit;
sbit LCD_D7_Direction at TRISA3_bit;
// Fin de declaración de variables de conexión.
char contador=0,estado=1;

void main(){
OSCCON=0x60;           //Oscilador interno a 4MHz (TCI=1 us).
//while (OSCCON.IOFS==0); //Esperar mientras el oscilador está inestable.
PORTB=0x00;             //Inicialización.
ANSEL=0x00;              //Bits AN6:AN0 como E/S digital.
NOT_RBPU_bit=0;          //Habilitar las pull-up.
Lcd_Init();               //Inicializa el LCD.
Lcd_Cmd(_LCD_CLEAR);    //Borra el display.
Lcd_Cmd(_LCD_CURSOR_OFF); //Apaga el cursor.
Lcd_Out(1,2,"Control PWM.");
PWM1_Init(5000);         //Frecuencia PWM.
PWM1_Start();
TRISB=0b11100011;        //RB<4:2> como salidas.

while (1){
if (RB0_bit==0){
Lcd_Out(1,14,"OFF");
Lcd_Out(2,3,"Veloc.= 0%");
PWM1_Set_Duty(0);
contador=0;
continue;
}
if (RB0_bit==1){
Lcd_Out(1,14,"ON ");
if (RB1_bit==0){
PORTB=0b00010000; //Giro a la derecha.
Lcd_Out(2,15,"D");
}
if (RB1_bit==1){
PORTB=0b00000100; //Giro a la izquierda.
Lcd_Out(2,15,"I");
}
}
if (Button(&PORTB,5,1,0)) estado=0; //Si se pulsa.
if (estado==0 && Button(&PORTB,5,1,1)){ //Si se pulsa y se libera.
contador++;
if (contador>4) contador=0;
estado=1;
}
switch (contador){
case 0:
Lcd_Out(2,3,"Veloc.= 0%");
PWM1_Set_Duty(0);
break;
case 1:
Lcd_Out(2,3,"Veloc.= 25%");
PWM1_Set_Duty(64);
break;
case 2:
Lcd_Out(2,3,"Veloc.= 50%");
PWM1_Set_Duty(127);
break;
case 3:
Lcd_Out(2,3,"Veloc.= 75%");
PWM1_Set_Duty(191);
break;
case 4:
Lcd_Out(2,3,"Veloc.=100%");
PWM1_Set_Duty(255);
}
}
}
}

```



**microC®**

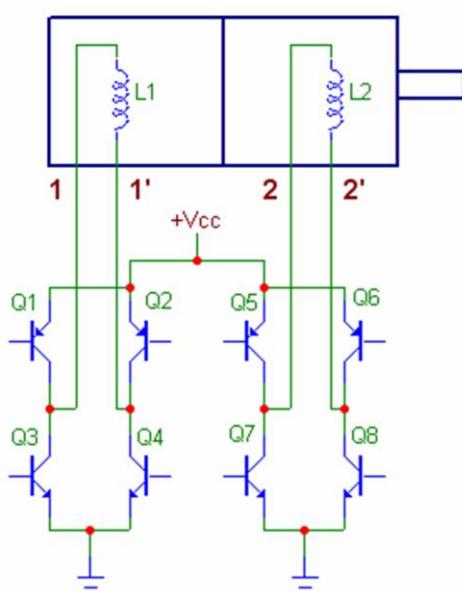
[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

## 12.4 MOTORES PASO A PASO-PAP (STEPPER / STEPPING MOTOR)

Los motores PAP son muy utilizados en dispositivos tales como impresoras, brazos de robots, lectores de discos compactos en ordenadores, aparatos electrodomésticos y, en general, en aquellas situaciones que requieren un posicionamiento preciso del eje de un motor. En conjunto con un microcontrolador los motores PAP pueden ser utilizados en una amplia gama de aplicaciones. Un motor PAP gira a intervalos correspondientes a una fracción de la circunferencia en función de una secuencia de pulsos aplicados a sus devanados. El eje del motor se mueve un determinado número de grados (**paso**) por cada pulso de entrada, pudiendo rotar en los dos sentidos. Comercialmente se encuentran motores PAP con pasos que van desde  $0,72^\circ$  (500 pasos) hasta  $22,5^\circ$  (16 pasos), siendo el más popular el motor PAP de  $7,5^\circ$  (48 pasos). La velocidad del motor queda determinada por la frecuencia de los pulsos de entrada, mientras que el sentido de giro depende del sentido en que se aplique la secuencia de pulsos.

### 12.4.1 Motores PAP bipolares



Estos motores se caracterizan por un funcionamiento particular, ya que requieren un método que permita la alimentación del estator con corrientes de dos sentidos o polaridades (de ahí el nombre de **bipolar**). El estator está conformado por dos bobinas (4 terminales) que al ser alimentadas con la secuencia apropiada permiten el control tanto del sentido de giro como del desplazamiento angular (figura 12.8 y tabla 12.3). Tiene dos modos de funcionamiento: *Full Step* (avanza un paso por cada pulso de entrada) y *Half Step* (avanza medio paso por cada pulso de entrada).

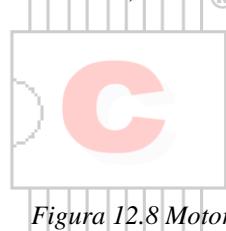


Figura 12.8 Motor PAP bipolar con circuito externo de control

**.nicroC®**

[www.programarpicenc.com](http://www.programarpicenc.com)

Paso	Q2Q3	Q1Q4	Q6Q7	Q5Q8		1	1'	2	2'	a) Full Step
1	ON	OFF	OFF	ON		-	+	+	-	
2	ON	OFF	ON	OFF		-	+	-	+	
3	OFF	ON	ON	OFF		+	-	-	+	
4	OFF	ON	OFF	ON		+	-	+	-	
		F U L L	S T E P							

Paso	Q2Q3	Q1Q4	Q6Q7	Q5Q8		1	1'	2	2'	b) Half Step
1	ON	OFF	OFF	ON		-	+	+	-	
2	ON	OFF	OFF	OFF		-	+	+	+	
3	ON	OFF	ON	OFF		-	+	-	+	
4	OFF	OFF	ON	OFF		+	+	-	+	
5	OFF	ON	ON	OFF		+	-	-	+	
6	OFF	ON	OFF	OFF		+	-	+	+	
7	OFF	ON	OFF	ON		+	-	+	-	
8	OFF	OFF	OFF	ON		+	+	+	-	
		H A L F	S T E P							

Tabla 12.3 Secuencia de control para un motor PAP bipolar: a) En modo Full Step, b) En modo Half Step

### 12.4.2 Motores PAP unipolares

En estos motores la corriente por las bobinas del estator circula en un solo sentido o polaridad (de ahí el nombre de **unipolar**). El estator está conformado por 4 bobinas que se conectan internamente formando dos grupos (por lo tanto se dispone de 6 terminales) que al ser alimentadas con la secuencia apropiada permiten el

control, tanto del sentido de giro como del desplazamiento angular (figura 12.9 y tabla 12.4). Tiene dos modos de funcionamiento: *Full Step* y *Half Step*.

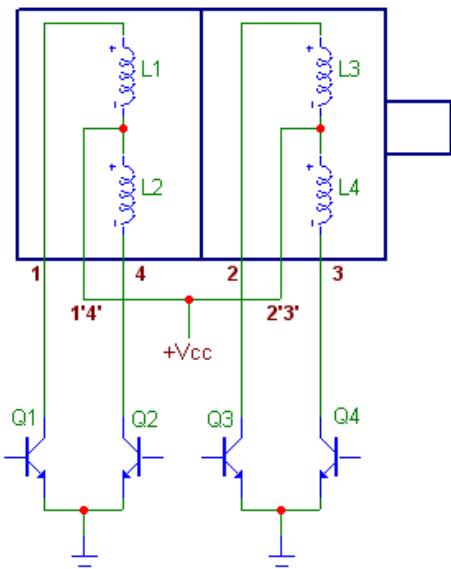


Figura 12.9 Motor PAP unipolar con circuito externo de control

Paso	Q1	Q2	Q3	Q4		1	2	3	4
1	ON	OFF	OFF	ON		-	D	D	-
2	ON	OFF	ON	OFF		-	D	-	D
3	OFF	ON	ON	OFF		D	-	-	D
4	OFF	ON	OFF	ON		D	-	D	-
F U L L   S T E P									

a)

Paso	Q1	Q2	Q3	Q4		1	2	3	4
1	ON	OFF	OFF	ON		-	D	D	-
2	ON	OFF	OFF	OFF		-	D	D	D
3	ON	OFF	ON	OFF		-	D	-	D
4	OFF	OFF	ON	OFF		D	D	-	D
5	OFF	ON	ON	OFF		D	-	-	D
6	OFF	ON	OFF	OFF		D	-	D	D
7	OFF	ON	OFF	ON		D	-	D	-
8	OFF	OFF	OFF	ON		D	D	D	-
H A L F   S T E P									

b)

Tabla 12.4 Secuencia de control para un motor PAP unipolar: a) En modo Full Step, b) En modo Half Step (D=desconectado)

### 12.4.3 Configuración de las bobinas

Comercialmente se dispone de motores PAP que presentan varias configuraciones en sus bobinas, lo que permite que el usuario pueda realizar las conexiones más adecuadas de acuerdo a sus necesidades (figura 12.10).

Debe tenerse en cuenta que los motores unipolares de seis u ocho hilos pueden hacerse funcionar como motores bipolares si no se emplean los terminales centrales, mientras que los de cinco terminales no podrán trabajar como bipolares ya que internamente se encuentran conectados los dos cables centrales.

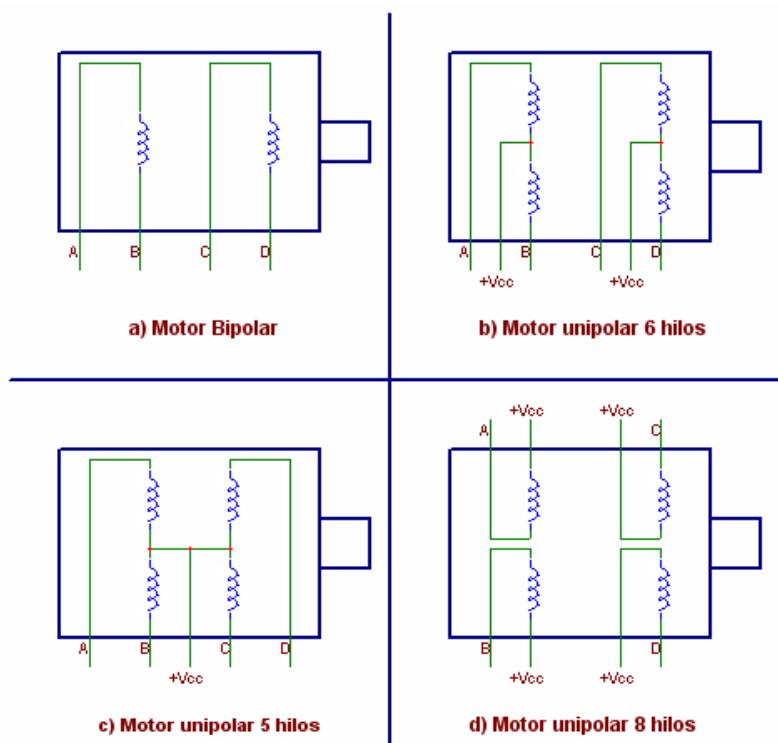


Figura 12.10 Disposición de las bobinas en motores PAP

#### 12.4.4 Conexión de un motor bipolar al PIC

El control de un motor bipolar por medio del *driver* L293D se detalla en el siguiente ejemplo. Nótese que el desplazamiento angular dependerá del paso del motor empleado, mientras que el tiempo total para la ejecución de la secuencia propuesta será una función de la frecuencia de los pulsos (para el ejemplo, el motor tardará 62 pasos x 20 ms/paso=1240 ms). Otra cuestión de suma importancia a la hora de realizar secuencias de movimientos es la de hacer que la secuencia de control sea continua (tabla 12.5), de tal manera que no se produzcan irregularidades en el giro del motor; con esa finalidad se han añadido los pulsos necesarios al principio y al final de cada segmento de la secuencia. Para aclarar la idea, al final de los 20 pasos  $\rightarrow$  el último pulso es  $RB<3:0>=1010_2$  (+ - - -); por lo tanto, el primero de los 20 pasos  $\leftarrow$  tiene que ser  $RB<3:0>=1001_2$  (+ - - +), y así sucesivamente.

	1	1'	2	2'
	-	+	+	-
	-	+	-	+
	+	-	-	+
	+	-	+	-
	RB3	RB2	RB1	RB0

FULL STEP

[www.programarpicenc.com](http://www.programarpicenc.com)

**MotorPAP\_01.c:** Conexión típica de un motor PAP bipolar en modo *Full Step* al PIC a través del *driver* L293D (figuras 12.11.1, 12.11.2 y tabla 12.5). La frecuencia de los pulsos es de 50 Hz (duración de los pulsos igual a 20 ms). Realiza la siguiente secuencia de movimientos: 20 pasos  $\rightarrow$  (CW) y 20 pasos  $\leftarrow$  (CCW) cada vez que se presiona un pulsador conectado en el pin RA0. *Button* 4MHz .

Tabla 12.5 Secuencia de control del problema MotorPAP\_01.c

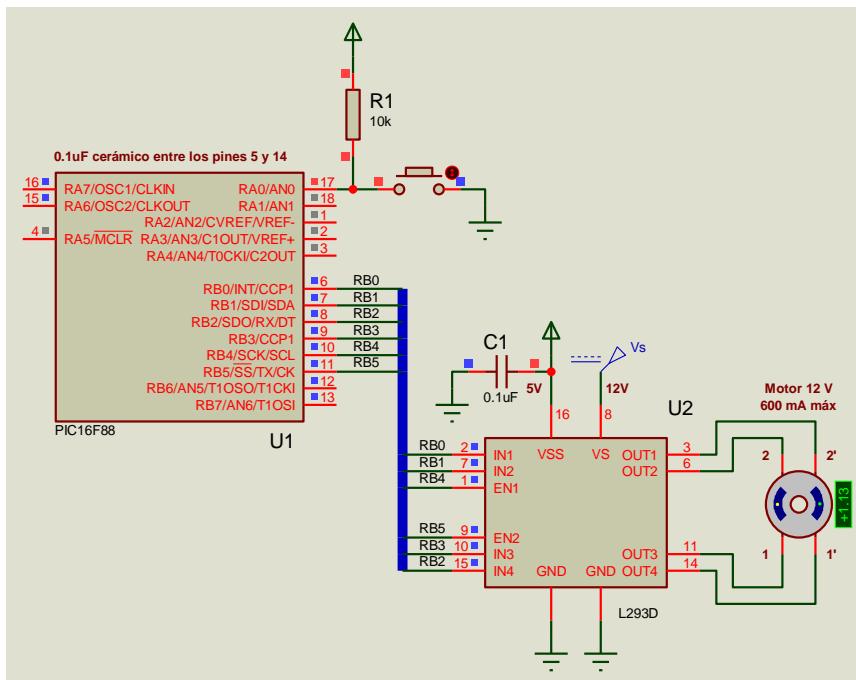


Figura 12.11.1 Circuito del problema MotorPAP\_01.c  
(También PIC16F628A)

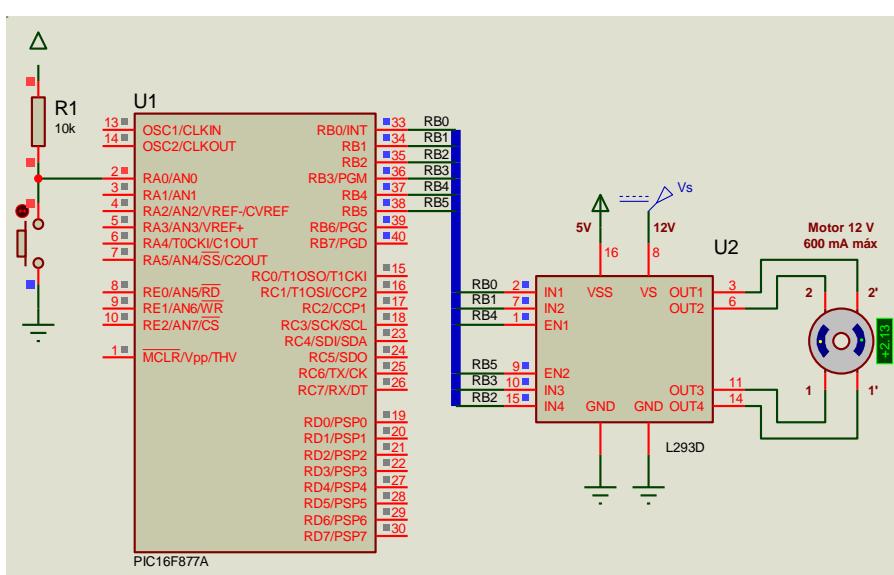


Figura 12.11.2 Circuito del problema  
MotorPAP\_01.c  
(PIC16F877A)

```


//MotorPAP_01.c
//Realiza una secuencia de 20 pasos CW y 20 pasos CCW cada vez que se
//activa un pulsador conectado en el pin RA0 (pin 17).
void pasosCW(); //Cuatro pasos CW.
void pasosCCW(); //Cuatro pasos CCW.

char estado=1;
void main(){
OSCCON=0x60; //Oscilador interno a 4MHz (TCI=1 us).
PORTB=0x00; //Inicialización.
ANSEL=0x00; //Bits AN6:AN0 como E/S digital.
TRISB=0b00000000; //RB<7:0> como salidas.
while(1{
    if (Button(&PORTA,0,1,0)) estado=0; //Si se pulsa.
    if (estado==0 && Button(&PORTA,0,1,1))//Si se pulsa y se suelta.
    {
        //20 pasos CW
        pasosCW();
        pasosCW();
        pasosCW();
        pasosCW();
}

```

```

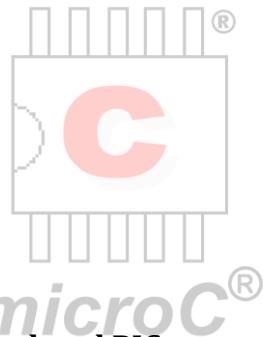
pasosCW();

//20 pasos CCW
PORTB=0b00111001;           //Secuencia continua.
Delay_ms(20);
PORTB=0b00110101;           //Secuencia continua.
Delay_ms(20);
PORTB=0b00110110;           //Secuencia continua.
Delay_ms(20);
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
PORTB=0b00111010;           //Secuencia continua.
Delay_ms(20);
PORTB=0b00000000;           //Apagar el motor.
estado=1;
}
}
}

//Definición de funciones.
void pasosCW(){
  PORTB=0b00110110;
  Delay_ms(20);
  PORTB=0b00110101;
  Delay_ms(20);
  PORTB=0b00111001;
  Delay_ms(20);
  PORTB=0b00111010;
  Delay_ms(20);
}

void pasosCCW(){
  PORTB=0b00111010;
  Delay_ms(20);
  PORTB=0b00111001;
  Delay_ms(20);
  PORTB=0b00110101;
  Delay_ms(20);
  PORTB=0b00110110;
  Delay_ms(20);
}
}

```



#### 12.4.5 Conexión de un motor unipolar al PIC

Se muestra un ejemplo de la conexión de un motor PAP unipolar al PIC. Es similar a la conexión del motor bipolar, tomando en cuenta que ahora el voltaje de alimentación del motor no se conecta al *driver* L293D, si no más bien a las tomas centrales del propio motor.

**MotorPAP\_02.c:** Conexión típica de un motor PAP unipolar en modo *Half Step* al PIC a través del *driver* L293D (figuras 12.12.1, 12.12.2 y tabla 12.6). El encendido/apagado del motor está determinado por el estado del pin RB7. La frecuencia de los pulsos es de 10 Hz (duración de los pulsos igual a 100 ms). Realiza la siguiente secuencia de movimientos de forma repetitiva: 37 pasos ↗ (CW) y 65 pasos ↘ (CCW). 4MHz☒.

	1	2	3	4	No.
	-	D	D	-	1
	-	D	D	D	2
	-	D	-	D	3
	D	D	-	D	4
	D	-	-	D	5
	D	-	D	D	6
	D	-	D	-	7
	D	D	D	-	8
	RB3	RB2	RB1	RB0	
HALF STEP					

Tabla 12.6 Secuencia de control del problema MotorPAP\_02.c

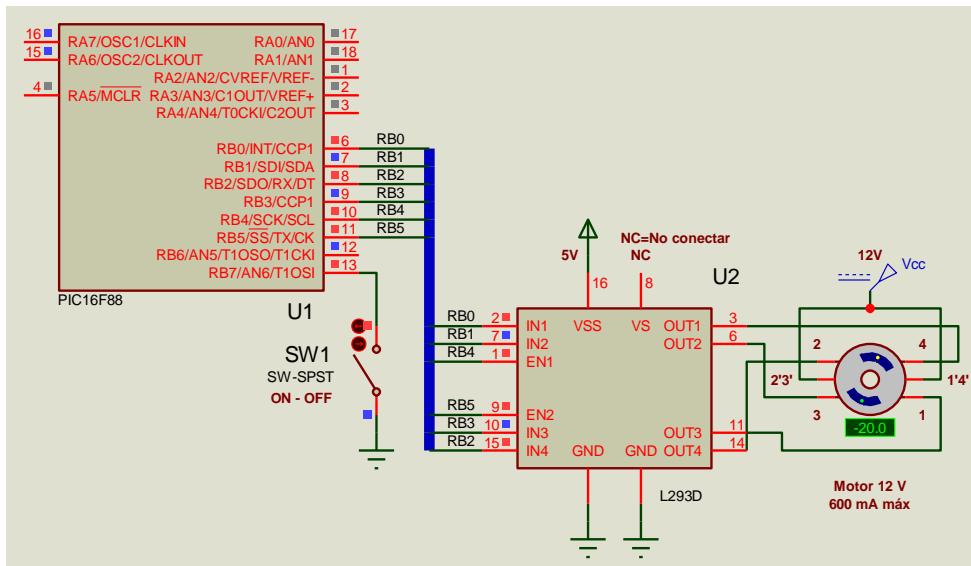


Figura 12.12.1  
Circuito del  
problema  
MotorPAP\_02.c  
(También  
PIC16F628A)

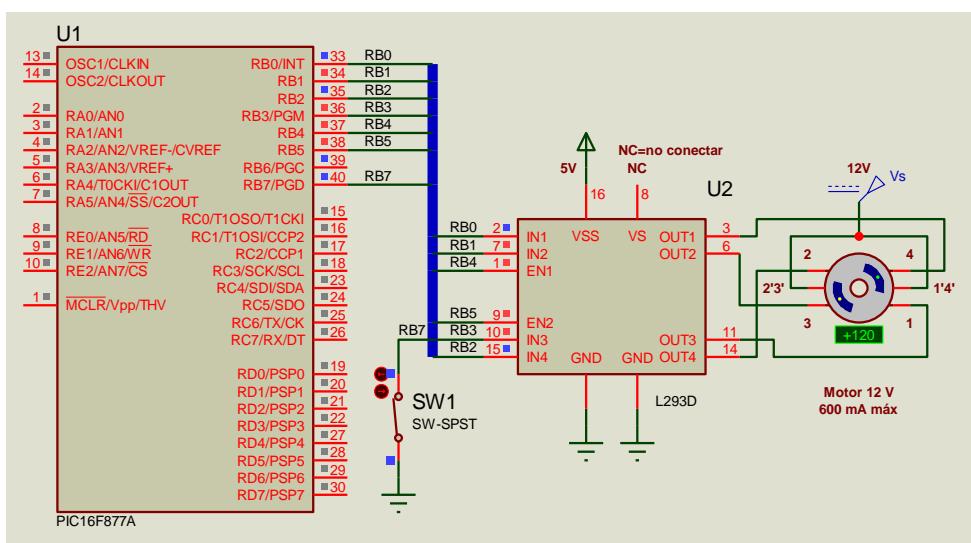


Figura 12.12.2  
Circuito del  
problema  
MotorPAP\_02.c  
(PIC16F877A)

```


//MotorPAP_02.c
void pasosCW();
void pasosCCW();

void main(){
OSCCON=0x60;
//while (OSCCON.IOFS==0);
PORTB=0x00;
NOT_RBPU_bit=0;
TRISB=0b10000000;

while(1{
if (RB7_bit==0){
RB4_bit=0;
RB5_bit=0;
}
if (RB7_bit==1){
//37 pasos CW
pasosCW();
pasosCW();
pasosCW();
pasosCW();
PORTB=0b00110110;
Delay_ms(100);
PORTB=0b00110111;
Delay_ms(100);
}
}


```

```

PORTB=0b00110101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111001;           //Secuencia continua.
Delay_ms(100);

//65 pasos CCW
PORTB=0b00111101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00110101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00110111;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00110110;           //Secuencia continua.
Delay_ms(100);
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
pasosCCW();
PORTB=0b00111110;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111010;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111011;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111001;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111101;           //Secuencia continua.
Delay_ms(100);
PORTB=0b00111111;           //Apagar el motor.
Delay_ms(5000);              //Esperar 5 segundos antes de repetir.

}

}

}

//Definición de funciones.
void pasosCW(){
PORTB=0b00110110;
Delay_ms(100);
PORTB=0b00110111;
Delay_ms(100);
PORTB=0b00110101;
Delay_ms(100);
PORTB=0b00111101;
Delay_ms(100);
PORTB=0b001111001;
Delay_ms(100);
PORTB=0b001111001;
Delay_ms(100);
PORTB=0b001111011;
Delay_ms(100);
PORTB=0b001111010;
Delay_ms(100);
PORTB=0b00111110;
Delay_ms(100);
}

void pasosCCW(){
PORTB=0b00111110;
Delay_ms(100);
PORTB=0b00111010;
Delay_ms(100);
PORTB=0b00111011;
Delay_ms(100);
PORTB=0b00111001;
Delay_ms(100);
PORTB=0b00111101;
Delay_ms(100);
PORTB=0b00110101;
Delay_ms(100);
PORTB=0b00110111;
Delay_ms(100);
PORTB=0b00110110;
Delay_ms(100);
}

```



**microC®**

www.programarpicenc.com  
©Ing. Juan Ricardo Penagos Plazas

## **CAPÍTULO XIII: LCD GRÁFICO (GRAPHIC LCD GLCD)**

### **13.1 TIPOS DE LCD**

Hay tres tipos de LCD (Pantalla o Display de Cristal Líquido):

- De segmentos (o alfanumérico).
- De matriz de puntos (o carácter).
- LCD gráfico.

#### **13.1.1 LCD de segmentos o alfanumérico**



Figura 13.1 LCD de segmentos

Esta pantalla puede mostrar números arábigos representados por 7 segmentos, o números arábigos y letras romanas representados por 14 segmentos. Los símbolos, tales como el signo menos (-) y el signo más (+), las unidades de medida y algunos iconos particulares también se pueden mostrar (figura 13.1).

El LCD de segmentos se emplea mucho en los instrumentos científicos. Se puede controlar fácilmente y tiene un costo relativamente bajo.

Se limita a mostrar números, letras romanas y símbolos fijos. Si se necesita mostrar algo más, se debe emplear una pantalla de matriz de puntos o una pantalla para gráficos.



#### **13.1.2 LCD de matriz de puntos (o LCD de caracteres)**



Figura 13.2 LCD de matriz de puntos

El LCD de matriz de puntos se emplea para mostrar una o varias líneas de caracteres. La pantalla más común muestra entre 1 y 4 líneas de 16 a 40 caracteres. Cada carácter se representa por una matriz de 5x7 puntos y un cursor (actualmente una matriz de 5x8 puntos ya incluye el cursor). Cada matriz individual se maneja independientemente y puede formar números, letras romanas, caracteres de otros idiomas, y una cantidad limitada de símbolos (figura 13.2).

Esta pantalla se emplea cuando la necesidad es mostrar más caracteres que los existentes en el idioma Inglés. Es relativamente fácil de controlar y tiene un costo menor que los modelos para gráficos. En esta categoría se encuentra el LCD estudiado en el capítulo III de este libro.

#### **13.1.3 LCD para gráficos (graphic LCD GLCD)**



Figura 13.3 LCD gráfico

El GLCD proporciona mucha flexibilidad. Está compuesto de píxeles dispuestos en filas y columnas. Cada pixel puede manejarse individualmente y permite mostrar texto, gráficos o una combinación de ambos (figura 13.3).

Se emplea en aquellos casos en los que es necesario tener un control total del área de la pantalla. Sin embargo, la flexibilidad implica una mayor dificultad en el diseño del circuito de control. Afortunadamente existen controladores especiales para este propósito (el circuito integrado T6963C de Toshiba es uno de los más utilizados actualmente).

## 13.2 TECNOLOGÍA LCD

Los LCDs constituyen una tecnología de presentación pasiva. Esto quiere decir que no emiten luz; en lugar de ello usan la luz ambiental. Por medio de esta luz se pueden mostrar imágenes con muy poco consumo de potencia. Esto ha hecho que los LCDs sean preferidos cuando los factores críticos en el diseño son el bajo consumo y el tamaño compacto. El cristal líquido (LC) es una sustancia orgánica que tiene un estado líquido y una estructura molecular cristalina. En este líquido, normalmente las moléculas (con forma de barra) se disponen paralelamente (figura 13.4), y puede emplearse un campo eléctrico para controlar dichas moléculas. La gran mayoría de LCDs en la actualidad usan un cristal líquido llamado *Twisted Nematic* (*TN*).

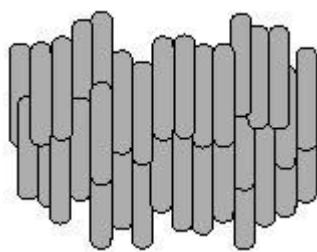


Figura 13.4 Disposición natural de las moléculas de cristal líquido

Un LCD está formado por dos sustratos que forman una “botella plana” que contiene la mezcla de cristal líquido. Las superficies internas de la botella están cubiertas con un polímero que alinea las moléculas de cristal líquido sobre las superficies. Para dispositivos TN, las dos superficies están tratadas de tal forma que la alineación de una cara con respecto a la otra produce un giro relativo de 90° en las moléculas (figura 13.5).

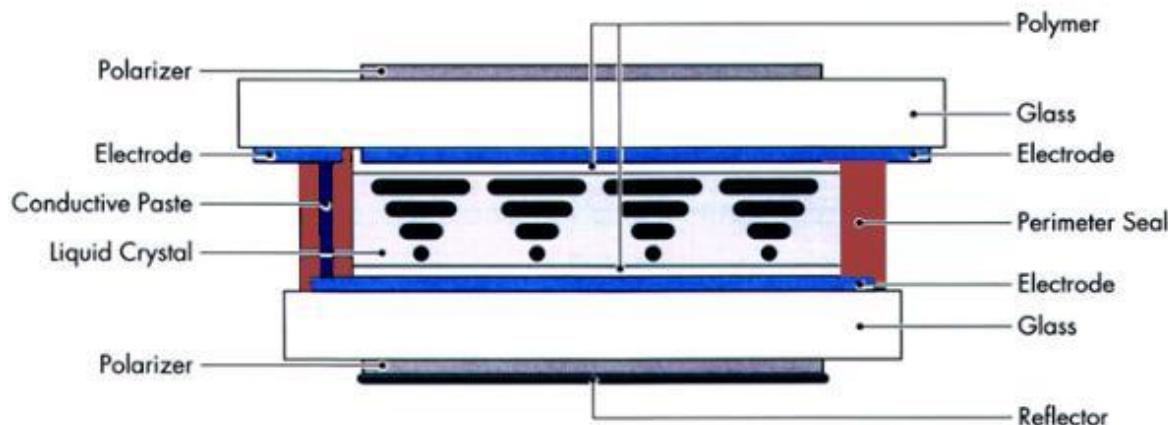


Figura 13.5 Estructura básica de un LCD  
©Ing. Juan Ricardo Penagos Plazas

Esta estructura helicoidal de las moléculas tiene la habilidad de controlar la luz. Se aplica un polarizador al frente y un analizador/reflector en la parte posterior de la celda (nótese que el analizador también es un polarizador, se le ha dado este nombre para distinguirlo del polarizador frontal). Cuando pasa luz polarizada aleatoriamente a través del polarizador frontal, ésta se polariza linealmente. Pasa a través del vidrio frontal, gira (guiada por las moléculas de cristal líquido) y pasa a través del vidrio posterior. Si el analizador está girado 90° con respecto al polarizador, la luz pasará a través del analizador y será reflejada nuevamente a través de la celda. El observador verá el fondo de la pantalla, que en este caso es el gris plateado del reflector.

### 13.2.1 Efecto de campo del LCD

Field Effect of LCD

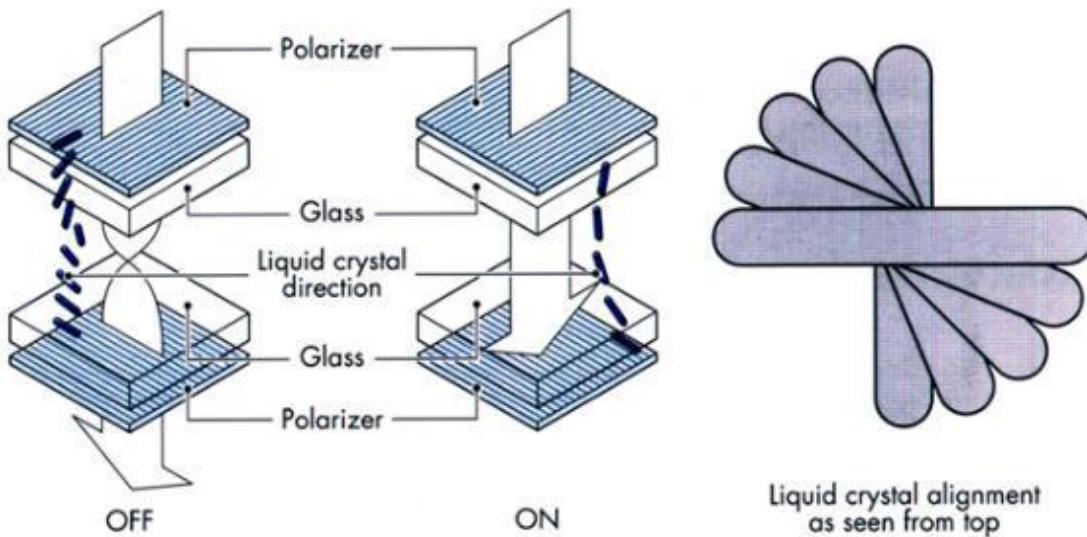


Figura 13.6 Efecto del campo eléctrico sobre las moléculas del LCD

El vidrio del LCD tiene conductores eléctricos transparentes sobre cada una de las dos superficies internas en contacto con el cristal líquido; estos conductores se emplean como electrodos. Cuando se aplica una señal apropiada a estos electrodos, aparece un campo eléctrico a través de la celda. Las moléculas de LC giran en la dirección del campo eléctrico. La luz entrante, polarizada linealmente, pasa sin verse afectada y es absorbida por el analizador. El observador ve un carácter negro sobre un fondo gris plateado (figura 13.6). Cuando se elimina el campo eléctrico, las moléculas retornan a su posición ortogonal original. Este tipo de presentación se conoce como imagen positiva, modo de visión reflectante. A partir de esta tecnología básica se construyen LCDs con múltiples electrodos que se pueden seleccionar en el momento de aplicar voltaje, con lo cual se logra mostrar una gran variedad de imágenes.

Se han producido muchos avances en los LCDs TN. El cristal líquido *Super Twisted Nematic (STN)* ofrece mayor ángulo de giro ( $\geq 200^\circ$ ), que proporciona mayor contraste y mejor ángulo de visión. Sin embargo, una característica negativa es la refracción, que modifica el fondo y lo torna verde-amarillo, mientras que el color de los caracteres se torna azul. El color de fondo se puede cambiar a gris por medio de un filtro especial.

El avance más reciente ha sido la introducción de las pantallas *Film compensated STN (FSTN)*. Estas añaden una película especial a la pantalla STN y compensan el color producido por la refracción. Con esto se logra una presentación en blanco y negro.

## 13.3 TIPOS DE LÍQUIDOS PARA LCDs

*TwistedNematic (TN)*, *Super Twisted Nematic (STN)*, *Film Compensated STN (FSTN)* y *Color STN (CSTN)* son los términos usados para describir cuatro tipos de LCDs. Cada uno cambia la orientación de la luz que pasa a través de la estructura del LCD de forma diferente, para producir un efecto en el contraste y la coloración.

### 13.3.1 LCDs Twisted Nematic (TN)

Las pantallas TN tienen un giro (la rotación de las moléculas desde una de las caras internas de la pantalla hasta el otro) de  $90^\circ$ . El LCD TN básico consta de una capa de cristal líquido (LC) entre dos capas de vidrio. El LC está conformado por moléculas de forma cilíndrica alargada con diferentes propiedades eléctricas y ópticas, dependiendo de la dirección. En las caras internas de las placas de vidrio hay electrodos transparentes,

que tienen la forma de la imagen deseada. Las superficies internas están cubiertas con un polímero que hace que las moléculas de LC en contacto con una de las superficies se encuentren perpendiculares a las moléculas en contacto con la otra superficie. A lo largo de la película de LC las moléculas alcanzan un giro de 90°, formando una estructura helicoidal. En las caras externas de las placas de vidrio se colocan polarizadores de luz paralelos a las moléculas de LC y perpendiculares entre sí. En el estado de apagado (OFF) la luz que atraviesa el primer polarizador es guiada por el giro de la capa de LC hasta el segundo polarizador, a través del cual se transmite. Cuando se enciende (ON), el LC se alinea con el campo eléctrico; la luz transmitida a través del primer polarizador es bloqueada por el segundo polarizador, formando una imagen obscura. El efecto puede invertirse si los polarizadores se colocan paralelos entre sí, y por lo tanto se formará una imagen clara sobre un fondo oscuro. La tecnología TN viene en un solo color: caracteres negros sobre un fondo gris. Es la más barata, pero tiene la menor calidad visual en lo referente al ángulo de visión.

### 13.3.2 LCDs High Twisted Nematic (HTN)

Las pantallas HTN se basan en un mayor giro molecular (usualmente 110°) que las pantallas TN y por lo tanto ofrecen mayores ángulos de visión y mejor contraste. De hecho, los resultados son muy parecidos a los de la tecnología STN. Ya que operan con voltajes tan bajos como 2,5V y el incremento en costos es mínimo con respecto a la tecnología TN se pueden aplicar en instrumentos manuales.

### 13.3.3 LCDs Super Twisted Nematic (STN)

Aunque los LCDs TN pueden ser manejados con técnicas de multiplexado para incrementar la cantidad de información presentada, están restringidos debido a su contraste reducido y ángulo de visión limitado. Para mejorar estas características se emplea la tecnología *Super Twist*. Los LCDs STN tienen un giro entre 90 y 360°. Actualmente la mayoría de pantallas STN se fabrican con un giro entre 180 y 270°. Los polarizadores no se montan paralelos a las moléculas de LC. Por lo tanto el funcionamiento no se basa en el principio de guía de luz, como en TN, sino más bien en la refracción. La posición de los polarizadores, el espesor de la celda, y la refracción del LC se seleccionan cuidadosamente para producir un color particular en el estado de apagado. Usualmente es un color verde-amarillo para maximizar el contraste.

La tecnología STN viene en dos colores: STN verde y STN plata. El STN verde tiene caracteres negro-violetas sobre fondo verde. El STN plata tiene caracteres negro-azules sobre fondo plateado. En términos de costos se encuentra en el medio, pero tiene muy buena calidad visual. El contraste es similar a la tecnología TN.

### 13.3.4 LCDs Film Compensated STN (FSTN)

El avance más reciente ha sido la introducción de las pantallas FSTN. Estas añaden una película a la pantalla STN lo cual compensa el color producido por la refracción, de esta forma se obtienen pantallas en blanco y negro con mejor contraste y mayor ángulo de visión. La tecnología FSTN viene en un solo color: caracteres negros sobre fondo gris. Es la más cara, pero tiene mejor ángulo de visión y mayor contraste que la tecnología STN.

### 13.3.5 LCDs Double Super Twisted Nematic (DSTN)

Esta fue la primera versión comercial en blanco y negro de la pantalla STN. Las pantallas DSTN realmente son dos celdas STN unidas. La primera es un LCD, la segunda es una celda de vidrio sin electrodos ni polarizadores, llena con material LC que actúa como compensador incrementando el contraste y dando una apariencia de blanco y negro. Las pantallas DSTN proporcionan mejor contraste que las STN y FSTN, y compensación automática de contraste con los cambios de temperatura. Su tiempo de respuesta es significativamente mayor. La tecnología DSTN reduce la tendencia de una pantalla a tener color rojo, verde o azul. Debido a su modo negativo de polarizador, necesitan alumbrado de fondo que se obtiene por LED o CCFL. Su resolución llega hasta 122x32 puntos y son apropiadas para automóviles, bombas de gasolina, etc.

### 13.3.6 LCDs Color Super Twisted Nematic (CSTN)

La tecnología CSTN realmente es una tecnología STN que emplea luz blanca de fondo y filtros de color para producir los tonos necesarios para una pantalla a color. Cada pixel de una pantalla CSTN está conformado

realmente por 3 pixeles separados que usan un filtro coloreado de rojo, verde y azul. Cada uno de los colores se controla individualmente por el chip controlador de gráficos. Así, una pantalla CSTN de 320x240 pixel realmente contiene 960x240 pixeles coloreados individualmente.

## 13.4 MODOS DE LCD

### 13.4.1 Modo positivo



*Figura 13.7 Imagen en modo positivo*

Una imagen positiva sobre un LCD es opaca (no deja pasar luz) cuando el pixel está ON, y transparente (deja pasar luz) cuando el pixel está OFF. En la mayoría de las pantallas la imagen es más pequeña que el fondo, de esta manera este modo de operación se ve favorecido en aplicaciones donde hay bastante luz ambiental, lo que ayudará a mejorar el contraste, especialmente para una pantalla que emplee un polarizador reflectante de fondo. Como ejemplo se tiene un carácter alfanumérico sobre un gran fondo. Los segmentos o puntos del carácter absorberían la luz (se verían oscuros) y el fondo (el área mayor) reflejaría la luz lo que resaltaría los caracteres. A continuación se tiene una lista de varias combinaciones típicas del Modo de operación y el Modo de visión y las imágenes resultantes (asumiendo que no hay luz de fondo que pueda colorear el fondo):

- TN: Caracteres negros sobre fondo gris.
- STN verde: Caracteres negro-violetas sobre fondo verde.
- STN plata: Caracteres negro-azules sobre fondo plateado.
- FSTN: Caracteres negros sobre fondo gris.

### 13.4.2 Modo negativo



*Figura 13.8 Imagen en modo negativo*

Una imagen negativa sobre un LCD es opaca (no deja pasar luz) cuando el pixel está OFF, y transparente (deja pasar luz) cuando el pixel está ON. Ya que el área de imagen típicamente es más pequeña que el fondo, la porción de la pantalla que pudiera reflejar la luz y dar definición a los caracteres queda minimizada. Por lo tanto, este modo se emplea normalmente cuando hay luz de fondo y la luz ambiental es pobre. Al emplear luz de fondo los segmentos transparentes de la pantalla brillarán debido a que la luz de fondo será visible cuando los pixeles estén ON. Si hay mucha luz ambiental se contrarrestará el efecto de la luz de fondo. A continuación se tiene una lista de varias combinaciones típicas del Modo de operación y el Modo de visión y las imágenes resultantes (asumiendo que hay luz de fondo con el color especificado):

- TN: Caracteres verde-amarillos brillantes sobre fondo gris (luz de fondo verde-amarilla).
- STN (Azul negativo): Caracteres amarillo-verdes brillantes sobre fondo azul (luz de fondo amarillo-verde).
- FSTN: Caracteres blancos brillantes sobre fondo negro (luz de fondo blanca).

## 13.5 MODOS DE POLARIZADOR

Cada LCD tiene dos polarizadores de luz, el frontal y el de fondo. El polarizador frontal siempre permite el paso de la luz y no puede ser seleccionado por el usuario, sin embargo, el polarizador de fondo tiene tres opciones y dos grados para cada opción.

### 13.5.1 Polarizador reflectante

Las pantallas reflectantes tienen un polarizador opaco de fondo que incluye un reflector difuso que puede ser de aluminio. Esta capa refleja la luz ambiental polarizada que ha ingresado por el frente de la pantalla. Las pantallas reflectantes requieren luz ambiental para poder verse. Muestran gran brillo, contraste excelente y grandes ángulos de visión. Son apropiadas para equipos operados por batería donde haya un nivel adecuado de luz ambiental. Estas pantallas no tienen luz de fondo, aunque pueden ser iluminadas frontalmente en algunos casos.

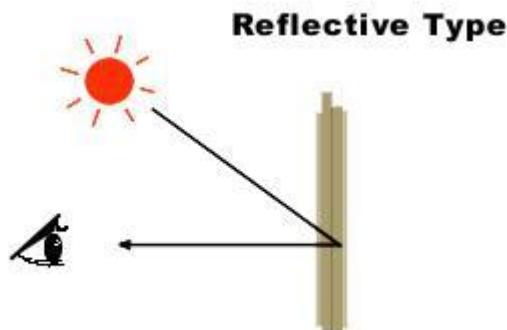


Figura 13.9 Pantalla reflectante

### 13.5.2 Polarizador transmisor

Las pantallas transmisoras tienen un polarizador transparente en el frente y en el fondo. La pantalla depende, por tanto, de la luz que provenga desde el fondo de la pantalla hacia el observador. La mayoría de estas pantallas generan una imagen negativa, y tienen filtros coloreados en algunos sectores para resaltar diferentes anuncios. Otro ejemplo de pantalla con polarizador transmisor pudiera ser una ventana transparente donde se verían los segmentos intercalados en la línea de vista del observador (se asume que existe suficiente luz ambiental en cualquiera de las caras de la ventana).

### Transmissive Type

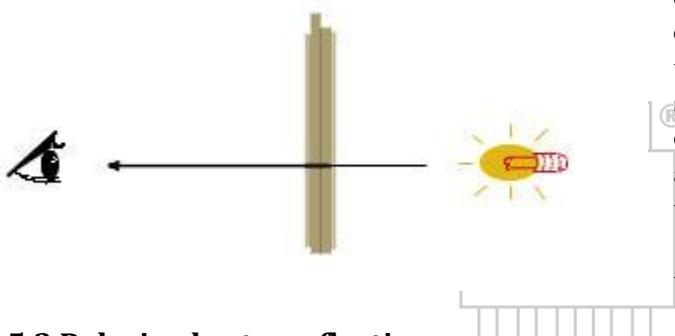
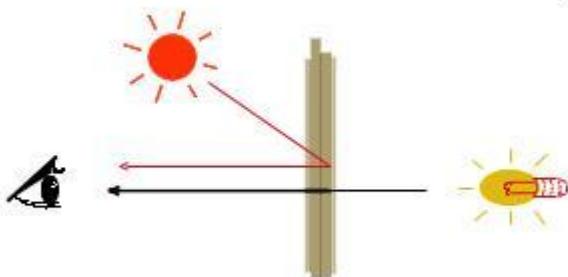


Figura 13.10 Pantalla transmisora

### 13.5.3 Polarizador transflectivo

Estas pantallas tienen un polarizador de fondo con un material translúcido que refleja parte de la luz ambiental y transmite la luz de fondo. Como el nombre lo indica, es una combinación de los modos transmisor y reflectante.

### Transflective Type



En el modo de reflexión no es tan brillante y tiene menor contraste con respecto al LCD reflectante, pero puede ser iluminado desde el fondo para emplearse en condiciones de poca luz ambiental. Esta es la mejor opción para una pantalla que se pueda usar en cualquier condición de luz ambiental con una luz de fondo.

Figura 13.11 Pantalla transflectiva

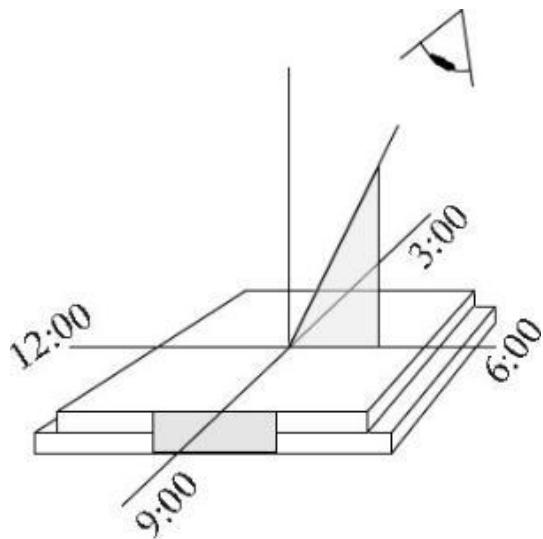
### 13.5.4 Polarizador de grado comercial

El grado de un polarizador determina el rango de temperaturas de almacenamiento y operación. El polarizador de grado comercial puede operar entre -10°C y 60°C.

### 13.5.5 Polarizador de grado industrial

El grado de un polarizador determina el rango de temperaturas de almacenamiento y operación. El polarizador de grado industrial puede operar entre -30°C y 80°C. Existen otros rangos de temperatura disponibles.

## 13.6 ÁNGULO DE VISIÓN



El ángulo de visión (o dirección de observación) es la dirección desde la cual se puede ver de mejor forma la imagen en la pantalla. Se configura durante el proceso de manufactura y no se puede cambiar después. Se especifica por las horas del reloj. Una dirección de observación de 12h00 significa que la dirección óptima está por encima de la normal a la pantalla (la normal es la perpendicular al plano de la pantalla), mientras que una pantalla de 6h00 se ve mejor por debajo de la normal.

Figura 13.12 Dirección de observación de los LCDs

Cuando se especifica la dirección de observación, se debe tener en cuenta el uso que se va a dar al dispositivo. Por ejemplo, una calculadora normalmente se ubica sobre un escritorio o en la palma de la mano, por lo tanto su dirección de observación es 6h00. Algunos instrumentos, como los termostatos de pared, pueden estar ubicados por debajo del observador y deben ser vistos desde la dirección 12h00. Hay otras posibilidades que no son muy frecuentes. La pantalla del reloj en un automóvil, que se encuentra normalmente a la derecha del conductor, puede tener una dirección de observación de 9h00, o 10h30 si el reloj está en la parte baja del tablero de instrumentos.

En una pantalla de accionamiento directo la dirección de observación no es crítica debido a que la pantalla se verá bien casi desde cualquier dirección. Se vuelve crítica cuando la pantalla es multiplexada. A mayor velocidad de multiplexado, mayor es el problema. En pantallas con velocidades de multiplexado extremadamente altas, se debe tener mucho cuidado cuando se diseña el circuito de accionamiento. Se pueden aplicar películas especiales al frente de la pantalla, pero esto incrementa el costo.

## 13.7 RANGOS DE TEMPERATURA

Cuando se selecciona un módulo LCD, es muy importante identificar el intervalo de temperatura ambiental donde va a operar. Los LCDs estándar no soportan un amplio intervalo de temperatura, y ese intervalo puede verse afectado por la radiación solar, la ventilación y la época del año. Además, se debe tomar en cuenta que la luz de fondo y el circuito de control pueden influir sobre la temperatura ambiental del LCD.

Otra cosa que se debe tener en cuenta es que el contraste se modifica con el cambio de temperatura. El diseño del circuito electrónico debería tener alguna forma de compensar estas variaciones de contraste. También se puede seleccionar un módulo LCD que ya incorpore estos circuitos de compensación.

### 13.7.1 Rangos de temperatura TN

El rango de temperaturas de almacenamiento y operación de la pantalla TN depende del control y el voltaje de alimentación. Los siguientes parámetros se deberían tomar como referencia:

- **Rango estándar de temperatura TN:**

Temperatura de operación: 0 a 50°C.

Temperatura de almacenamiento: -10 a 60°C.

- **Rango extendido de temperatura TN:**

Temperatura de operación: -20 a 70°C.

Temperatura de almacenamiento: -30 a 80°C.

- **Rango extendido de temperatura TN:**

Temperatura de operación: -30 a 80°C.

Temperatura de almacenamiento: -35 a 85°C.

### 13.7.2 Rangos de temperatura STN/FSTN

El rango de temperaturas de almacenamiento y operación de las pantallas STN y FSTN se encuentra en dos intervalos para los productos estándar:

- **Rango estándar de temperatura STN/FSTN:**

Temperatura de operación: 0 a 50°C.

Temperatura de almacenamiento: -10 a 60°C.

- **Rango extendido de temperatura STN/FSTN:**

Temperatura de operación: -20 a 70°C.

Temperatura de almacenamiento: -30 a 80°C.

## 13.8 LUZ DE FONDO DEL LCD

Los LCDs crean las imágenes con la manipulación de la luz ambiental visible. En ausencia de esta luz, se tiene que añadir luz de fondo para que las imágenes sean visibles. Hay muchas opciones para agregar alumbrado de fondo a un LCD. Cada una tiene sus ventajas y desventajas, y no hay un único método para todas las aplicaciones.

### 13.8.1 Alumbrado de fondo con LED (*LED backlighting*)

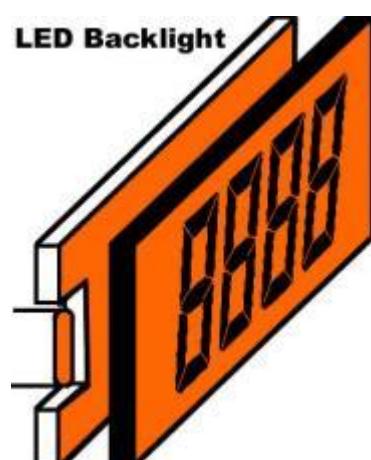


Figura 13.13 Alumbrado de fondo con LED

Esta es la técnica más popular para LCDs medianos y pequeños. Las ventajas de los LEDs son su bajo costo, larga vida, inmunidad a la vibración, bajo voltaje de operación, y control de intensidad preciso. La principal desventaja es que requiere más potencia que la mayoría de los demás métodos. Los LEDs de fondo vienen en muchos colores, siendo el amarillo-verde el más común, mientras que el blanco se está volviendo muy popular y de bajo costo. La vida de operación es muy larga (50.000 horas como mínimo), y son más brillantes que los ELPs. Al ser de estado sólido, se configuran para operar con 5 o 12 V<sub>DC</sub>, así que no requieren un inversor.

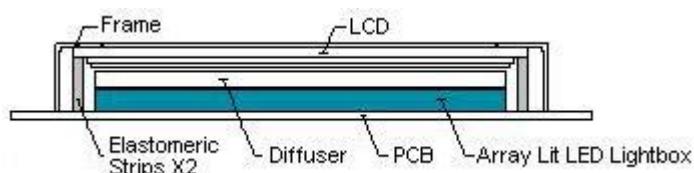


Figura 13.14 Configuración matricial (array) de LEDs de fondo

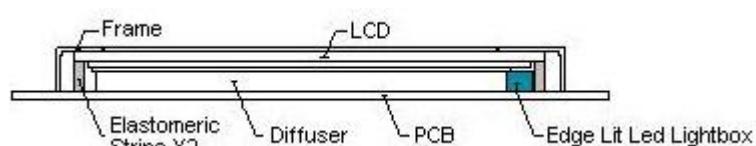


Figura 13.15 Configuración de borde (edge) de LEDs de fondo

El LED de fondo tiene dos configuraciones básicas: matricial (*array*) y de borde (*edge*) (figuras 13.14 y 13.15). En ambos tipos la luz de los LEDs se enfoca sobre un difusor que distribuye la luz uniformemente detrás de la pantalla. En la configuración matricial hay muchos LEDs distribuidos uniformemente tras la pantalla, esto produce un alumbrado más uniforme y brillante y consume más potencia. En la configuración de borde, los LEDs se ubican en una de las orillas (típicamente la orilla superior) y se enfocan sobre el borde del difusor, esto permite tener un módulo más delgado con menor consumo de potencia.

### 13.8.2 Panel electroluminiscente (*Electroluminescence Panel ELP*)

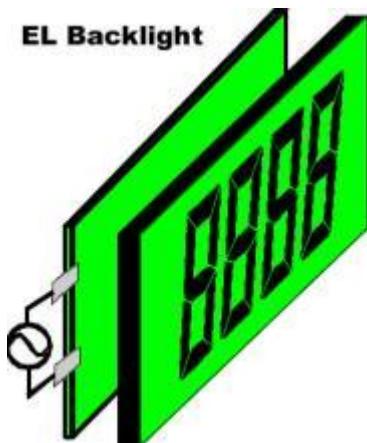


Figura 13.16 Alumbrado con panel electroluminiscente

La electroluminiscencia (EL) es un fenómeno del estado sólido que emplea fósforo de color, sin calor, para generar luz. Los paneles EL son muy delgados, ligeros y proporcionan luz uniforme. Están disponibles en muchos colores, siendo el blanco el más popular entre los LCDs. Mientras que su consumo de potencia es bastante bajo, requieren voltajes de 100V<sub>AC</sub> a 400 Hz. Esto se obtiene de un inversor que convierte una fuente de 5, 12 o 24V<sub>DC</sub> al voltaje AC requerido. Los ELPs tienen una vida limitada de 3.000 a 5.000 horas hasta reducir su brillo a la mitad. Las mayores desventajas de los ELPs son la necesidad de un inversor y su vida limitada.

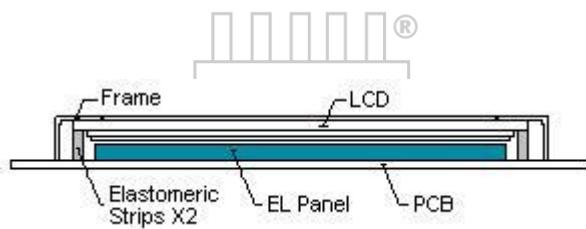


Figura 13.17 Detalles del alumbrado ELP.

### 13.8.3 Lámpara fluorescente de cátodo frío (*Cold Cathode Fluorescent Lamp CCFL*)



Figura 13.18 Alumbrado con lámpara fluorescente de cátodo frío

La luz de fondo CCFL ofrece bajo consumo de potencia y una luz blanca muy brillante. Se emplea una configuración de borde para el alumbrado de los LCDs. Se emplea un difusor para distribuir la luz de manera uniforme. Las lámparas CCFL requieren un inversor para suministrar los 270 a 300 V<sub>AC</sub> con una frecuencia de 35kHz requeridos. Se emplean fundamentalmente en los LCDs para gráficos y tienen una vida más larga que los ELPs (de 10.000 a 20.000 horas). Las mayores desventajas son: el clima frío puede reducir la emisión de luz hasta en un 60% (figura 13.19), requieren un inversor, no se puede ajustar la intensidad de la luz, y las vibraciones pueden reducir la expectativa de vida hasta en un 50%.

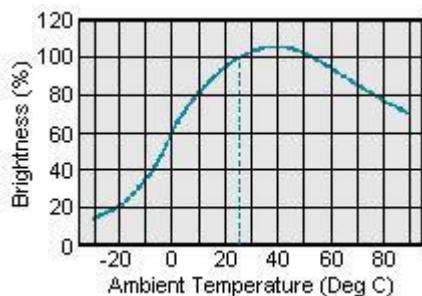


Figura 13.19 Brillo de la lámpara CCFL en función de la temperatura ambiental

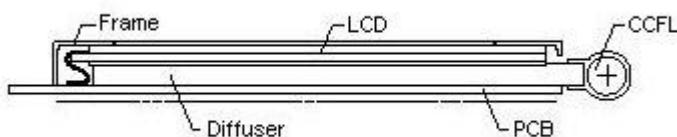


Figura 13.20 Detalles del alumbrado CCFL

### 13.8.4 Malla de fibra óptica en zigzag (*Woven Fiber Optic Mesh WFOM*)



Figura 13.21 Alumbrado con malla de fibra óptica

El alumbrado WFOM proporciona una luz de fondo muy uniforme, sin necesidad de inversor. El tiempo de vida depende del tipo de lámpara que se emplee; con luz halógena (que genera mucho calor) o LEDs se pueden obtener hasta 100.000 horas. Las lámparas se ubican normalmente lejos del LCD, donde se puedan remplazar fácilmente. Los paneles WFOM son algo más costosos, pero el costo extra se ve compensado por la uniformidad y el brillo.

[www.programarpicenc.com](http://www.programarpicenc.com)  
©Ing. Juan Ricardo Penagos Plazas

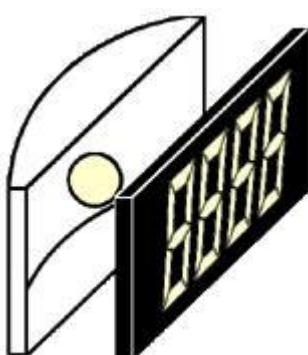


Figura 13.22 Alumbrado incandescente

Esta técnica se emplea cuando el costo es el factor de mayor importancia. Si bien las lámparas incandescentes son muy brillantes, por otro lado no son uniformes, generan bastante calor, tienen cortos períodos de vida y emplean bastante potencia. Pueden proporcionar luz blanca, pero el color puede variar con el voltaje de alimentación, además son sensibles a los golpes y las vibraciones.

## 13.9 TIPOS DE MONTAJES DE MÓDULOS LCD

Los tipos de montajes se refieren a cómo han sido instalados los chips controladores del LCD. Los tipos más comunes son SMT, COB, TAB y COG. Hay otras variantes basadas en estos tipos.

### 13.9.1 Tecnología de montaje superficial (*Surface Mount Technology SMT*)

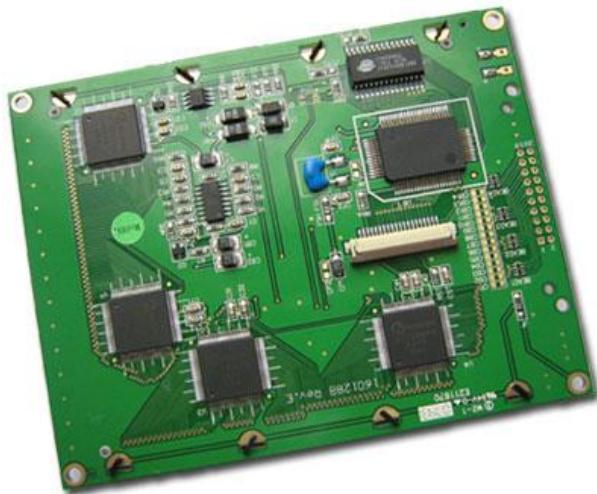


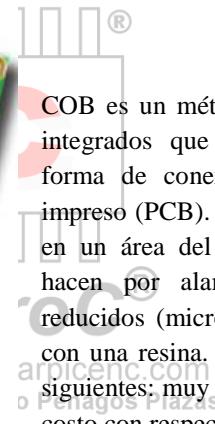
Figura 13.23 Circuito impreso tipo SMT

El uso de chips delgados (con pinos de conexión en sus cuatro lados) sobre tableros de circuito impreso es la técnica más popular en la industria del LCD, y se encuentra disponible para producción en masa de la gran mayoría de módulos LCD. Es el método más confiable y robusto para el montaje de LCDs.

### 13.9.2 Chip en tablero (Chip On Board COB)



Figura 13.24 Circuito impreso tipo COB



COB es un método popular de montaje de circuitos integrados que utiliza uniones de alambre como forma de conexión directa al tablero de circuito impreso (PCB). El controlador del LCD se construye en un área del PCB. Las conexiones eléctricas se hacen por alambres de oro de diámetros muy reducidos (micro alambres). Luego se cubre el área con una resina. Las ventajas de este método son las siguientes: muy compacto, ahorro de espacio y menor costo con respecto a la técnica SMT.

### 13.9.3 Unión automatizada de cinta (Tape Automated Bonding TAB)



Figura 13.25 Circuito impreso tipo TAB

El controlador del LCD se encierra en una burbuja fuerte y delgada, de la cual salen los terminales de conexión sobre un sustrato plástico delgado. Se emplea un borde adhesivo para pegar los terminales al LCD o al PCB. La técnica TAB usa el mismo tipo de circuitos integrados que la tecnología COG.

### Ventajas:

- Muy compacto (la circuitería de control se puede colocar detrás del propio LCD).
- En algunas ocasiones es más eficiente en costo que la tecnología COG, si el diseñador tiene que integrar un teclado o indicador alrededor de la pantalla.
- El área activa se encuentra centrada (a diferencia de la tecnología COG).
- Sirve en aplicaciones con separaciones (*pitch*) muy reducidas.

### Desventajas:

- El área de unión es débil. Menos confiable que la técnica COG.
- Más caro que la técnica COG. Aún cuando los módulos TAB usan el mismo tipo de circuitos integrados que los módulos COG, la tecnología TAB requiere un encapsulado.

#### 13.9.4 Chip en vidrio (*Chip On Glass COG*)



Figura 13.26 LCD gráfico con tecnología COG

La tecnología COG es una de las técnicas de alta tecnología que emplea circuitos integrados especiales, y se implementa en la mayoría de aplicaciones compactas. Estos circuitos integrados fueron introducidos por Epson. El circuito integrado no está encapsulado, sino montado directamente sobre el PCB (tablero de circuito impreso). Como no hay encapsulado, el espacio del chip se puede reducir, así como el tamaño del PCB. Esta técnica es muy apropiada para manejar señales de alta frecuencia.

### Ventajas:

- Mucho ahorro de espacio. Los módulos LCD COG pueden tener un espesor de 2mm.
- Muy efectivo en costos con respecto a la técnica COB, especialmente en LCDs para gráficos.
- Más confiable que la técnica TAB, debido a la debilidad de la conexión de esta última.

### Desventajas:

- Puede usarse hasta un cierto nivel de resolución donde las líneas no son muy finas. Cuando la separación (*pitch*) es muy pequeña se prefiere la técnica TAB.
- Las técnicas TAB o COB pueden ser más efectivas en costos, si el diseñador tiene que integrar un teclado o indicador alrededor de la pantalla.
- El área activa no se encuentra centrada, debido al espacio que ocupan los circuitos.

#### 13.10 CIRCUITERÍA DE CONTROL DEL LCD: ESTÁTICO

La configuración para la técnica de control estático del LCD es aquella en la que hay un punto común (plano de fondo) para uno de los extremos de cada segmento, y los otros extremos de los segmentos se dirigen a puntos individuales de conexión que se unen al circuito de control. Este método usa un gran número de interconexiones y no es factible para pantallas complejas, aunque produce la mejor presentación.

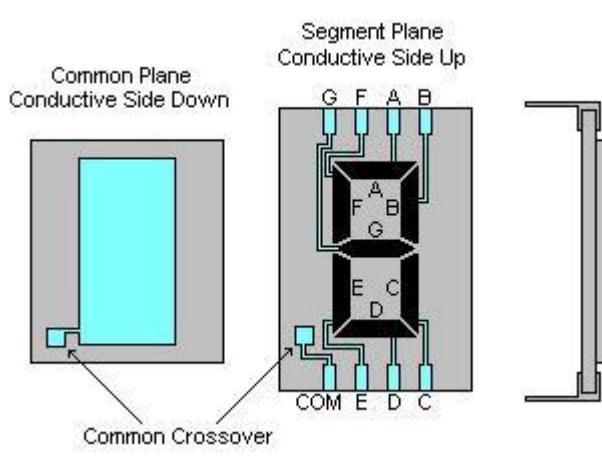


Figura 13.27 Control estático del LCD

Los LCDs requieren una fuente de AC sin componente DC. Los segmentos se controlan por la magnitud del voltaje AC sobre el segmento del LCD, pero siempre tiene que haber voltaje AC sobre todos los segmentos del LCD. La operación prolongada con DC puede causar reacciones electromecánicas dentro de las pantallas, lo que reducirá la vida útil de manera significativa. Los primeros síntomas de degradación de la pantalla debido a la corriente directa son la pérdida de alineación a lo largo de los bordes, y la apariencia borrosa de algunos caracteres.

El LCD TN es un dispositivo dependiente del valor eficaz (RMS), esto es, el contraste de un segmento depende del valor RMS del voltaje aplicado, medido con respecto al plano de fondo.

Las frecuencias para las pantallas de excitación directa están entre los 30 y los 100Hz. Dependiendo del tamaño de la pantalla, estas pueden ser operadas a mayores frecuencias, pero esto incrementa el consumo de potencia. Los LCDs se comportan como una carga capacitiva, lo que reduce la impedancia con el incremento de la frecuencia. La operación por debajo de los 30 Hz da como resultado un parpadeo de la imagen.

Los LCDs pueden ser sobreexcitados por una combinación de voltaje y frecuencia, lo que produce fantasmas. Los fantasmas son la activación parcial de los segmentos que están apagados. Ya que la corriente es proporcional a la frecuencia, existe un producto voltaje-frecuencia que no debe ser superado. Estos valores dependen del diseño, así que es muy importante el diseño apropiado de la pantalla y la elección correcta de la excitación. También es muy importante que todos los segmentos no utilizados se conecten al plano de fondo, y así evitar que estén flotando.

### 13.11 CIRCUITERÍA DE CONTROL DEL LCD: MULTIPLEXADO

La configuración para la técnica de control multiplexado del LCD difiere de la técnica estática en que usa más de un punto común o plano de fondo. Con esta configuración, cada línea de control de segmento se puede conectar a tantos segmentos como planos de fondo existan, con la condición de que cada uno de los segmentos a los que se conecta se encuentren a su vez conectados a planos de fondo separados. Este método multiplexa cada línea de control de segmento y minimiza el número de conexiones. Es el método usado con pantallas complejas que tienen una superficie limitada para la interconexión o un número limitado de circuitos de control; por otra parte, la reducción en el número de conexiones externas aumenta la confiabilidad. La mayor tasa de multiplexado afectará la calidad de la pantalla, el rango de temperatura de operación, y puede ser necesario incrementar la complejidad de la circuitería de control (o tal vez del software del microcontrolador).

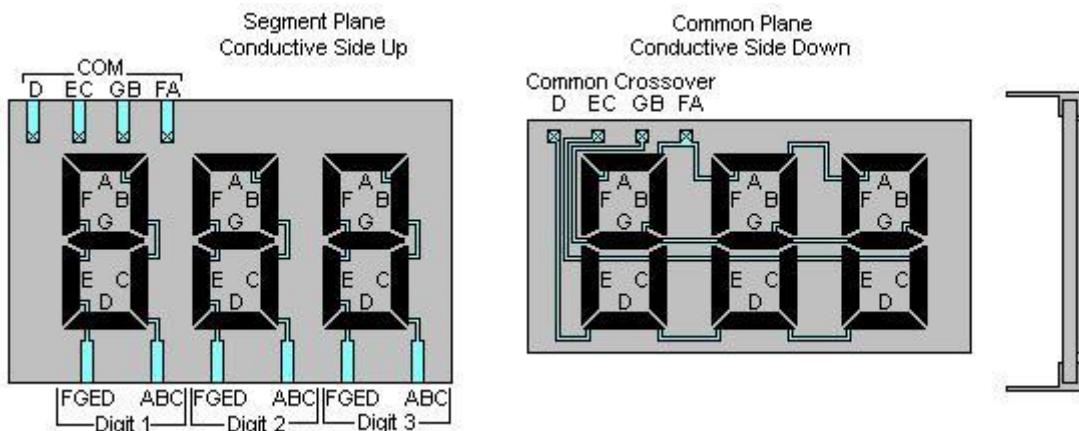


Figura 13.28 Control multiplexado del LCD

El método de control de pantallas multiplexadas consiste en la división del tiempo, siendo el número de fracciones de tiempo igual al doble del número de planos de fondo. Como es el caso de los LCDs convencionales, para evitar que los efectos electromecánicos destruyan la pantalla, el voltaje en todos los segmentos tiene que invertirse periódicamente, de tal forma que el valor medio sea igual a cero. Esta es la razón para duplicar la cantidad de fracciones de tiempo: Cada plano de fondo tiene que ser activado alternativamente con un pulso de voltaje de polaridad opuesta.

Igual que en las pantallas no multiplexadas, la frecuencia de control debe estar por encima de la frecuencia de parpadeo ( $>30\text{Hz}$ ). Debido a que el incremento de la frecuencia incrementa el consumo del circuito CMOS de control, y para evitar problemas debidos a la conductividad finita de los segmentos de la pantalla y de los electrodos que forman el punto común, se recomienda una frecuencia límite de 60 a 90Hz.

## 13.12 CONTROL DE CONTRASTE DEL LCD

El contraste de un LCD es la relación entre el área en blanco y el área oscura. Cuando se usa un LCD, se debe disponer de un control para ajustar el voltaje de polarización de la pantalla. Este ajuste permitirá el control del contraste entre los segmentos encendidos y apagados, y debe ser configurado para tener la mejor apariencia. Puede ser necesario realizar un ajuste dinámico del contraste durante el uso de los productos, debido a las variaciones extremas de temperatura.

### 13.12.1 Ajuste del contraste del LCD

El ajuste del contraste también modifica el ángulo de visión de la pantalla. Una pantalla de 12h00 se puede optimizar para ser vista desde las 6h00 por medio del ajuste del contraste; aunque el resultado no será tan bueno como una pantalla de 6h00 optimizada para ser vista desde las 6h00. Si se va a ver la pantalla de frente, se puede escoger una de 12h00 o una de 6h00 y realizar un ajuste ligero del contraste para optimizar la imagen desde esa posición.

Para el ajuste del contraste se emplea normalmente un potenciómetro (conectado entre Vdd y Vss, ó Vee y Vdd para módulos LCD de alto voltaje). El terminal móvil del potenciómetro se conecta a Vo (voltaje de contraste). Actuando sobre este potenciómetro se logra la apariencia deseada.

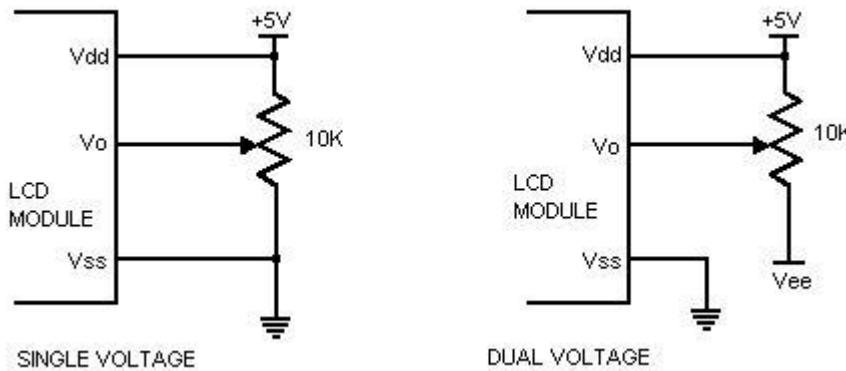


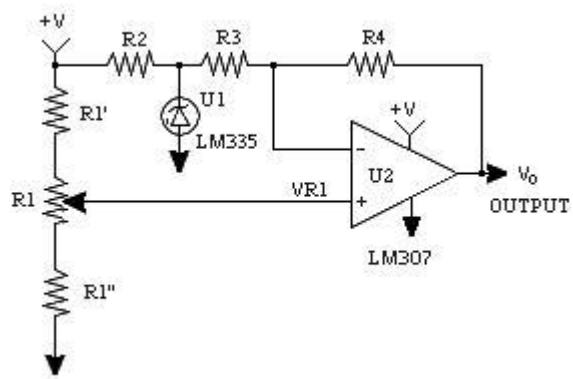
Figura 13.29 Circuitos de control de contraste

### 13.12.2 Ajuste de contraste con compensación de temperatura

Puede ser necesario compensar los cambios de contraste en aplicaciones expuestas a cambios extremos de temperatura. Puede ocurrir que el cristal líquido muestre imágenes desvanecidas a bajas temperaturas, y fantasmas a altas temperaturas, si no se ha empleado alguna técnica de compensación.

Una solución es optar por un módulo LCD que incorpore compensación de temperatura. En el caso de los módulos que no tienen esta opción incorporada, se puede fabricar el circuito de compensación, de acuerdo a los dos ejemplos de referencia que se muestran a continuación.

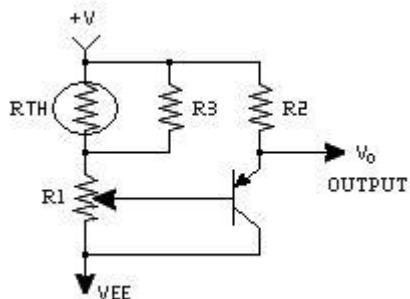
#### Ejemplo 1



La figura 13.30 muestra el circuito de compensación básico. El corazón del circuito es un sensor de temperatura LM335 (U1), que se debe ubicar cerca del LCD. El LM335 tiene una salida de  $110\text{mV}/^\circ\text{C}$ . R2 proporciona la corriente de operación a U1 (1mA). El amplificador U2 invierte y amplifica la salida de U1 de acuerdo a los requerimientos del LCD. El potenciómetro P1 permite ajustar el voltaje de operación (voltaje de contraste) del LCD.

Figura 13.30 Ajuste de contraste con compensación de temperatura

#### Ejemplo 2



En base a los requerimientos del LCD, se debe seleccionar un termistor ( $R_{th}+R_3$ ) para producir el desplazamiento de voltaje. R1 sirve para seleccionar el voltaje inicial correcto y el transistor Q1 actúa como amplificador de corriente, permitiendo que el termistor opere en condiciones de mínima carga. R2 se utiliza como elemento de polarización del transistor. VEE tiene que ser una fuente regulada.

Figura 13.31 Ajuste de contraste con compensación de temperatura

### 13.13 FUNCIONES DE mikroC™ PARA GLCD

mikroC PRO for PIC proporciona una librería de funciones para el trabajo con GLCDs que tengan incorporado el controlador Toshiba T6963C.

Función	Descripción
<code>T6963C_Init(ancho, alto, bits por carácter);</code>	Inicializa el controlador del GLCD.
<code>T6963C_WriteData(dato);</code>	Escribe un dato en el controlador.
<code>T6963C_WriteCommand(dato);</code>	Envía un comando hacia el controlador.
<code>T6963C_WaitReady();</code>	Lee el byte de estado y repite la lectura hasta que el controlador esté listo.
<code>T6963C_Fill(byte, inicio, longitud);</code>	Llena un bloque de memoria del controlador con el byte dado.
<code>T6963C_Dot(x, y, color);</code>	Dibuja un punto en las coordenadas (x, y) del color dado (blanco o negro).
<code>T6963C_Write_Char(carácter, x, y, modo);</code>	Escribe un carácter en las coordenadas x, y.
<code>T6963C_Write_Text(texto, x, y, modo);</code>	Escribe texto en las coordenadas x, y.
<code>T6963C_Line(x0, y0, x1, y1, color);</code>	Dibuja una línea desde (x0, y0) hasta (x1, y1).
<code>T6963C_Rectangle(x0, y0, x1, y1, color);</code>	Dibuja un rectángulo.
<code>T6963C_Box(x0, y0, x1, y1, color);</code>	Dibuja una caja (rectángulo lleno).
<code>T6963C_Circle(x, y, radio, color);</code>	Dibuja un círculo.
<code>T6963C_Image(imagen);</code>	Muestra un mapa de bits (imagen en formato bmp).
<code>T6963C_Set_Cursor(x, y);</code>	Ubica el cursor en la fila x, columna y.
<code>T6963C_PanelFill(byte);</code>	Llena completamente el panel actual (gráfico + texto) con el byte dado (0 para borrar).
<code>T6963C_GrFill(byte);</code>	Llena el panel gráfico con el byte dado (0 para borrar).
<code>T6963C_TxtFill(byte);</code>	Llena el panel de texto con el byte dado (0 para borrar).
<code>T6963C_Cursor_Height(n);</code>	Configura el tamaño del cursor (n=0...7).
<code>T6963C_Graphics(n);</code>	Habilita/deshabilita la presentación de gráficos (n=0 ó 1).
<code>T6963C_Text(n);</code>	Habilita/deshabilita la presentación de texto (n=0 ó 1).
<code>T6963C_Cursor(n);</code>	Enciende/apaga el cursor (n=0 ó 1).
<code>T6963C_Cursor_Blink(n);</code>	Habilita/deshabilita el parpadeo del cursor (n=0 ó 1).

Tabla 13.1 Funciones para trabajo con el GLCD

### 13.14 CONTROLADOR T6963C

El T6963C es un controlador para LCDs muy popular que se usa en módulos gráficos de pequeño tamaño. Es capaz de controlar pantallas con una resolución de hasta 240x128 puntos. Debido a su bajo consumo de potencia y tamaño reducido es más apropiado para aplicaciones móviles tales como PDAs, reproductores MP3 o equipo portátil de medición. Este controlador tiene la capacidad de mostrar y combinar texto (incluye varias funciones de atributos) y gráficos y maneja todas las señales de interfaz hacia los controladores de filas y columnas. Este dispositivo tiene un bus paralelo de datos de 8 bits, y líneas de control para lectura y escritura a través de un microcontrolador. Tiene una ROM de 128 palabras (generador de caracteres) y puede manejar una RAM externa de pantalla de hasta 64 Kbyte. Se puede asignar fácilmente RAM para el texto, los gráficos y la CGRAM externa, y la porción de memoria que se ve en la pantalla se puede seleccionar libremente dentro del rango asignado de memoria. El dispositivo soporta una gran variedad de formatos LCD permitiendo la selección a través de un grupo de entradas programables.

#### Características

- Formato de pantalla (se selecciona por pines)

- Columnas: 32, 40, 64, 80  
 Líneas: 2, 4, 6, 8, 10, 12, 14, 16, 20, 24, 28, 32
- Tamaño de fuente (se selecciona por pines)
  - Puntos horizontales: 5, 6, 7, 8
  - Puntos verticales: 8 (fijo)
  - Memoria externa de pantalla: 64Kbyte máximo.  
 Las direcciones en la memoria de pantalla (para el texto, los gráficos y el generador externo de caracteres) se determinan por software.
  - La lectura o escritura desde el microcontrolador no perjudica la presentación.
  - La RAM externa tiene que ser estática.
  - Las funciones de atributos se pueden usar únicamente en modo Texto.
  - ROM generador de 128 caracteres (CGROM) incorporado.
  - Bus paralelo de 8 bits y líneas de señal para interconexión con un microcontrolador. Los datos y comandos hacia y desde el microcontrolador son multiplexados en este bus.
  - Consumo de 3-4 mA máximo.
  - Oscilador de cristal incorporado.
  - Temperatura de operación de -20 a 70°C.

### 13.14.1 Definiciones

Símbolo	Significado
<b>MPU/MCU</b>	Micro Procesor/ControllerUnit
<b>SRAM</b>	RAM estática (mantiene el contenido mientras esté aplicado el voltaje).
<b>DRAM</b>	RAM dinámica (mantiene el contenido unos pocos microsegundos y debe refrescarse).
<b>CGROM</b>	ROM del generador de caracteres (memoria no volátil dentro del T6963C). Contiene caracteres predefinidos que se pueden mostrar fácilmente.
<b>VRAM</b>	RAM de video, memoria de pantalla o RAM externa. Aunque es una memoria externa, es la RAM para el T6963C y, por tanto, todas las operaciones de presentación se efectúan a través de esta RAM.
<b>240x128</b>	Ejemplo de resolución de pantalla. El primer número indica la resolución horizontal y el segundo la resolución vertical.

Tabla 13.2 Definiciones básicas para GLCD

### 13.14.2 Diagrama general de aplicación del T6963C

Como se puede ver el T6963C se conecta al LCD a través de los controladores de filas y columnas. Por otro lado, se conecta al microcontrolador y a la RAM externa por medio de las líneas de datos, direcciones y control disponibles para ello. Por supuesto, la conexión a una pantalla y a un microcontrolador (MCU) necesita el software adecuado. Por software se entiende el conjunto de comandos desde el MCU; el T6963C no puede funcionar sin un dispositivo de control (un microcontrolador). El control se logra por medio de comandos y datos desde el MCU. No es posible grabar un programa o instrucciones en el T6963C; cada operación y parámetro de configuración tiene que venir desde el MCU.

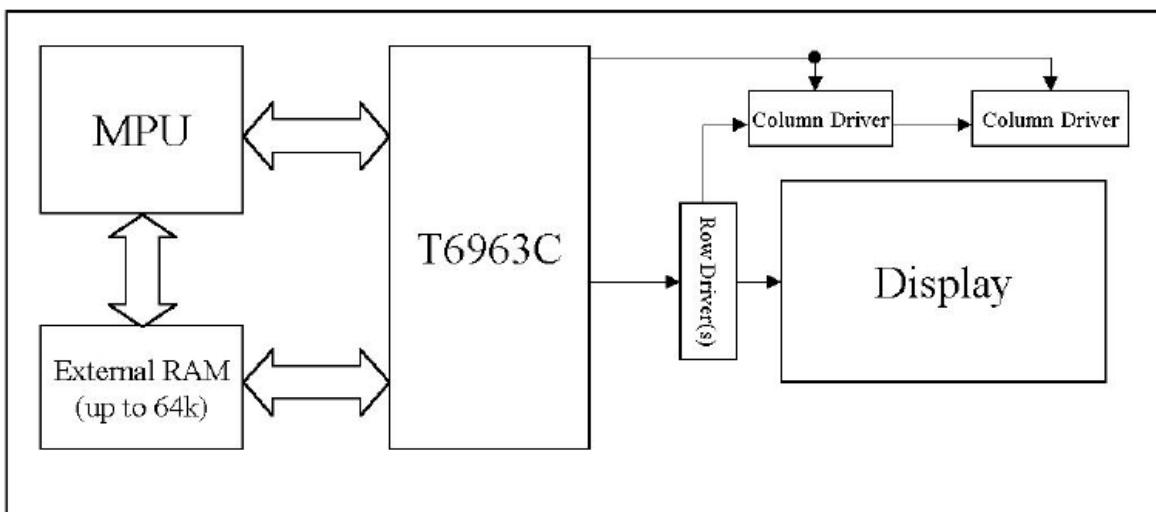


Figura 13.32 Diagrama de bloques de aplicación del T6963C

### 13.14.3 Conexión a una pantalla

#### Single Scan (1 Screen) / Dual Scan (2 Screens)

El T6963C se usa para manejar los controladores de filas y columnas de pantallas gráficas STN con una resolución de hasta 240x128 pixeles. Los controladores de filas y columnas se conectan al T6963C a través de varias señales de temporización y una o dos líneas de datos en serie. Estas líneas varían dependiendo de si la pantalla es *Single Scan* o *Dual Scan*. Las pantallas *Single Scan* tienen una resolución de, por ejemplo, 240x128 pixeles y tienen tres controladores de columnas y dos controladores de filas (los controladores de columnas pueden manejar 80 líneas, y los controladores de filas 64 líneas). Las pantallas *Dual Scan* se dividen en una mitad superior y una mitad inferior. Una pantalla 240x128 tendría aún 2 controladores de filas, pero 6 controladores de columnas; tres controladores para la mitad superior y tres para la mitad inferior. El método *Dual Scan* se usa para incrementar el contraste de las pantallas STN (se requiere menos voltaje para lograr el mismo contraste de una pantalla *Single Scan*).

### 13.14.4 Conexión a un microcontrolador y a la RAM de video

#### Memoria de video VRAM

El T6963C puede controlar una memoria de video de hasta 64Kbyte (tiene que ser RAM estática (SRAM)). Tiene 16 líneas de dirección, 8 líneas de datos y varias líneas de control de acceso a la VRAM. El T6963C se diferencia de la mayoría de controladores de LCD en el uso de la VRAM. Normalmente se asigna un área fija de memoria para el texto, los gráficos y el generador de caracteres, pero con el T6963C el tamaño de cada área se puede configurar por software.

#### Conexión al microcontrolador

La comunicación entre el T6963C y el microcontrolador (MCU) es sencilla y sólo hay algunas consideraciones a tener en cuenta. El T6963C tiene un bus de datos de 8 bits y varias señales para la comunicación con el MCU. Las tablas siguientes resumen las funciones de los pines del T6963C que se emplean para la conexión al MCU.

Pines		Descripción
<b>D&lt;7:0&gt;</b>		Pines de entrada/salida (E/S) de datos para la comunicación entre el MCU y el T6963C.
<b>#WR</b>		Escritura de datos. Tiene que configurarse en nivel bajo para escribir datos en la VRAM a través del T6963C.
<b>#RD</b>		Lectura de datos. Tiene que configurarse en nivel bajo para leer datos de la VRAM a través del T6963C.
<b>#CE</b>		Pin de habilitación. Tiene que configurarse en nivel bajo mientras el MCU se comunica con el T6963C.
<b>C/#D</b>		Selección de Comando o Datos. Para leer o escribir datos se debe configurar en nivel bajo. Para escribir un comando o leer el estado se debe configurar en nivel alto.
<b>VDD</b>		Voltaje de alimentación (+5VDC). Si la fuente no está bien filtrada puede ser necesario añadir capacitores entre VDD y VSS para eliminar el ruido y el rizado.
<b>VSS</b>		Referencia (negativo del circuito).
<b>#RST</b>		Reset. Restablece el T6963C cuando se configura en nivel bajo.

Tabla 13.3 Pines del T6963C para conexión al microcontrolador

Pin Name	I / O	Functions																																																																																																																														
MDS MDO MD1	Input	<p>Pins for selection of LCD size</p> <table border="1"> <tr><td>DUAL</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td><td>L</td></tr> <tr><td>MDS</td><td>L</td><td>L</td><td>L</td><td>L</td><td>H</td><td>H</td><td>H</td><td>H</td><td>L</td><td>L</td><td>L</td><td>L</td><td>H</td><td>H</td><td>H</td><td>H</td><td>H</td></tr> <tr><td>MD1</td><td>H</td><td>H</td><td>L</td><td>L</td><td>H</td><td>H</td><td>L</td><td>L</td><td>H</td><td>H</td><td>L</td><td>L</td><td>H</td><td>H</td><td>L</td><td>L</td><td>L</td></tr> <tr><td>MDO</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>H</td><td>L</td><td>L</td></tr> <tr><td>LINES</td><td>2</td><td>4</td><td>6</td><td>8</td><td>10</td><td>12</td><td>14</td><td>16</td><td>4</td><td>8</td><td>12</td><td>16</td><td>20</td><td>24</td><td>28</td><td>32</td><td></td></tr> <tr><td>V-DOTS</td><td>18</td><td>32</td><td>48</td><td>64</td><td>80</td><td>96</td><td>112</td><td>128</td><td>32</td><td>64</td><td>96</td><td>128</td><td>160</td><td>192</td><td>224</td><td>256</td><td></td></tr> <tr><td></td><td colspan="8">1 SCREEN</td><td colspan="9">2 SCREENS</td></tr> </table>	DUAL	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L	L	MDS	L	L	L	L	H	H	H	H	L	L	L	L	H	H	H	H	H	MD1	H	H	L	L	H	H	L	L	H	H	L	L	H	H	L	L	L	MDO	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	L	LINES	2	4	6	8	10	12	14	16	4	8	12	16	20	24	28	32		V-DOTS	18	32	48	64	80	96	112	128	32	64	96	128	160	192	224	256			1 SCREEN								2 SCREENS								
DUAL	H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L	L																																																																																																															
MDS	L	L	L	L	H	H	H	H	L	L	L	L	H	H	H	H	H																																																																																																															
MD1	H	H	L	L	H	H	L	L	H	H	L	L	H	H	L	L	L																																																																																																															
MDO	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	L																																																																																																															
LINES	2	4	6	8	10	12	14	16	4	8	12	16	20	24	28	32																																																																																																																
V-DOTS	18	32	48	64	80	96	112	128	32	64	96	128	160	192	224	256																																																																																																																
	1 SCREEN								2 SCREENS																																																																																																																							
MD2 MD3	Input	<p>Pins for selection of number of columns</p> <table border="1"> <tr><td>MD2</td><td>H</td><td>L</td><td>H</td><td>L</td></tr> <tr><td>MD3</td><td>H</td><td>H</td><td>L</td><td>L</td></tr> <tr><td>Columns</td><td>32</td><td>40</td><td>64</td><td>80</td></tr> </table>	MD2	H	L	H	L	MD3	H	H	L	L	Columns	32	40	64	80																																																																																																															
MD2	H	L	H	L																																																																																																																												
MD3	H	H	L	L																																																																																																																												
Columns	32	40	64	80																																																																																																																												
FS0 FS1	Input	<p>Pins for selection of font</p> <table border="1"> <tr><td>FS0</td><td>H</td><td>L</td><td>H</td><td>L</td></tr> <tr><td>FS1</td><td>H</td><td>H</td><td>L</td><td>L</td></tr> <tr><td>Font</td><td>5 × 8</td><td>8 × 8</td><td>7 × 8</td><td>8 × 8</td></tr> </table>	FS0	H	L	H	L	FS1	H	H	L	L	Font	5 × 8	8 × 8	7 × 8	8 × 8																																																																																																															
FS0	H	L	H	L																																																																																																																												
FS1	H	H	L	L																																																																																																																												
Font	5 × 8	8 × 8	7 × 8	8 × 8																																																																																																																												
D0 to D7	I / O	Data I / O pins between CPU and T6963C (D7 is MSB)																																																																																																																														
<u>WR</u>	Input	Data Write. Write data into T6963C when <u>WR</u> = L.																																																																																																																														
<u>RD</u>	Input	Data Read. Read data from T6963C when <u>RD</u> = L.																																																																																																																														
<u>CE</u>	Input	Chip Enable for T6963C. <u>CE</u> must be L when CPU communicates with T6963C.																																																																																																																														
C / D	Input	<u>WR</u> = L ..... C / D = H: Command Write      C / D = L: Data Write <u>RD</u> = L ..... C / D = H: Status Read      C / D = L: Data Read																																																																																																																														
<u>HALT</u>	Input	H ..... Normal, L ..... Stops the oscillation of the clock																																																																																																																														
<u>RESET</u>	Input	H ..... Normal (T6963C has internal pull-up resistor) L ..... Initialize T6963C. Text and graphic have addresses and text and graphic area settings are retained.																																																																																																																														

Tabla 13.4 Función de los pines del T6963C

### 13.14.5 Configuración del T6963C

El T6963C tiene que ser configurado para que opere correctamente con la pantalla dada. Hay cuestiones de hardware y software que se deben considerar. Con respecto al hardware, por ejemplo , se debe seleccionar el tamaño de pantalla (resolución) o el tamaño de la fuente por medio de los pines correspondientes. Con respecto al software, enviar los comandos desde el microcontrolador, por ejemplo, para configurar el tamaño real de la pantalla.

#### Tamaño de fuente, número de columnas y pixeles por columna

La hoja de datos del T6963C indica que el número de columnas puede seleccionarse por medio de los pines MD2 y MD3. Se pudiera pensar que la fórmula “tamaño de fuente” multiplicado por la “configuración MD2/MD3” debería dar como resultado la resolución horizontal de la pantalla. Pero en la práctica, el tamaño de fuente no tiene que ver con la resolución horizontal. La tarea más importante para el tamaño de fuente es determinar la separación entre caracteres y lograr que la ubicación en memoria de los pixeles en pantalla sea clara y lógica. La resolución física real de la pantalla se determina por medio de comandos (software). El estado de los pines MD2 y MD3 debería seleccionarse de tal forma que el número de pixeles horizontales sea un múltiplo del número de columnas. Por ejemplo, para 240 pixeles en sentido horizontal se recomienda 40 columnas.

#### Consideraciones generales acerca de comandos y datos

Cuando se envíen comandos y datos hacia el T6963C se debe respetar un cierta secuencia. Antes de enviar comandos o datos se debe realizar una lectura del estado para determinar si el T6963C está listo para aceptar comandos o datos. Un comando siempre tiene un tamaño de un byte; algunos comandos requieren 2 bytes de datos, 1 byte de datos o ningún dato. Los datos tienen que ser enviados antes de los comandos.

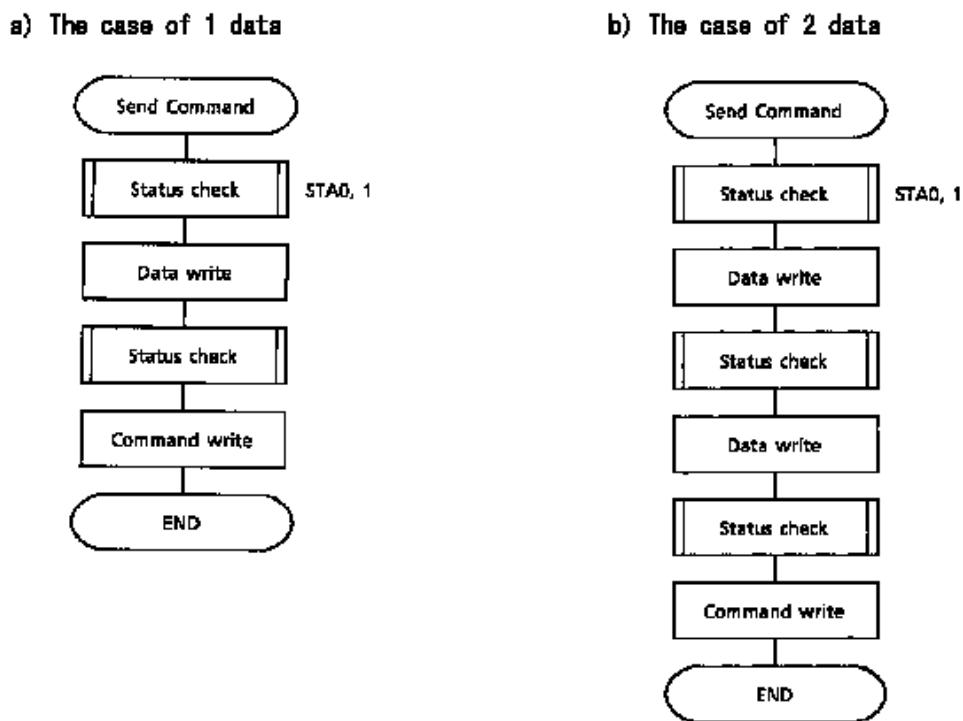


Figura 13.33 Secuencia para el envío de un comando (con uno o dos datos)

#### Comandos para configurar el tamaño de la pantalla

Como se mencionó antes, el tamaño real de la pantalla se configura por medio de comandos. Estos comandos se agrupan bajo el nombre de *Set Control Word*. Los comandos *Set Text Area* y *Set Graphic Area* son importantes para configurar el tamaño de la pantalla. Ambos necesitan un byte de datos. Sus valores indican cuantos bytes se emplean para una fila completa; es decir, qué tan lejos se encuentra un byte del byte

inmediatamente inferior de la fila siguiente. Los otros dos comandos son *Set Text Home Address* y *Set Graphic Home Address*. Se usan para configurar el inicio del área de texto y el área de gráficos en la VRAM.

### Memoria y tamaño de fuente

El tamaño de fuente determina cuántos bits (de un byte) se usan para mostrar texto y gráficos. Si la fuente tiene un ancho de 8 píxeles, cada bit de un byte en la VRAM controla un pixel de la pantalla. Si la fuente tiene un ancho de 6 píxeles, únicamente se usan 6 bits de cada byte para el control de los píxeles. Los dos primeros bits no importan. Eso significa, que independientemente de su tamaño, un carácter siempre es controlado por un byte en memoria.

### Comandos para configurar el modo

Con los comandos *Mode Set* el usuario puede configurar dos funciones. La primera es la manera en que el texto y los gráficos se combinan en la pantalla. La segunda permite habilitar o deshabilitar características tales como “Cursor ON/OFF”, “Texto ON/OFF”, etc. Estos comandos no requieren dato alguno.

**Modo OR:** En este modo se pueden mostrar texto y gráficos y los datos mostrados se combinan de acuerdo a la operación OR. Es la forma más común de combinar texto y gráficos.

**Modo EXOR:** En este modo los datos de texto y gráficos se combinan de acuerdo a la operación EXOR (OR exclusivo). Puede ser útil para mostrar texto en modo negativo (texto blanco sobre fondo negro).

**Modo AND:** En este modo los datos de texto y gráficos se combinan de acuerdo a la operación AND.

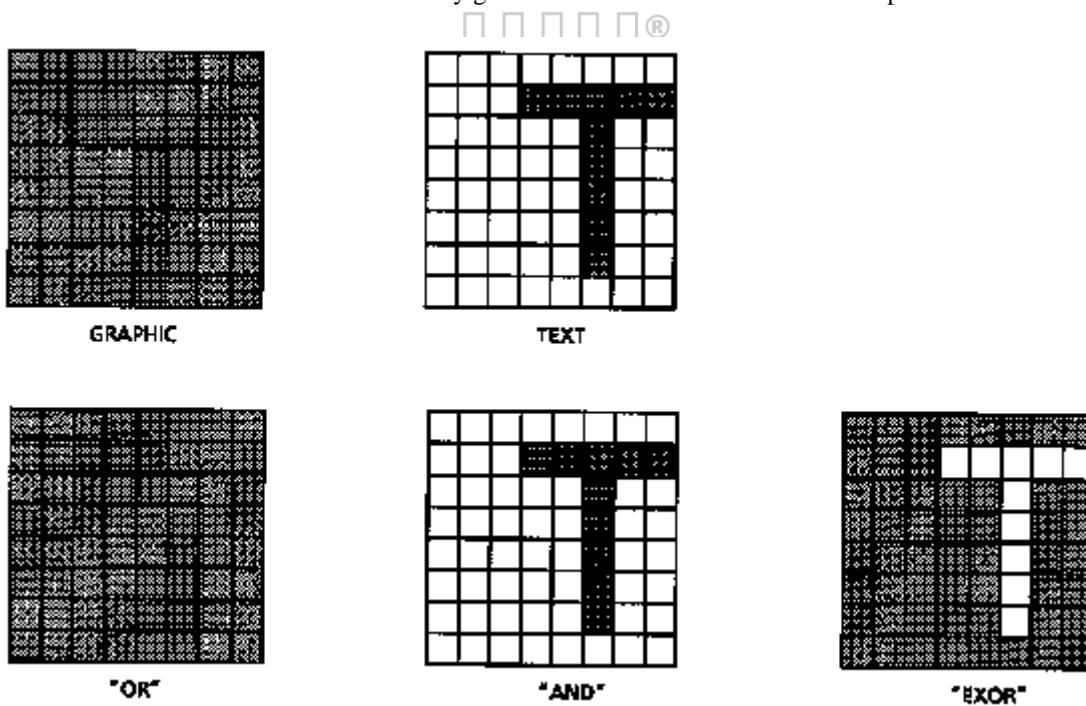


Figura 13.34 Modos de combinación de texto y gráficos

**Modo de Atributos de Texto:** Esta opción está disponible únicamente cuando se muestre texto exclusivamente. Los valores de atributos de texto se almacenan en el área para gráficos de la VRAM, que fue definida por el comando *Set Graphic Home Address*. Debido a que la memoria para gráficos también se usa, se tiene que habilitar la presentación de texto y gráficos con los comandos *Mode Set*. Las opciones de atributos de texto son “Invertir la pantalla” (muestra los caracteres en modo negativo), “Parpadeo de caracteres” e “Inhibir” (no mostrar este carácter). Cuando se selecciona un modo, se aplica a toda la pantalla. Por ejemplo, no es posible aplicar atributos al texto en una parte de la pantalla y mostrar gráficos en otra parte de la pantalla (la presentación de gráficos se deshabilita automáticamente).

Command	Code	D1	D2	Function
REGISTERS SETTING	00100001	X address	Y address	Set Cursor Pointer
	00100010	Data	00H	Set Offset Register
	00100100	Low address	High address	Set Address Pointer
SET CONTROL WORD	01000000	Low address	High address	Set Text Home Address
	01000001	Columns	00H	Set Text Area
	01000010	Low address	High address	Set Graphic Home Address
	01000011	Columns	00H	Set Graphic Area
MODE SET	1000X000	—	—	OR mode
	1000X001	—	—	EXOR mode
	1000X011	—	—	AND mode
	1000X100	—	—	Text Attribute mode
	10000XXX	—	—	Internal CG ROM mode
	10001XXX	—	—	External CG RAM mode
DISPLAY MODE	10010000	—	—	Display off
	1001XX10	—	—	Cursor on, blink off
	1001XX11	—	—	Cursor on, blink on
	100101XX	—	—	Text on, graphic off
	100110XX	—	—	Text off, graphic on
	100111XX	—	—	Text on, graphic on
CURSOR PATTERN SELECT	10100000	—	—	1-line cursor
	10100001	—	—	2-line cursor
	10100010	—	—	3-line cursor
	10100011	—	—	4-line cursor
	10100100	—	—	5-line cursor
	10100101	—	—	6-line cursor
	10100110	—	—	7-line cursor
	10100111	—	—	8-line cursor

Tabla 13.5 Comandos de control del T6963C

## 13.15 LM4228: LCD GRÁFICO 128x64 (GLCD 128x64)

El LM4228 es un módulo LCD de matriz de puntos (128x64), con tecnología CMOS y controlador Toshiba T6963C. Armado en tablero de circuito impreso, con marco de soporte de metal y alumbrado de fondo por LEDs. Disponible en versiones STN y STN-H (rango de temperatura extendido). Otras opciones incluyen la generación incorporada de voltaje negativo y el circuito de compensación de temperatura.

### 13.15.1 Precauciones

[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

#### Precauciones de manipulación

- Este dispositivo es susceptible a los daños por descarga electrostática (ESD). Se deben tener en cuenta los cuidados contra la electricidad estática.

#### Precauciones de la fuente de alimentación

- Tomar siempre en cuenta los valores máximos permitidos.
- Evitar la aplicación de voltaje con polaridad invertida en los pines VDD y VSS, aunque sea por un tiempo muy breve.
- Usar una fuente bien regulada y libre de transitorios.
- Evitar las señales en el bus de datos mientras el módulo está apagado.
- No colocar un capacitor entre Vo (contraste) y el pin de referencia VSS. VDD tiene que ser siempre mayor que Vo (en el instante de apagado el voltaje almacenado en el capacitor puede hacer que Vo sea mayor que VDD, lo cual puede dañar el módulo).

#### Precauciones de operación

- No conectar o desconectar el módulo cuando el sistema está alimentado.
- Minimizar la longitud de cable entre el módulo y el microcontrolador (máximo 30 cm).
- Para modelos con luz de fondo EL o CCFL, no deshabilitar la luz de fondo interrumpiendo la línea de alto voltaje. Los inversores sin carga pueden generar muy altos voltajes que pueden producir arcos eléctricos nocivos.
- El módulo debe trabajar dentro de los límites de temperatura especificados por el fabricante.

### Precauciones ambientales y mecánicas

- Uno de los mayores motivos de dificultades es un proceso inapropiado de soldadura. No se recomienda el uso de pastas o cremas limpiadoras ya que pueden deslizarse en el interior del módulo y producir fallas.
- El módulo instalado debe estar libre de esfuerzos mecánicos.
- La superficie del módulo LCD no se debe tocar ni rayar. La superficie frontal es un polarizador plástico muy fácil de rayar. Limpiar la pantalla solo cuando sea necesario con un algodón humedecido en un líquido apropiado para la limpieza de LCDs.
- Siempre se deben tener en cuenta los procedimientos anti estática cuando se manipule el módulo.
- Evitar el ingreso de humedad al módulo y tener en cuenta los límites de temperatura de operación y almacenamiento.
- No exponerlo a la luz del sol de forma directa.
- Si se produce una fuga del cristal líquido, evitar el contacto y la ingestión. Si se produce el contacto, lavarlo con abundante agua y jabón.

### 13.15.2 Valores máximos absolutos

Los valores mostrados aquí deberían explicarse por sí solos, no obstante es necesario recalcar que el máximo voltaje de alimentación es de 7V (lo normal es 5V). Nótese que los voltajes VDD-VSS y VDD-VEE no deberían ser negativos en ningún momento, ya que, como se dijo anteriormente esto puede dañar seriamente al dispositivo.

Item	Symbol	NTN				Unit
		Min.	Max.	Min.	Max.	
Logic supply voltage	VDD-VSS	0	7	0	7	V
LC driver supply voltage	VDD-VEE	0	24	0	24	V
Operating temperature	T <sub>OP</sub>	0	+50	-20	+70 (Note 3)	°C
Storage temperature (Note 1)	T <sub>ST</sub>	-20	+70	-30	+80	
Humidity: Operating (@40°C) Non-operating (@40°C)	- -	- -	85% 95%	- -	85% 95%	RH (Note 2) RH (Note 2)

Notes: 1: Tested to 100 hrs.

2: Refers to non-condensing conditions.

3. With backlight off.

[www.programarpicenc.com](http://www.programarpicenc.com)

Tabla 13.6 Valores máximos para el módulo LM4228

### 13.15.3 Características eléctricas

El LM4228 consume una corriente de 10 mA para su funcionamiento (aparte de la corriente de 480 mA para el alumbrado de fondo). Si se emplea una fuente negativa externa VEE de 18V, el GLCD consumirá una corriente de 6 mA de esta fuente.

Item	Symbol	Test Condition	V <sub>DD</sub> =5±0.25V; T <sub>a</sub> =25°C			
			Min.	Typ.	Max.	Unit
Input "High" voltage	V <sub>IH</sub>	-	0.8	-	V <sub>DD</sub>	V
Input "Low" voltage	V <sub>IL</sub>	-	V <sub>SS</sub>	-	0.2V <sub>DD</sub>	V
Output "High" voltage	V <sub>OH</sub>	I <sub>OH</sub> =0.205mA	2.2	-	-	V
Output "Low" voltage	V <sub>OL</sub>	I <sub>OL</sub> =1.2mA	-	-	0.8	V
Power supply current	I <sub>EE</sub>	V <sub>EE</sub> =-18V	-	6	-	mA
Power supply current	I <sub>DD</sub>	V <sub>DD</sub> =5.0V	-	10	-	mA

Tabla 13.7 Características eléctricas del LM4228

### 13.15.4 Especificaciones de la luz de fondo

El LM4228 no incorpora el resistor de polarización (R1) de los LEDs, por lo que debe ser añadido externamente (R2), como se muestra en la figura 13.35. Es importante notar que el consumo de los LEDs de

fondo es relativamente alto (480 mA con 5V), este hecho debe tenerse en cuenta al diseñar la fuente de alimentación.

$T_a=20^\circ\text{C}, 60\% \text{RH, Darkroom.}$				
Item	Symbol	Typ.	Max.	Unit
LED input voltage	V <sub>LED</sub>	5	6	V
LED input current	I <sub>LED</sub>	480	550	mA
Built-in current limit resistor	R <sub>1</sub>	-	-	Ohms, W
Recommended external current limit resistor	R <sub>2</sub>	1.6 Ohm, 1W	-	Ohms, W
Number of LED nodes	N	48	-	-

Tabla 13.8 Especificaciones del alumbrado de fondo

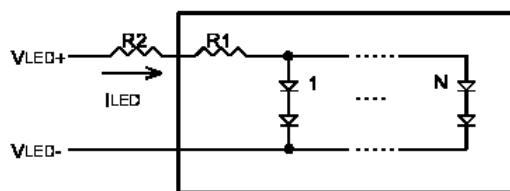


Figura 13.35 Polarización de los LEDs de fondo por medio de un resistor externo

### 13.15.5 Fuente de alimentación

La figura 13.36 muestra las posibles formas de conectar la fuente de alimentación al módulo, dependiendo de las características particulares de éste. En la mayoría de los casos el GLCD ya incorpora el generador de voltaje negativo  $V_{EE}$ , pero no trae el circuito de compensación de temperatura, así que se debe tomar como referencia la figura de la parte superior derecha. VR es un potenciómetro de  $10\text{k}\Omega$  que se emplea para ajustar el contraste.

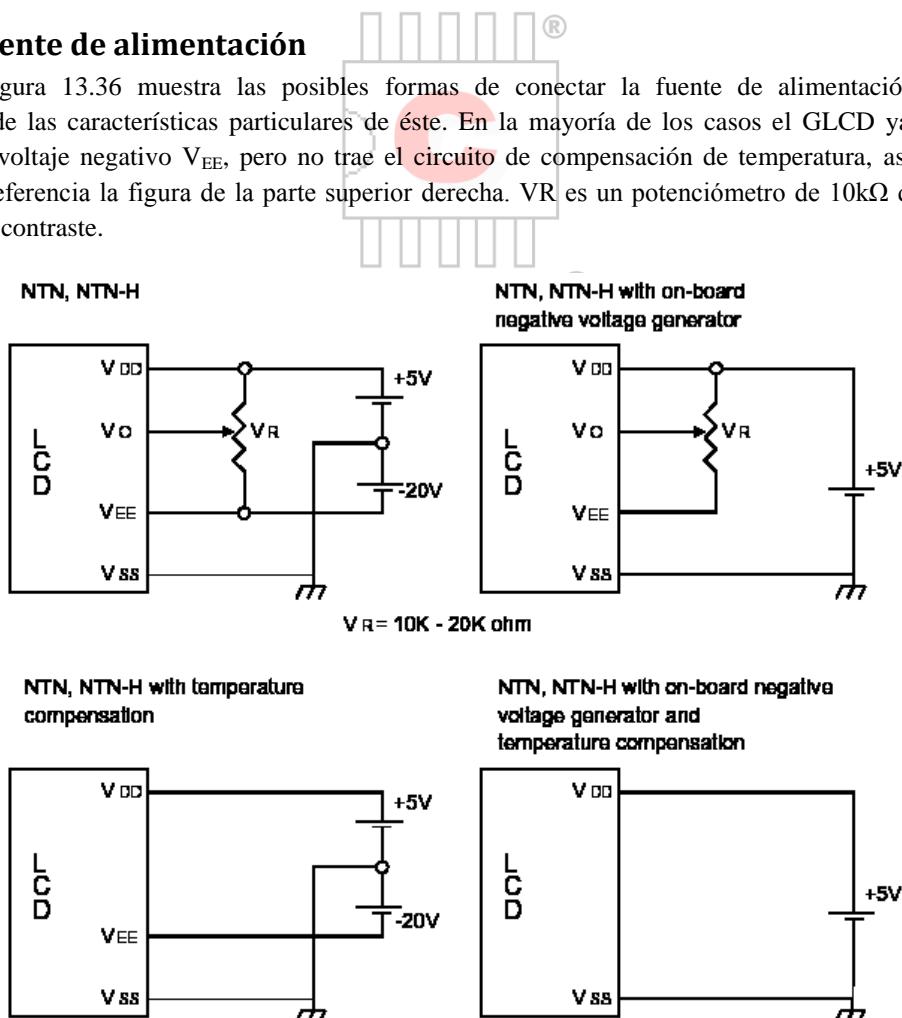


Figura 13.36 Opciones para la conexión de la fuente de alimentación

### 13.15.6 Descripción de los pines

Pin No.	Symbol	I/O	Function
1	V <sub>SS</sub>	-	Ground (0V)
2	V <sub>DD</sub>	-	Logic Supply Voltage (+5V)
3	V <sub>O</sub>	-	LC drive voltage for contrast adjustment
4	C/D	I	WR="L"...C/D="H": Command write RD="L"...C/D="H": Status read C/D="L": Data write C/D="L": Data read
5	RD	I	Data read Active Low
6	WR	I	Data write Active Low
7	DB0	I/O	Bi-directional data bus line 0
8	DB1	I/O	Bi-directional data bus line 1
9	DB2	I/O	Bi-directional data bus line 2
10	DB3	I/O	Bi-directional data bus line 3
11	DB4	I/O	Bi-directional data bus line 4
12	DB5	I/O	Bi-directional data bus line 5
13	DB6	I/O	Bi-directional data bus line 6
14	DB7	I/O	Bi-directional data bus line 7
15	CE	I	Chip enable Active Low
16	RESET	I	Chip reset Active Low
17	V <sub>EE</sub>	I(O)	Negative voltage input for LC drive (Negative voltage output for models with on-board negative voltage generator)
18	MD2	I	Mode Selection (see below)
19	FS1	I	Terminals for selection of font size
20	HALT	-	Halt Function (H= Normal, L = Stop oscillation)
BL1	V <sub>LED+</sub>	-	Anode (+): LED backlight input voltage
BL2	V <sub>LED-</sub>	-	Cathode (-): LED backlight input voltage

Tabla 13.9 Función de los pines del LM4228

Tamaño de fuente	FS1	Número de columnas	MD2
6x8	1	32	1
8x8	0	40	0

### 13.15.7 Diagrama de bloques

El diagrama muestra que el LM4228 es una pantalla *Single Scan (1 Screen)*, pues los controladores de columnas (COL) no aparecen duplicados (una pantalla *Dual Scan (2 Screens)* mostraría en total cuatro controladores de columnas: dos en la parte inferior y dos en la parte superior del LCD PANEL).

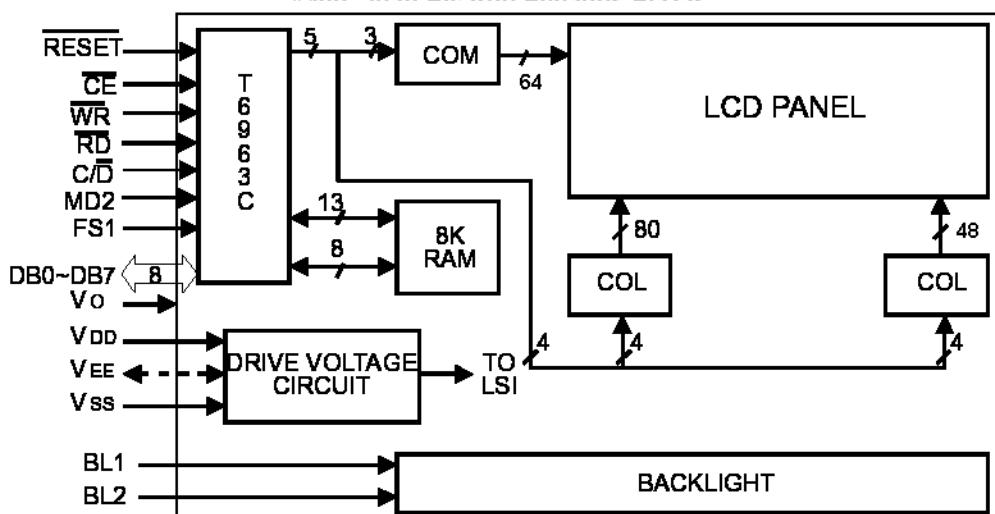


Figura 13.37 Diagrama de bloques del LCD LM4228

### 13.15.8 Dimensiones del módulo

En la figura 13.38 se puede ver que los pines del 1 al 20 se encuentran en la parte inferior del GLCD, distribuidos de acuerdo al detalle de la figura 13.39. Los pines para la alimentación de los LEDs de fondo se encuentran en el medio, a la izquierda (BL1 y BL2).

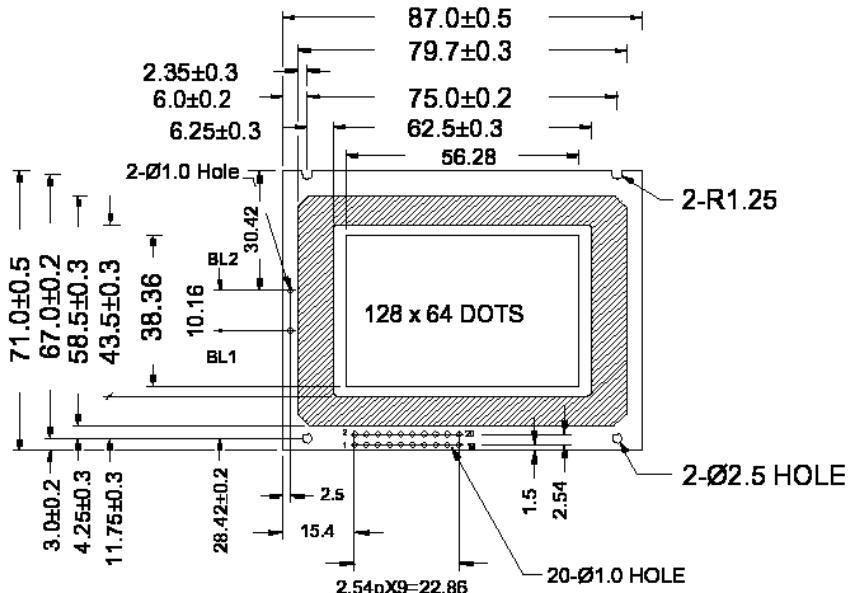


Figura 13.38 Apariencia física y dimensiones del módulo LM4228

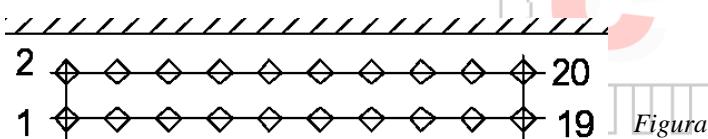


Figura 13.39 Detalle de la numeración de los pines

### 13.15.9 Descripción del número de parte para las opciones disponibles

Las opciones disponibles se especifican por medio de los caracteres identificados con las posiciones 1 a 5 (tablas 13.10 y 13.11).  
©Ing. Juan Ricardo Penagos Plazas

**LM4228①②64G128③④⑤**

Posición	Opciones disponibles
1	B E
2	G (Estándar)
3	D S H W
4	N C
5	B (Sólo para polarizador tipo E) G(Sólo para polarizador tipo B) Y(Sólo para polarizador tipo B)

Tabla 13.10 Opciones disponibles para el LM4228

Por ejemplo, el LM4228 EG64G128SNB tiene las siguientes características:

- **E** Polarizador transmisor. Fondo oscuro con alumbrado por LEDs.
- **G** LEDs de fondo de color amarillo-verde.
- **S** Rango estándar de temperatura. Generación de voltaje negativo incorporado.
- **N** Cristal líquido STN. Sin compensación de temperatura.
- **B** Fondo de color azul.

<b>LM/LE41290</b>	<b>B</b>	<b>G</b>	<b>128G240</b>	<b>S</b>	<b>N</b>	<b>G</b>
<b>MODEL NUMBER 4/5 digit</b>						
<b>POLARISER OPTIONS</b>						
A= Reflective: light background, no backlight						
B= Transflective: light background, with backlight						
E= Transmissive: dark background, with EL or CFL backlight and LED						
<b>LED BACKLIGHT COLOUR</b>						
G = Yellow-green						
A = Amber						
R = Red						
W = White						
<b>ROW X COLUMN DOT FORMAT</b>						
<b>TEMPERATURE RANGE AND POWER SUPPLY</b>						
D = Standard temperature range; negative supply voltage required						
S = Standard temperature range; on-board negative voltage generator						
H = Wide temperature range; negative supply voltage required						
W = Wide temperature range; on-board negative voltage generator						
<b>FLUID TYPE AND COMPENSATION CIRCUIT</b>						
F = FSTN (Film Supertwisted Nematic)						
N.S = STN (Supertwisted Nematic)						
C = STN with on-board temperature compensation circuit						
<b>BACKGROUND COLOUR</b>						
B = Blue (transmissive polarisers)						
G = Grey (reflective and transflective)						
Y = Yellow background (reflective and transflective)						

Tabla 13.11 Significado de las opciones disponibles para el LM4228

## 13.16 EJEMPLOS DE PROGRAMACIÓN

### Animación de imágenes de mapas de bits (bmp).

Para animar una imagen se debe tener en cuenta que el GH (*Graphic Home Address*) corresponde a la esquina superior izquierda de la pantalla. Por lo tanto, al incrementar el GH en pasos iguales a GA (*Graphic Area*) se logra que la imagen se desplace hacia arriba de línea en línea, logrando así la animación. Para desplazar la imagen completamente se debe hacer que el GH vaya tomando los valores de la primera columna (desde 000 hasta 3F0).

El LCD gráfico 128x64 (128 columnas y 64 filas) tiene un total de 8.192 puntos. Cada punto está asociado a un bit en la VRAM, de tal manera que se necesitan 8.192 bits (1.024 bytes) para almacenar una pantalla completa. Cada fila requiere 16 bytes; en total hay 64 filas de 16 bytes cada una. Los 1.024 bytes ocupan un espacio de direcciones que va desde 000 hasta 3FF.

		COLUMNAS															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
FILAS	1	000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F
	2	010	011	012	013	014	015	016	017	018	019	01A	01B	01C	01D	01E	01F
	3	020	021	022	023	024	025	026	027	028	029	02A	02B	02C	02D	02E	02F
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	64	3F0	3F1	3F2	3F3	3F4	3F5	3F6	3F7	3F8	3F9	3FA	3FB	3FC	3FD	3FE	3FF

Tabla 13.12 Mapa de memoria del GLCD 128x64

Las imágenes se dibujan en una aplicación que permita la creación de gráficos monocromáticos (blanco y negro), por ejemplo Microsoft® Paint. Para esto se ingresa a *Imagen>Atributos>Blanco y negro (Paint)*.

>Properties). El tamaño de la imagen, que también se define dentro de los Atributos debe coincidir con el tamaño de la pantalla, es decir 128x64 pixeles (figura 13.40). La imagen se debe guardar como Mapa de bits (\*.bmp). Esta imagen debe ser convertida a lenguaje C, para insertarlo en el código fuente de control del microcontrolador. Para la conversión se utiliza la herramienta Tools>GLCD Bitmap Editor de mikroC (figura 13.42). Luego de abrir esta herramienta se debe seleccionar el controlador del LCD (T6963C) y el tamaño del LCD (128x64); a continuación cargar el archivo bmp creado anteriormente (con el botón Load BMP Picture). Esto da como resultado un arreglo (array) de 1024 bytes. Seleccionar el código en lenguaje C (*mikroC code*) y copiarlo al portapapeles (*Copy CODE to Clipboard*) para pegarlo en el editor de mikroC.

**Nota:**◆En mikroC 3.2 no se ha implementado la opción para el tamaño de pantalla 128x64, así que para superar este pequeño problema lo que se debe hacer es crear imágenes de 128x128 pixeles en Paint, y hacer los dibujos únicamente en la mitad superior (128x64 pixeles) (Ver la figura 13.41). Realizar la conversión de la imagen 128x128 con la herramienta *GLCD Bitmap Editor*, copiar el código generado, pegarlo en el editor de mikroC y eliminar manualmente los últimos 1024 bytes del arreglo que se obtiene (el arreglo resultante de la conversión de una imagen 128x128 tiene un tamaño de 2048 bytes).

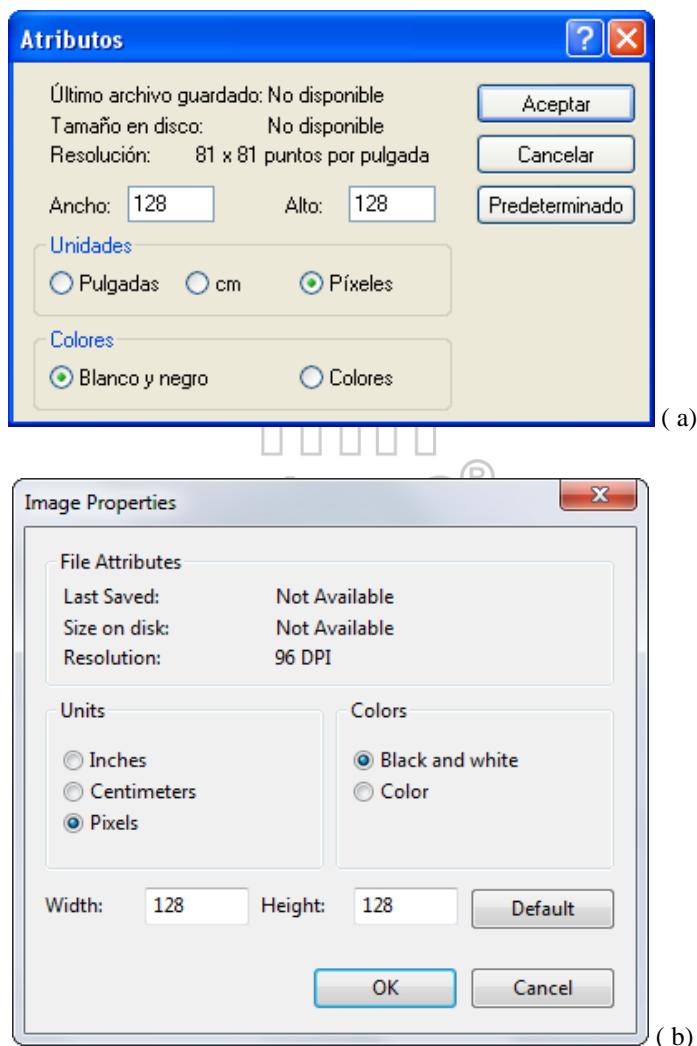


Figura 13.40 Ventana de atributos de Paint: a) Windows XP, b) Windows 7

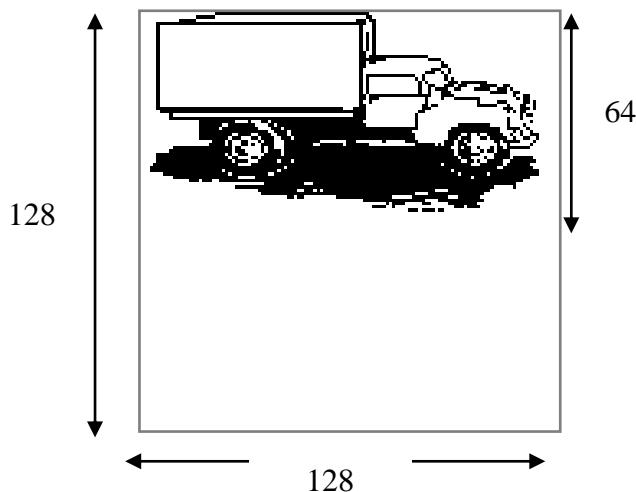


Figura 13.41 Dimensiones de la imagen (en pixeles)

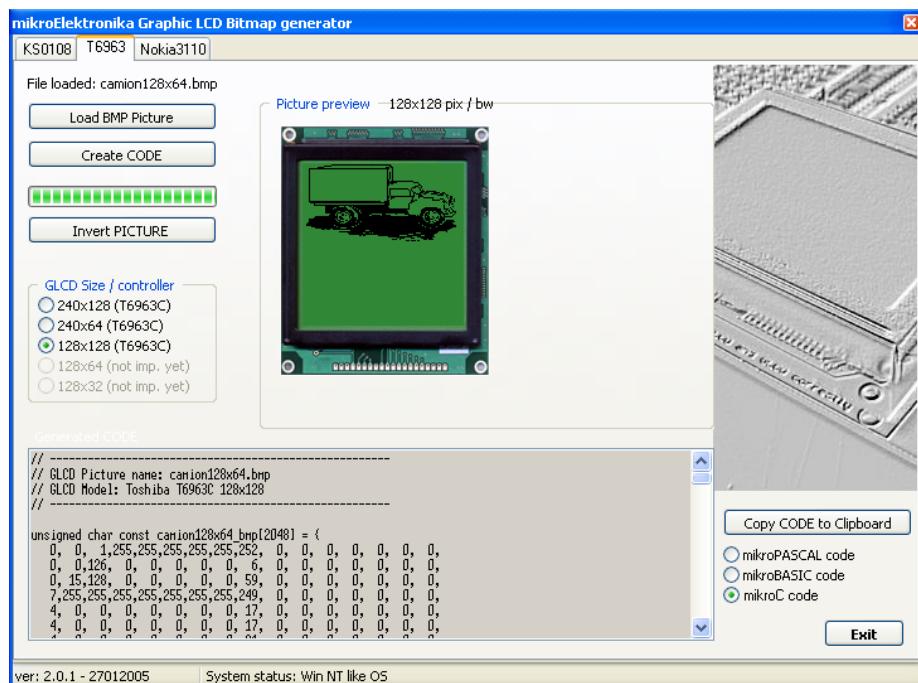


Figura 13.42 Ventana de la herramienta GLCD Bitmap Editor

El código generado y corregido se muestra abajo. Nótese que se ha modificado el número de bytes a 1024 y al final del último byte se ha eliminado la coma (,).

```

// -----
// GLCD Picture name: camion128x64.bmp
// GLCD Model: Toshiba T6963C 128x128
//
unsigned char const camion128x64_bmp[1024] = {
    0, 0, 1, 255, 255, 255, 255, 252, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 126, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0,
    ...
Dejar las primeras 64 filas y eliminar las últimas 64.
...
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

```

A continuación el planteamiento del primer problema de ejemplo y su solución.

**GLCD\_01.c:** Programa para animar la imagen de un camión en sentido vertical, y a continuación presentar un mensaje de texto. Se emplea una pantalla LM4228 y un PIC16F877A. *T6963C*. *Trigonometry*.

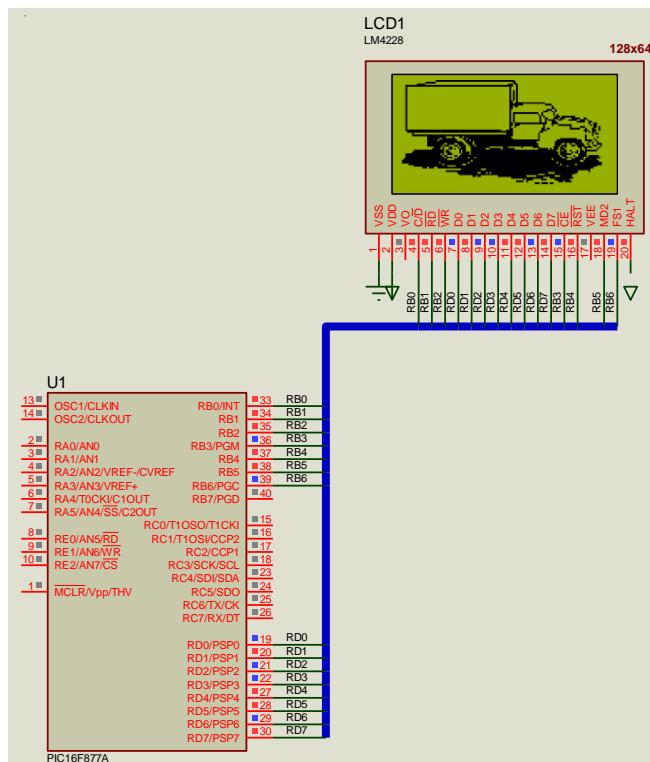


Figura 13.43 Circuito del problema GLCD 01.c

```

4, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 27, 255, 255, 0, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 31, 224, 0, 240, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 27, 0, 0, 8, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 24, 0, 0, 12, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 16, 0, 0, 114, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 16, 0, 1, 202, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 23, 255, 242, 4, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 20, 0, 10, 5, 0, 0, 0, 0,
4, 0, 0, 0, 0, 0, 0, 20, 0, 5, 3, 191, 192, 0, 0,
4, 0, 0, 0, 0, 0, 0, 20, 0, 4, 199, 112, 249, 128, 0,
4, 0, 0, 0, 0, 0, 0, 20, 0, 4, 188, 195, 231, 64, 0,
4, 0, 0, 0, 0, 0, 0, 20, 0, 4, 142, 0, 31, 48, 0,
4, 0, 0, 0, 0, 0, 0, 27, 255, 255, 143, 0, 0, 38, 0,
4, 0, 0, 0, 0, 0, 0, 27, 128, 1, 8, 255, 0, 22, 0,
4, 0, 0, 0, 0, 0, 0, 18, 0, 1, 0, 1, 7, 6, 128,
4, 0, 0, 0, 0, 0, 0, 24, 0, 3, 96, 1, 248, 140, 128,
7, 255, 255, 255, 255, 255, 255, 252, 120, 0, 3, 128, 0, 1, 206, 128,
3, 255, 255, 255, 255, 255, 255, 248, 0, 6, 0, 0, 0, 172, 0,
0, 64, 0, 0, 0, 0, 0, 120, 0, 8, 0, 0, 0, 11, 0,
0, 127, 255, 255, 255, 255, 255, 248, 0, 8, 0, 0, 0, 136, 0,
0, 0, 63, 248, 70, 255, 255, 255, 240, 0, 8, 0, 0, 0, 129, 0,
0, 0, 63, 159, 225, 191, 255, 255, 248, 0, 8, 0, 127, 224, 132, 0,
0, 0, 63, 127, 240, 239, 255, 255, 255, 248, 1, 255, 252, 242, 0,
0, 0, 63, 255, 254, 123, 255, 255, 255, 224, 3, 255, 252, 41, 0,
0, 0, 63, 243, 207, 59, 255, 255, 255, 240, 3, 201, 62, 73, 192,
0, 0, 63, 238, 55, 157, 255, 255, 255, 240, 3, 184, 223, 38, 64,
0, 0, 3, 209, 139, 143, 254, 1, 192, 32, 120, 7, 104, 13, 240, 128,
0, 0, 7, 209, 167, 143, 249, 253, 252, 0, 15, 215, 112, 4, 129, 0,
0, 63, 255, 246, 5, 143, 255, 255, 255, 255, 254, 89, 22, 254, 0,
0, 127, 255, 247, 5, 239, 255, 255, 255, 255, 255, 120, 183, 128, 0,
7, 255, 255, 211, 131, 239, 255, 255, 255, 255, 255, 127, 15, 128, 0,
7, 255, 255, 236, 55, 255, 255, 255, 255, 255, 255, 184, 79, 191, 0,
7, 255, 255, 247, 143, 221, 255, 255, 255, 255, 255, 205, 62, 127, 128,
15, 255, 255, 252, 63, 255, 255, 255, 255, 255, 247, 248, 254, 254, 0,
31, 255, 255, 255, 255, 247, 255, 255, 255, 255, 255, 255, 249, 255, 0,
31, 255, 255, 191, 255, 223, 255, 255, 255, 255, 255, 254, 255, 255, 192,
1, 255, 255, 239, 254, 127, 255, 255, 255, 255, 255, 191, 223, 255, 192,
0, 63, 255, 252, 31, 255, 255, 255, 255, 255, 251, 255, 254, 0,
0, 1, 220, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 248, 0,
0, 0, 0, 0, 3, 255, 199, 255, 255, 255, 255, 255, 255, 255, 224, 0,
0, 0, 0, 0, 7, 255, 255, 255, 255, 255, 255, 255, 255, 240, 32, 0,
0, 0, 0, 0, 15, 255, 58, 255, 255, 255, 255, 195, 255, 224, 0, 0,
0, 0, 0, 0, 7, 255, 255, 255, 255, 255, 240, 255, 255, 128, 0, 0,
0, 0, 0, 0, 0, 6, 192, 15, 255, 248, 31, 252, 127, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 15, 255, 254, 31, 255, 128, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 131, 131, 254, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 228, 0, 0, 239, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 216, 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
};

void main() {
    T6963C_ctrlce_Direction = 0;
    T6963C_ctrlce = 0; //Habilitar el T6963C.
    T6963C_ctrlfs_Direction = 0; //Fuente 8x8 (FS<1:0>=00).
    T6963C_ctrlmd_Direction = 0;
    T6963C_ctrlmd = 1; //32 columnas (MD<3:2>=11) (pantalla virtual)

    T6963C_Init(128, 64, 8); //Inicializa el T6963C. Caracteres de 8 bits.
                                //16 columnas (pantalla real).
    T6963C_Graphics(1); //Habilita la presentación de gráficos.
    T6963C_Text(1); //Habilita la presentación de texto.

    while(1){
        T6963C_GrFill(0); //Borra la pantalla gráfica.
        T6963C_TxtFill(0); //Borra la pantalla de texto.

        Delay_ms(800);

        T6963C_Image(camion128x64_bmp); //Dibuja la imagen en el GLCD.
        Delay_ms(1500);
    }
}

```

```

//Animación. Se logra cambiando el valor de GH.
//GH corresponde a la esquina superior izquierda del GLCD.
for (j=0;j<=3;j++){
    for (i=0;i<=0x0F;i++){
        T6963C_WriteData(i*0x10);
        T6963C_WriteData(j);
        T6963C_WriteCommand(T6963C_GRAPHIC_HOME_ADDRESS_SET); //Configurar GH.
        Delay_ms(30);
    }
}
T6963C_GrFill(0); //Borrar la pantalla gráfica.

T6963C_WriteData(0);
T6963C_WriteData(0);
T6963C_WriteCommand(T6963C_GRAPHIC_HOME_ADDRESS_SET); //Restablecer GH.

T6963C_Write_Text("Animacion de",0,2,T6963C_ROM_MODE_OR);
T6963C_Write_Text("imagenes en",0,3,T6963C_ROM_MODE_OR);
T6963C_Write_Text("mikroC.",0,4,T6963C_ROM_MODE_OR);
Delay_ms(1000);
}
}

```

Una de las principales aplicaciones del GLCD es la presentación gráfica en tiempo real de magnitudes (mecánicas, eléctricas, térmicas y químicas). Éstas son detectadas por medio de sensores y transformadas por el convertidor A/D para su procesamiento interno en el microcontrolador. El siguiente es un ejemplo básico de un osciloscopio, que muestra un voltaje de entrada en función del tiempo.

◆Cuando se dibuja un pixel o punto en la pantalla se debe especificar su color:

- T6963C\_WHITE: Representa un pixel encendido, es decir que el pixel se mostrará en pantalla (se verá un punto negro).
- T6963C\_BLACK: Representa un pixel apagado, es decir que el pixel no se mostrará en pantalla (se verá el color de fondo).

Para mostrar una onda sinusoidal se debe tener en cuenta que el convertidor A/D sólo admite voltajes positivos de entrada, es por esta razón que se ha modificado el OFFSET del generador utilizado en la simulación, de tal manera que la onda desplazada se ubique totalmente sobre el eje x.

El procedimiento de visualización consiste en tomar un valor instantáneo de la señal de entrada (muestra de 10 bits), determinar el valor numérico del voltaje correspondiente y graficar un punto en la pantalla que representará ese valor. Realizando este procedimiento en repetidas ocasiones se obtiene un total de 128 puntos que se dibujan a lo largo de la pantalla. Luego se borra la pantalla y los pasos anteriores se repiten continuamente. Para establecer las divisiones en los ejes se procede experimentalmente, observando un voltaje de amplitud y período conocidos; por medio de prueba y error se hace la ubicación de las marcas correspondientes. La pantalla permite la presentación simultánea de texto y gráficos. Para ubicar el texto la pantalla se divide en 8 filas (F0:F7) y 16 columnas (C0:C15), comenzando por la esquina superior izquierda. Para los gráficos la pantalla se divide en 64 filas (F0:F63) y 128 columnas (C0:C127), empezando por la esquina superior izquierda.

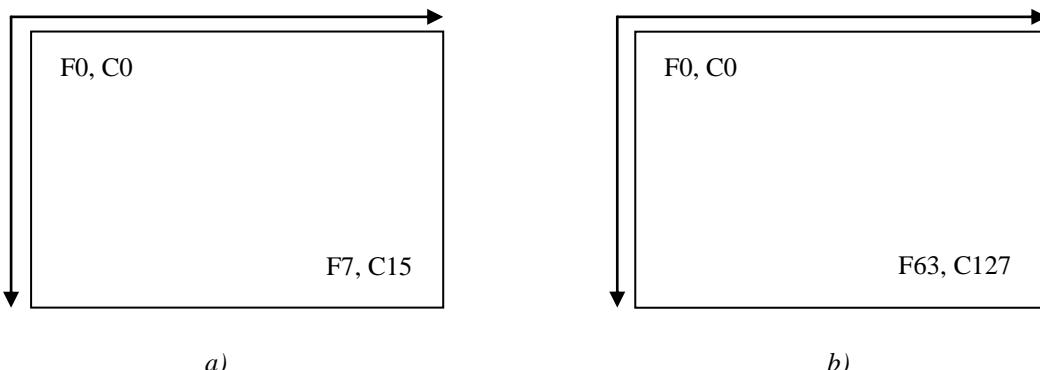
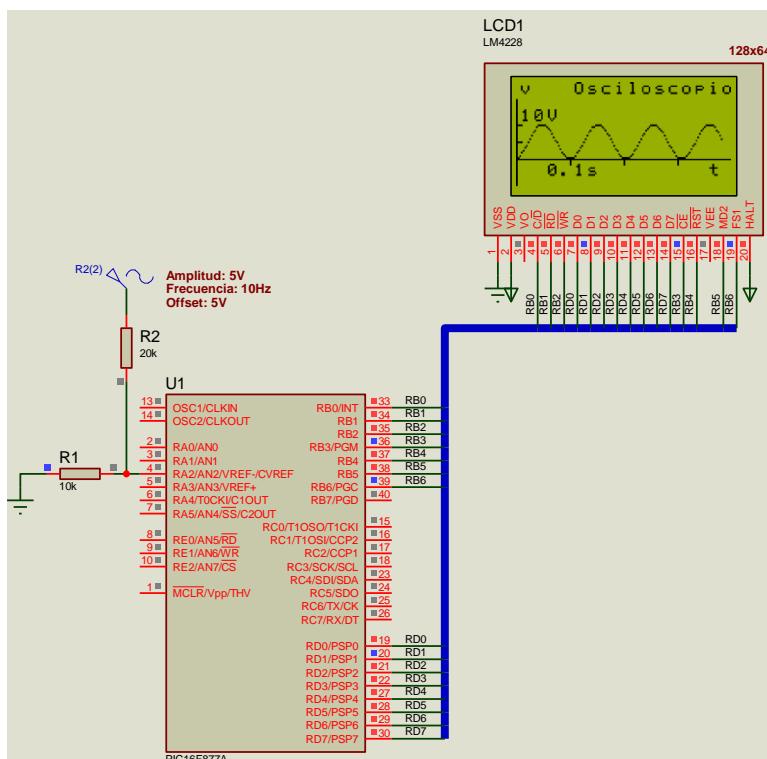


Figura 13.44 Sistemas de coordenadas para el GLCD 128x64: a) Texto, b) Gráficos

Lo que sigue es el planteamiento del problema y su solución.



**GLCD\_02.c:** Osciloscopio básico. Muestra la forma de onda del voltaje de entrada en función del tiempo (figura 13.45). El máximo voltaje de entrada es de 15V. Se emplea una pantalla LM4228 y un PIC16F877A. *T6963C*✓. *Trigonometry*✓. *ADC*✓.

Figura 13.45 Circuito del problema GLCD\_02.c

```

//GLCD_02.c
//Por defecto los pines del puerto A ya están configurados como
//entradas analógicas (para el convertidor A/D).
//Eje x: 32 pixeles corresponden a 100 ms (determinado experimentalmente).
//Eje y: 10 pixeles corresponden a 5 V (determinado experimentalmente).
//El archivo T6963C.h contiene la definición de constantes y macros. Este
//archivo se encuentra en la carpeta del proyecto.
#include "T6963C.h"

//Declaración de variables de conexión del T6963C
char T6963C_dataPort at PORTD; // Puerto de Datos
©Ing. Juan Ricardo Penagos Plazas

sbit T6963C_ctrlwr at RB2_bit; // Señal WR write
sbit T6963C_ctrlrd at RB1_bit; // Señal RD read
sbit T6963C_ctrlcd at RB0_bit; // Señal CD command/data
sbit T6963C_ctrlrst at RB4_bit; // Señal RST reset
sbit T6963C_ctrlwr_Direction at TRISB2_bit; // Señal WR write
sbit T6963C_ctrlrd_Direction at TRISB1_bit; // Señal RD read
sbit T6963C_ctrlcd_Direction at TRISB0_bit; // Señal CD command/data
sbit T6963C_ctrlrst_Direction at TRISB4_bit; // Señal RST reset

//Señales no empleadas por la librería, se configuran en la función main
sbit T6963C_ctrlce at RB3_bit; // Señal CE
sbit T6963C_ctrlfs at RB6_bit; // Señal FS
sbit T6963C_ctrlmd at RB5_bit; // Señal MD
sbit T6963C_ctrlce_Direction at TRISB3_bit; // Señal CE
sbit T6963C_ctrlfs_Direction at TRISB6_bit; // Señal FS
sbit T6963C_ctrlmd_Direction at TRISB5_bit; // Señal MD
//Final de declaración de variables de conexión del T6963C

int temp_res;
float voltaje;
char x;

void main() {
    T6963C_ctrlce_Direction = 0;
    T6963C_ctrlce = 0; //Habilitar el T6963C.
    T6963C_ctrlfs_Direction = 0;
    T6963C_ctrlfs = 0; //Fuente 8x8 (FS<1:0>=00).
    T6963C_ctrlmd_Direction = 0;
    T6963C_ctrlmd = 1; //32 columnas (MD<3:2>=11) (pantalla virtual)
}

```

```

T6963C_Init(128, 64, 8);           //Inicializa el T6963C. Caracteres de 8 bits.
T6963C_Graphics(1);               //Habilita la presentación de gráficos.
T6963C_Text(1);                  //Habilita la presentación de texto.
T6963C_Write_Text("v  Osciloscopio",0,0,T6963C_ROM_MODE_OR);
T6963C_Write_Text("10V",0,2,T6963C_ROM_MODE_OR);
T6963C_Write_Text("t",14,6,T6963C_ROM_MODE_OR);
T6963C_Write_Text("0.1s",2,6,T6963C_ROM_MODE_OR);

do{
    T6963C_Line(0,45,127,45,T6963C_WHITE); //Eje horizontal.
    T6963C_Line(0,10,0,60,T6963C_WHITE);   //Eje vertical.
    T6963C_Line(32,45,32,48,T6963C_WHITE); //División en el eje x (100ms).
    T6963C_Line(64,45,64,48,T6963C_WHITE); //División en el eje x (200ms).
    T6963C_Line(96,45,96,48,T6963C_WHITE); //División en el eje x (300ms).
    T6963C_Line(0,35,3,35,T6963C_WHITE);   //División en el eje y (5V).
    T6963C_Line(0,25,3,25,T6963C_WHITE);   //División en el eje y (10V).
    for (x=0;x<=127;x++){
        temp_res=ADC_Read(2);           //Obtiene 10 bits de la conversión del canal 2.
        voltaje=(float)(temp_res*0.00488); //Transforma el número de 10 bits al
                                         //voltaje correspondiente.
        voltaje*=3.0;                 //VT=3VIN.
        (int) voltaje;                //Convierte el voltaje en número entero.
        voltaje*=2;                  //Modifica V/DIV.
        T6963C_Dot(x,45-voltaje,T6963C_WHITE); //Dibuja el punto correspondiente.
    }
    T6963C_GrFill(0);               //Borra la pantalla gráfica.
} while(1);
}

```

El ejemplo mostrado debe tomarse únicamente como punto de partida y presenta los lineamientos generales que se utilizan en cualquier osciloscopio. Para obtener resultados más profesionales se debe emplear un GLCD de mayor resolución (por ejemplo un GLCD 240x128) que permite mejores presentaciones. Además se debe hacer el análisis cuidadoso para la selección del microcontrolador y el convertidor A/D, dependiendo del ancho de banda deseado para el osciloscopio.

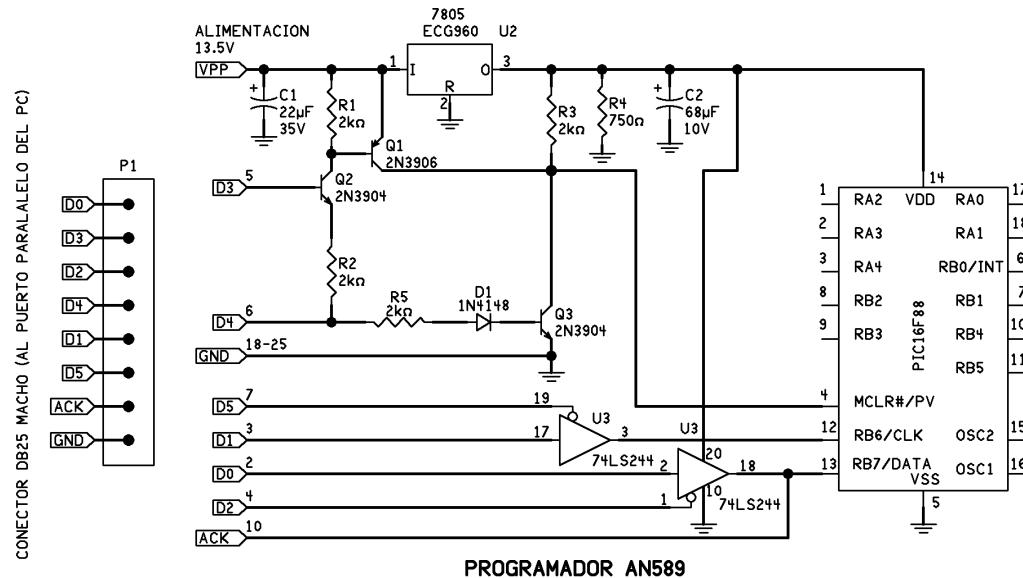


[www.programarpicenc.com](http://www.programarpicenc.com)  
©Ing. Juan Ricardo Penagos Plazas

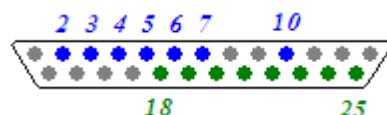
## APÉNDICE A: PROGRAMADOR AN589

Este programador para el puerto paralelo (puerto de impresora LPT1) es muy sencillo, fácil de construir y efectivo, y ha sido probado por el autor con los programas presentados en este libro con mucho éxito, en conjunto con el software IC-Prog en un ambiente WindowsXP Professional. Se aplica sin ninguna modificación a los PICs 16F88 y 16F628A debido a que son compatibles pin a pin. El PIC16F877A también se puede programar si se realizan las conexiones adecuadas de los cinco pines de programación de acuerdo al estándar ICSP.

### ESQUEMA ELÉCTRICO

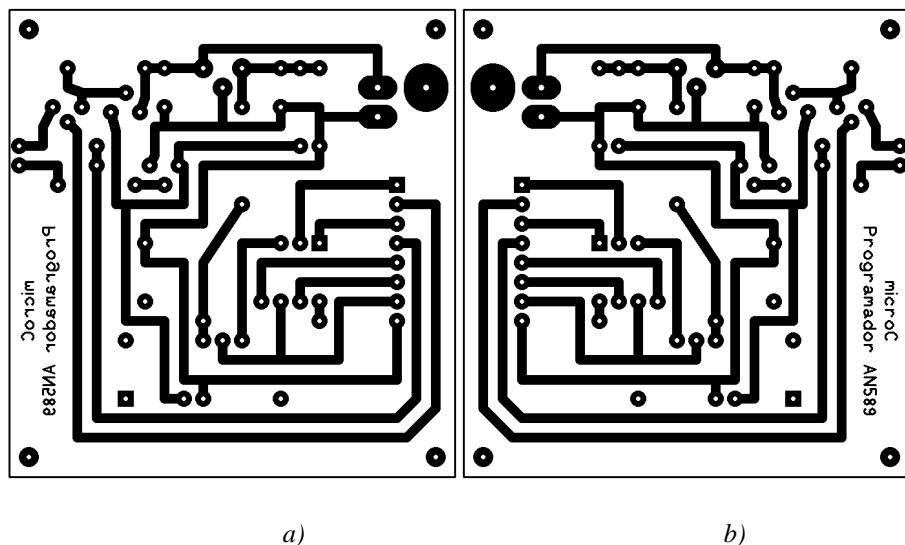


CONECTOR DB25 MACHO



### CIRCUITO IMPRESO PCB

Las dos siguientes figuras muestran el circuito impreso, visto desde la cara de los componentes y, además, visto desde la cara del cobre, de tal manera que el usuario tenga la posibilidad de seleccionar la imagen apropiada dependiendo de la técnica de manufactura empleada. El circuito impreso tiene un tamaño de 58x61 mm (base x altura).



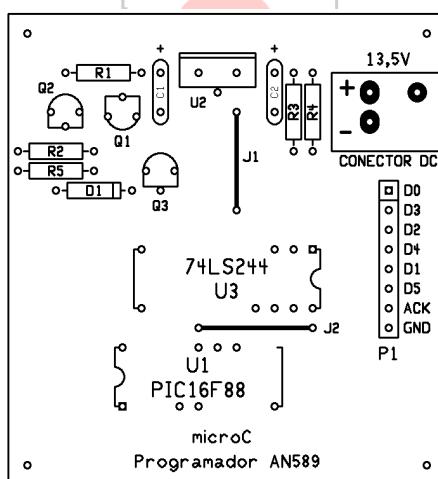
a)

b)

*Circuito impreso: a) Visto desde la cara de los componentes y b) Visto desde la cara del cobre*

## SIMBOLOGÍA (SILK)

Para facilitar el proceso de ubicación de componentes se muestra la imagen de la simbología. J1 y J2 representan dos puentes (*jumper*) de cobre. El conector DC se lo puede conseguir en cualquier tienda de Electrónica, mientras que los terminales de P1 permiten soldar un grupo de 8 cables de una longitud apropiada (puede ser de 1 metro cada uno) para el conector DB25 macho.



Se recomienda colocar zócalos DIP20 y DIP18 para los dos circuitos integrados, eliminando los pines que no se van a utilizar y dejando los demás para soldar de manera firme el zócalo al tablero de circuito impreso.

## APÉNDICE B: SOFTWARE IC-Prog

El IC-Prog es un programa que funciona bajo Windows para controlar un programador de microcontroladores PIC. Para operar este programa se necesitan conocimientos básicos de Windows y de electrónica. En la página [www.ic-prog.com](http://www.ic-prog.com) se puede encontrar mayor información así como los enlaces de descarga de la aplicación (icprog.exe) y del driver para Windows NT/2000/XP (icprog.sys).

Para que el programa funcione se deberá conectar a la computadora un programador, tal como el AN589, y configurar correctamente tanto a éste como al programa.

El IC-Prog requiere Windows 95, 98, ME, NT, 2000 o XP y un coprocesador interno o externo para funcionar. Todos los procesadores compatibles y superiores a un 386 con 8MB de memoria RAM deberían funcionar correctamente. El IC-Prog es un programa registrado aunque es de libre distribución.

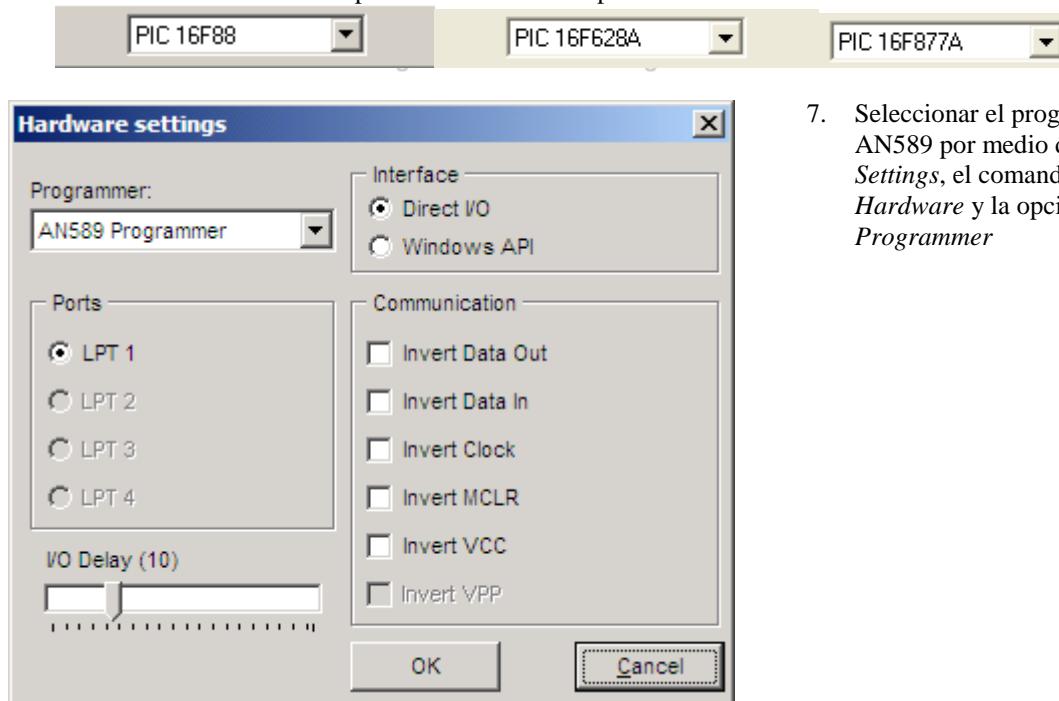
## INSTALACIÓN EN WINDOWS XP

1. Con el Explorador de Windows ir a la carpeta que contiene la aplicación icprog.exe
2. Hacer clic derecho e ir a *Propiedades*.
3. Ir al menú de *Compatibilidad*.
4. Verificar que la compatibilidad esté DESHABILITADA.
5. Clic en *Aplicar* y luego en *Aceptar*.

Se debe copiar el archivo icprog.sys en la MISMA CARPETA de icprog.exe. Luego iniciar la aplicación icprog.exe, ir a *Settings*, *Options* y seleccionar *Misc*. Marcar la opción “*Enable NT/2000/XP Driver*”, con lo cual éste se instalará.

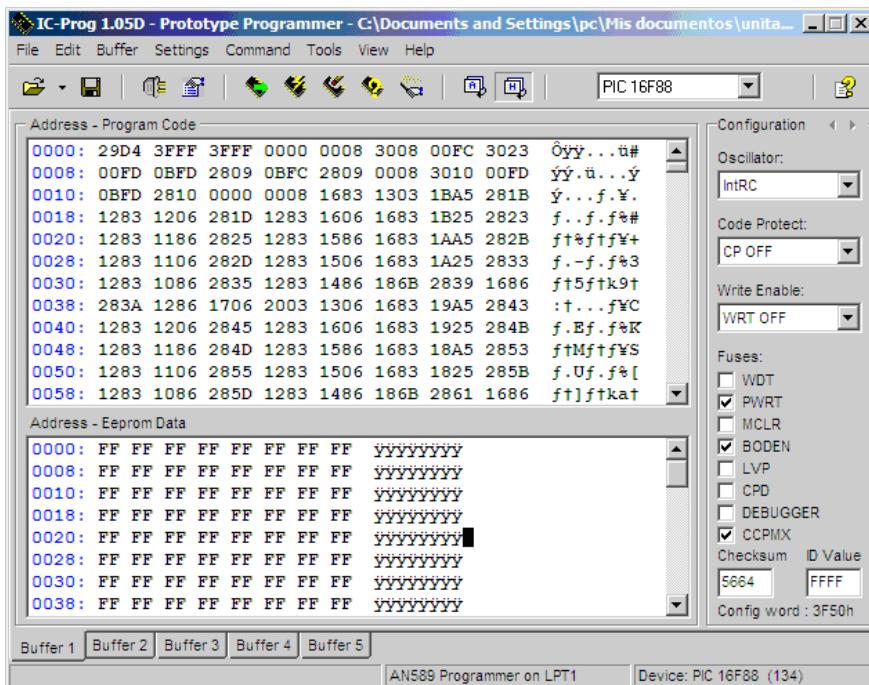
## PROGRAMACIÓN DE LOS PICS 16F88, 16F628A Y 16F877A

1. Conectar el programador AN589 al conector del puerto de impresora (LPT1).
2. Encender la computadora y esperar a que finalice el procedimiento de encendido.
3. Conectar el PIC (16F88, 16F628A o 16F877A) al programador AN589.
4. Conectar la fuente de alimentación de 13,5 V<sub>CD</sub> al programador.
5. Iniciar el icprog.exe
6. Seleccionar el PICen el campo de selección del dispositivo



7. Seleccionar el programador AN589 por medio del menú *Settings*, el comando *Hardware* y la opción *Programmer*

8. Abrir el archivo ejecutable \*.hex (que fue creado en mikroC™ luego de la compilación del código fuente) del programa que se desea grabar en el PIC por medio del menú *File* y el comando *Open File*.



9. Realizar la grabación seleccionando el comando

*Program All* del menú *Command* o el ícono . Luego de unos pocos segundos el programa será transferido al PIC y se realizará una verificación automática del código grabado. Si todo está correcto se indicará por medio de un mensaje.

10. Apagar la fuente de alimentación del programador, retirar el PIC y colocarlo en el circuito de aplicación final.



◆ Los bits de configuración NO deberían ser modificados con el IC-Prog pues éstos ya fueron establecidos dentro de mikroC, y al abrir el archivo \*.hex esta configuración se copia automáticamente.

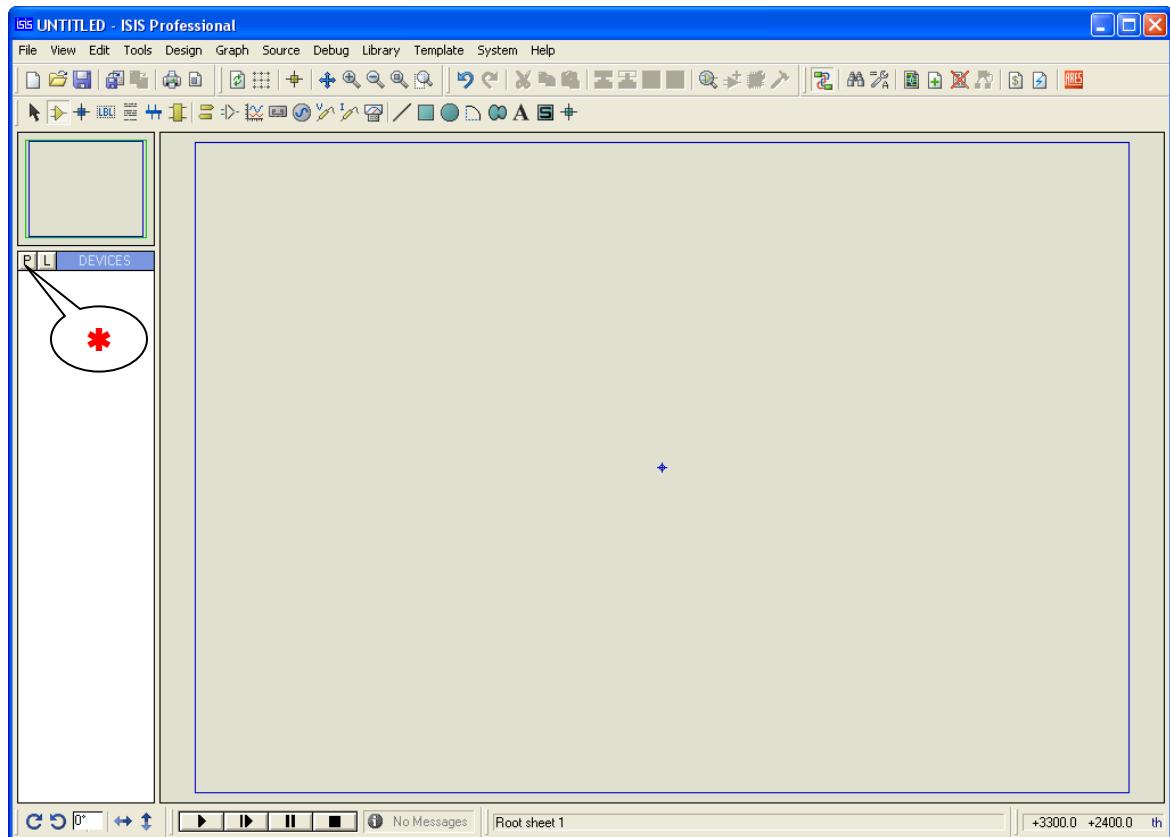
## APÉNDICE C: SIMULACIÓN BÁSICA EN ISIS® DE PROTEUS®

ISIS es uno de los componentes del sistema de diseño PROTEUS, desarrollado por la empresa Labcenter Electronics. El programa ISIS permite la elaboración de esquemas electrónicos empleando una amplia variedad de dispositivos de todos los fabricantes de renombre a nivel mundial. Estos circuitos electrónicos pueden ser simulados a través del simulador incorporado PROTEUS VSM. Además es posible diseñar circuitos impresos por medio del programa ARES, otro de los elementos integrantes de PROTEUS.

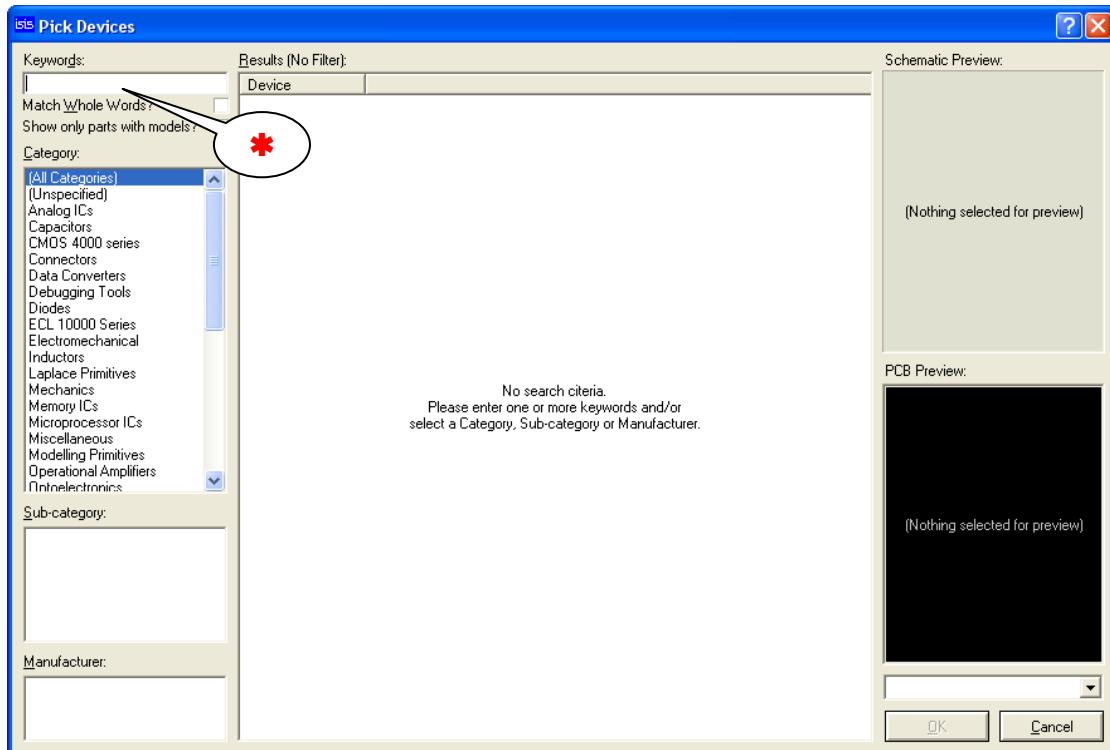
El propósito de este apartado es describir el proceso básico de construcción y simulación de circuitos con microcontroladores y otros componentes electrónicos, a partir del ejemplo EncenderLED.c empleando el PIC16F628A.

## CONSTRUCCIÓN DEL ESQUEMA ELÉCTRICO

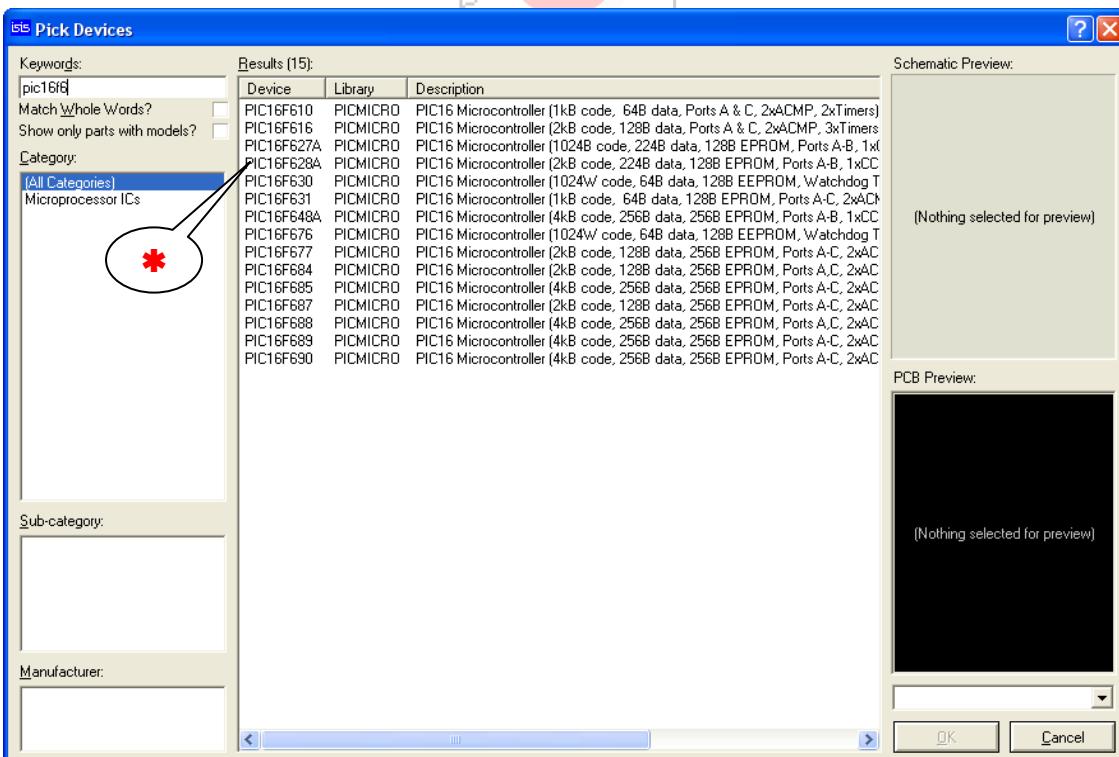
Una vez que ha iniciado el programa ISIS deberá tener una pantalla como la siguiente:



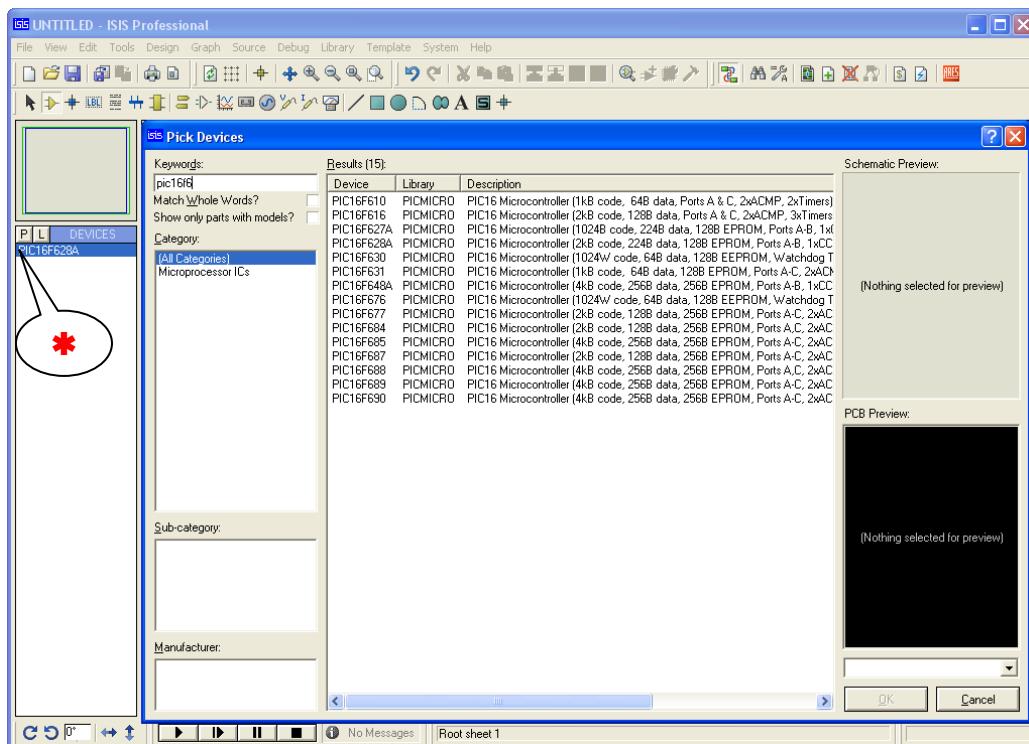
Al hacer clic en la letra “P” usted tendrá la oportunidad de seleccionar y reunir los elementos electrónicos con los cuales construirá su circuito:



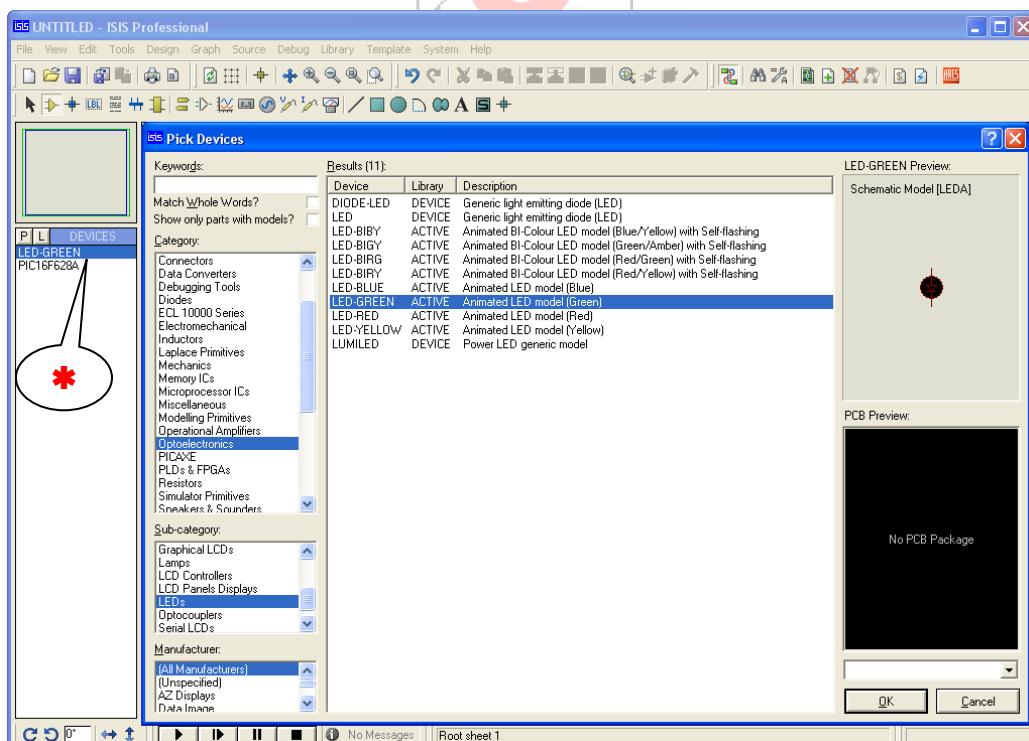
En el campo *Keywords* puede empezar a teclear las primeras letras de uno de los componentes, por ejemplo PIC16F6, y automáticamente aparecerá una lista de los resultados relacionados; mientras más letras usted escriba, la lista se irá haciendo más corta y la búsqueda se irá enfocando en unos pocos elementos:



Ahora haga doble clic en el dispositivo PIC16F628A para que aparezca en el campo del selector de dispositivos:

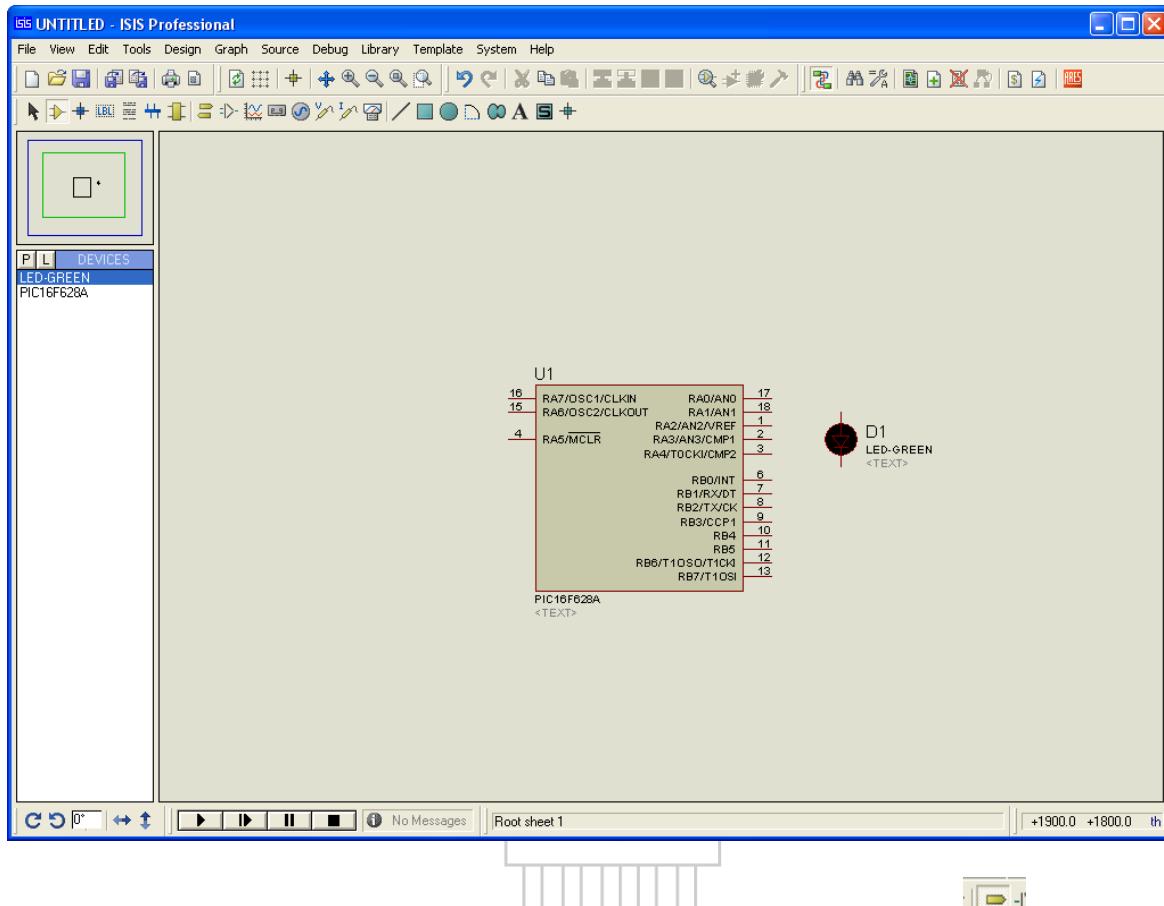


Ahora se va a seleccionar un LED, para lo cual se deja en blanco el campo *Keywords*. En *Category* hacer clic en *Optoelectronics* y en *Sub-category* hacer clic en *LEDs*. En la lista de resultados hacer doble clic en *LED-GREEN* para añadirlo al selector de dispositivos:

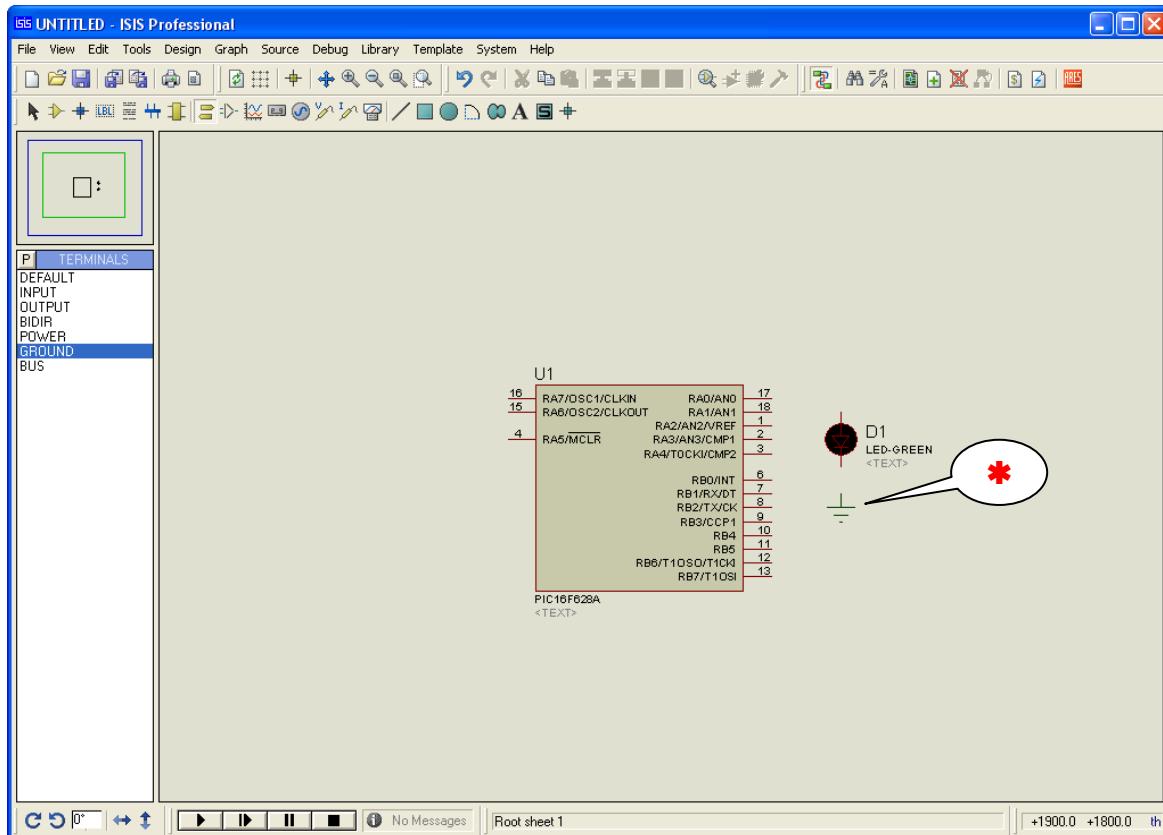


Una vez que se han seleccionado todos los elementos se hace clic en *OK*. Se procede ahora a la construcción del esquema eléctrico. Hacer clic en el elemento PIC16F628A del selector de dispositivos, mover el cursor hacia la ventana de edición (el cursor debe adoptar la figura de un lápiz ) y hacer clic (ahora se ve la silueta del PIC16F628A en color lila), ubicarla en cualquier parte de la ventana de edición y hacer clic una vez más (ahora aparece el símbolo completo del PIC16F628A). Hacer clic en el elemento LED-GREEN del selector

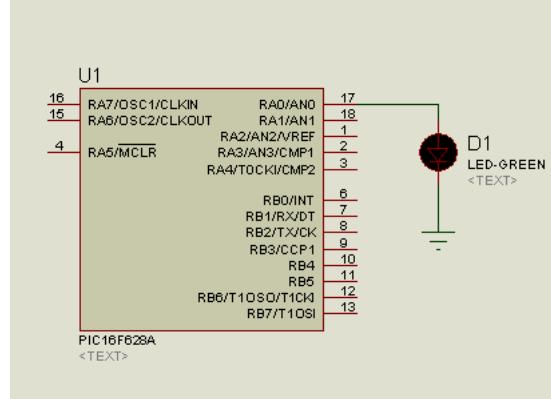
de dispositivos y seguir el mismo procedimiento para ubicarlo en la ventana de edición a una altura adecuada para conectarlo al pin RA0:



Para colocar una referencia (GND) se hace clic en el ícono *Terminals Mode* , hacer clic en el elemento GROUND y seguir el procedimiento descrito previamente:



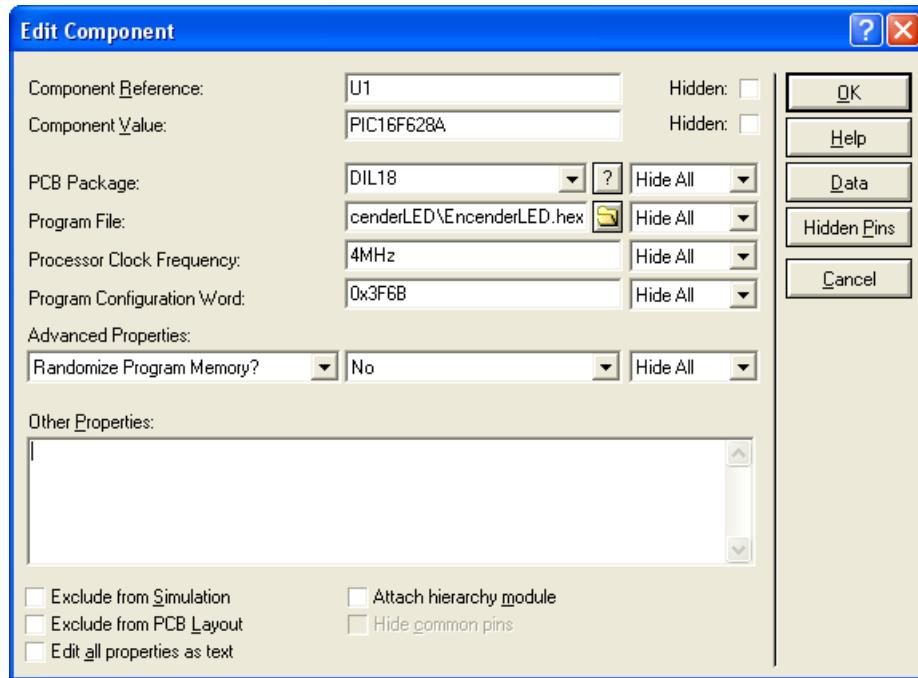
Para realizar las conexiones se ubica el cursor en el pin RA0, de modo que aparezca un pequeño cuadro rojo y se hace clic; mover el cursor hasta el ánodo del LED para que aparezca un cuadro rojo y hacer clic nuevamente. Repetir el procedimiento para conectar el cátodo a la referencia (GND). Para borrar una conexión se debe hacer doble clic derecho sobre ella. La polarización del microcontrolador se encuentra conectada inicialmente por defecto, por esta razón no aparecen los pines V<sub>SS</sub> y V<sub>DD</sub> en el símbolo:



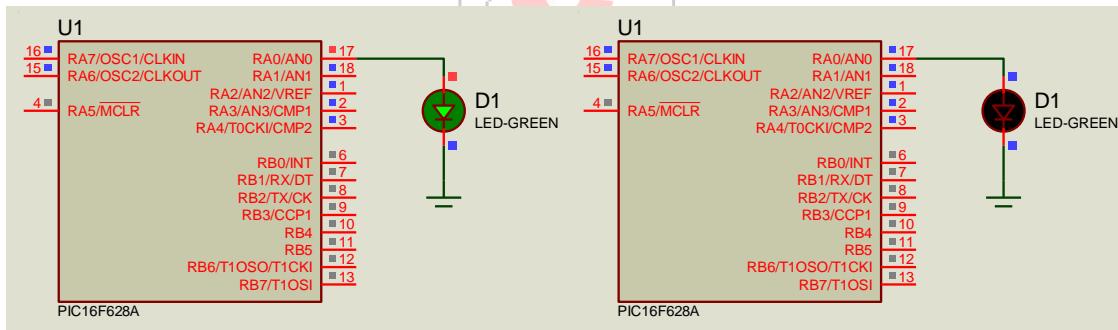
## SIMULACIÓN

Lo siguiente es cargar el código ejecutable (EncenderLED.hex) en el microcontrolador y configurar la frecuencia de operación. Hacer doble clic sobre el símbolo del PIC, esto abre la ventana *Edit Component*. En esta

ventana se puede buscar el código ejecutable haciendo clic en la carpeta del campo *Program File*. La frecuencia de operación (4MHz) se ingresa en el campo *Processor Clock Frequency*. Hacer clic en *OK*:



Para iniciar la simulación hacer clic en el botón *Play* , luego de unos segundos se podrá observar cómo el LED empieza a parpadear. Para detener la simulación hacer clic en el botón *Stop* . Si desea puede guardar este esquema con el comando *File ->Save Design As:*



Lo que se ha explicado constituye una introducción a la simulación en ISIS. Es posible emplear instrumentos tales como el osciloscopio, el voltímetro, el amperímetro y otros más; y una gran variedad de elementos animados: pulsadores, interruptores, motores, LEDs, etc. que usted irá conociendo a medida que profundice en el estudio de la programación de microcontroladores.

## SOBRECARGA DEL ORDENADOR

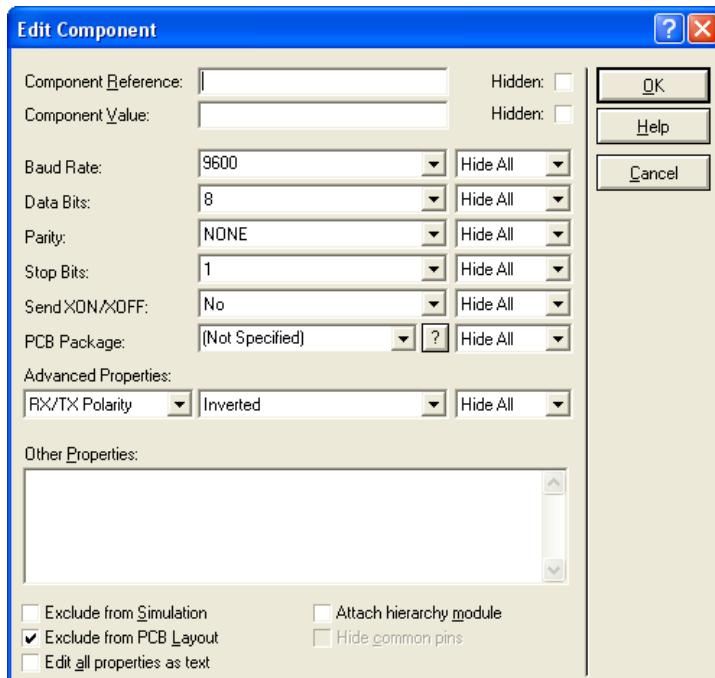
Algunas simulaciones en ISIS de PROTEUS pueden ocasionar la sobrecarga del microprocesador del ordenador, debido a la inmensa cantidad de cálculos que éste debe realizar en períodos muy cortos de tiempo; de ser así, el simulador hará una llamada de atención como la siguiente :



Al hacer clic en el símbolo de admiración se verá el siguiente mensaje de advertencia: *Simulation is not running in real time due to excessive CPU load*. En algunos casos puede corregirse este inconveniente reduciendo la frecuencia del oscilador, para lo cual se hace clic derecho sobre el microcontrolador y se selecciona el comando *Edit Properties*, que abre la ventana *Edit Component*; allí será posible cambiar la frecuencia en la casilla *Processor Clock Frequency*. Otra opción es excluir de la simulación algunos de los componentes del circuito (obviamente no será posible ver el resultado completo de la simulación); para ello, se debe hacer clic derecho sobre el componente que se desea excluir y seleccionar el comando *Edit Properties*, que abre la ventana *Edit Component*; marcar la opción *Exclude from Simulation*. La idea al hacer esto es utilizar un instrumento, por ejemplo el osciloscopio, para ver algunas formas de onda de

relevancia que nos brinden información fundamental acerca del funcionamiento del circuito en cuestión. Como ejemplo puede tomarse el problema resuelto **PWM\_1.dsn** que produce la sobrecarga del ordenador. Allí es posible excluir de la simulación los componentes 4N25, IRFZ44N y el motor DC. A continuación se puede conectar el osciloscopio en el pin CCP1 y observar la forma de onda PWM de salida, actuando sobre el pulsador.

## SIMULACIÓN DEL MÓDULO USART



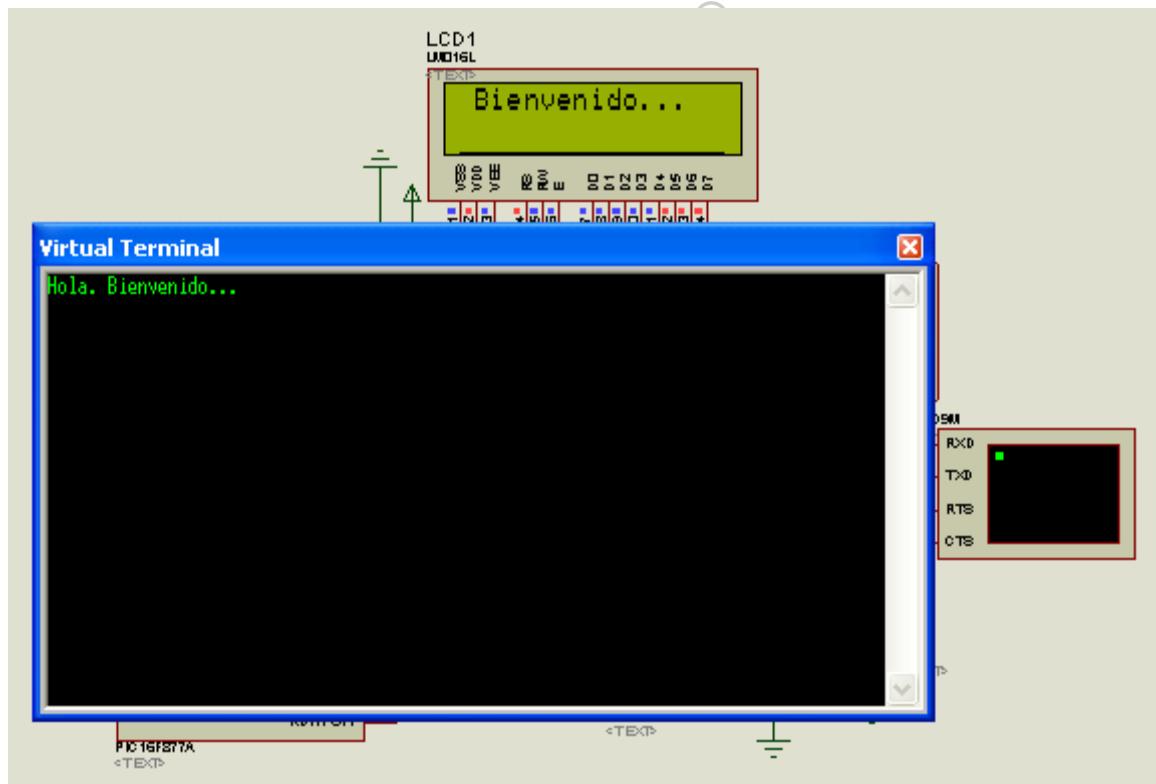
del ordenador. Para ingresar texto a través del teclado del ordenador se debe hacer clic en esta ventana para que aparezca el cursor parpadeante. Lo explicado se puede verificar fácilmente con el problema resuelto **USART\_01.dsn**.

Para simular la comunicación con el ordenador a través del módulo USART del microcontrolador, ISIS dispone de un instrumento excepcional llamado Terminal Virtual (*Virtual Terminal*). Para emplearlo se lo debe seleccionar de la lista de instrumentos haciendo clic en el ícono de

instrumentos virtuales . A continuación conectar los terminales RXD <-> T2OUT y TXD <-> R2IN (T2OUT y R2IN son pines del MAX232). Por último, hacer doble clic sobre el Terminal Virtual y constatar que esté configurado de acuerdo a los datos de la siguiente figura:

*Configuración del Terminal Virtual*

Al ejecutar la simulación se podrá ver una ventana similar a la que se muestra en la siguiente figura, que representa la pantalla



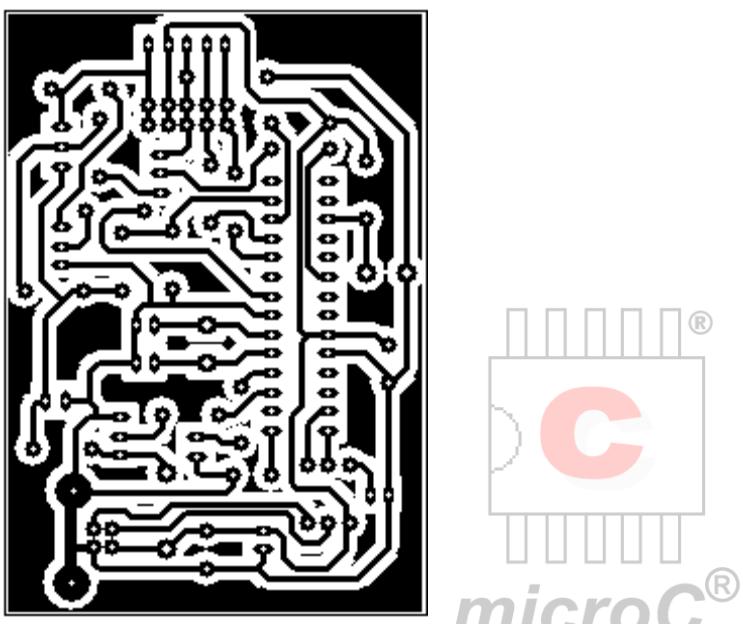
*Ventana del del Terminal Virtual (USART\_01.dsn)*

## APÉNDICE D: PROGRAMADOR PICkit2 Clone PARA PUERTO USB

Este programador es una versión simplificada del original PICkit2 de Microchip y trabaja con el programa de aplicación PICkit2 v2.61 de Microchip, por lo tanto su buen funcionamiento está garantizado y asegurado. En la página de Microchip se informa que puede trabajar correctamente en Windows XP y Windows Vista. Adicionalmente ha sido probado exitosamente en el ambiente Windows 7 Home Premium. La lista de microcontroladores PIC compatibles con este programador es muy extensa y se puede ver ingresando a *Help->ReadMe* en el programa de aplicación PICkit2 v2.61, o en nuestra página, ingresando a *Ofertas-> Más Detalles*.

### CIRCUITO IMPRESO VISTO DESDE LA CARA DE COMPONENTES.

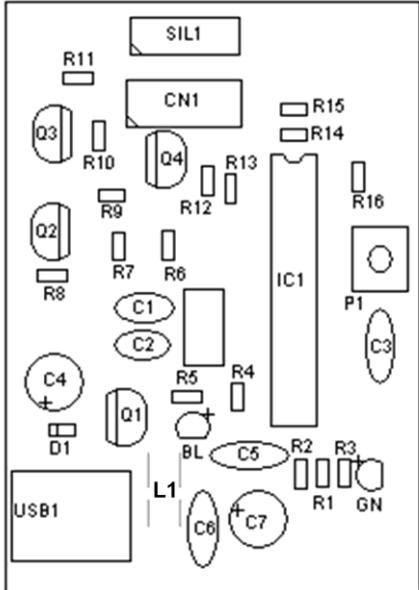
Esta es la imagen del circuito impreso para la construcción de este programador visto desde la cara de componentes (no desde la cara de la soldadura).



### UBICACIÓN DE COMPONENTES

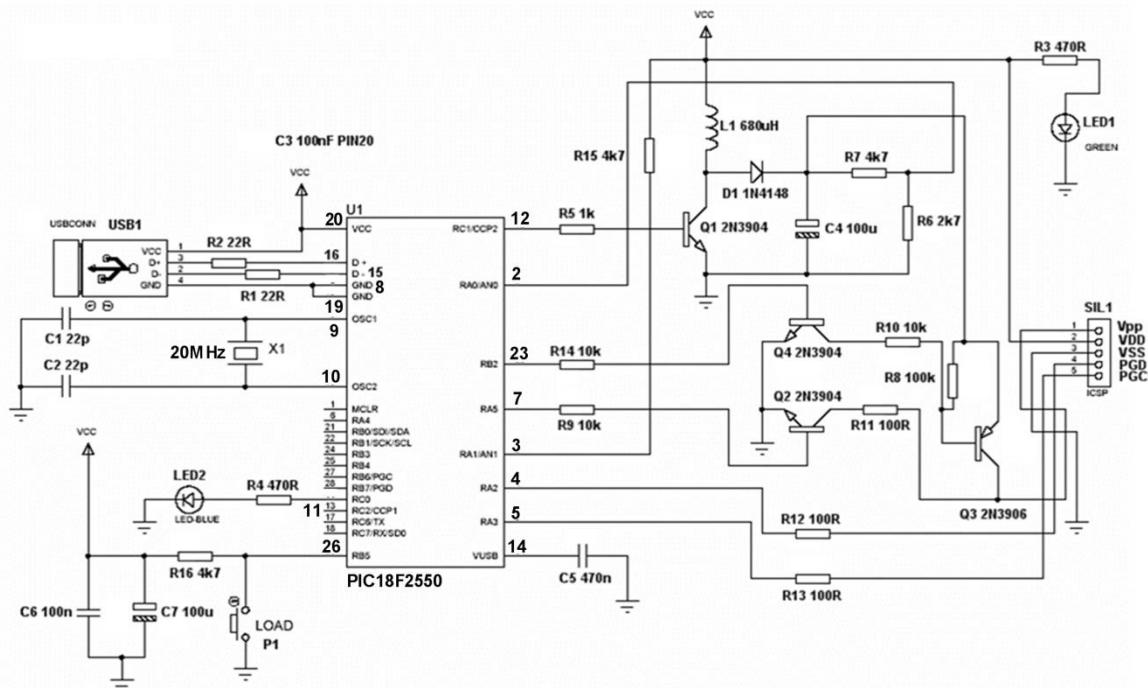
[www.programarpicenc.com](http://www.programarpicenc.com)

©Ing. Juan Ricardo Penagos Plazas

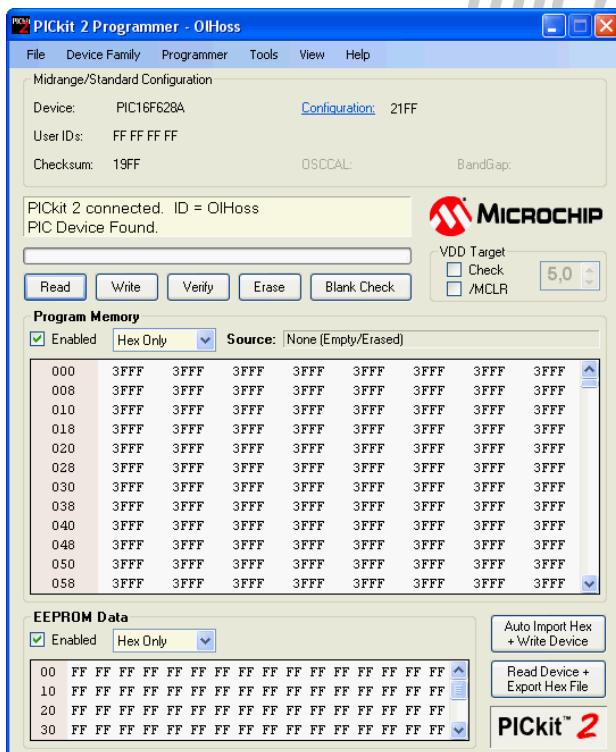


## ESQUEMA ELÉCTRICO DEL PROGRAMADOR

Este esquema ha sido probado con el software PICkit2 v2.61 y el *firmware* (PK2V023200.hex) correspondiente a esta versión de la aplicación. El *firmware* es un programa ejecutable que debe ser grabado en el PIC18F2550 (lamentablemente eso requiere tener a disposición otro programador de PICs). El software de aplicación y el firmware se pueden descargar de la página web de Microchip (el firmware también se puede encontrar en la carpeta *PICkit 2 v2* de la instalación del programa de aplicación PICkit2 v2.61). Los números de los componentes corresponden con la numeración de la placa de circuito impreso.



## PROCEDIMIENTO DE PROGRAMACIÓN



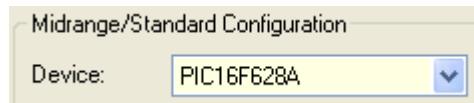
Su uso es muy sencillo. Se puede encontrar un video explicativo en nuestra web ingresando a *Ofertas-Más Detalles*.

1. Conecte el dispositivo (EEPROM, PIC, dsPIC, etc.) al programador. La conexión se realiza a través de los pines VPP (voltaje de programación), VDD (voltaje de alimentación), VSS (referencia), PGD (datos) y PGC (reloj) del programador y los pines correspondientes del microcontrolador (ver la hoja de especificaciones para cada dispositivo particular para identificar los cinco pines de programación). Debe emplear un tablero de proyectos de inserción a presión.
2. Conecte el cable USB al programador y a continuación el otro extremo del cable a un puerto USB del computador u ordenador.
3. Inicie el programa de aplicación “*PICkit 2 v2.61*”.

4. El programador será detectado automáticamente, al igual que el dispositivo a ser programado (siempre y cuando pertenezca a una de las familias que soporten autodetección). Para que la autodetección del dispositivo tenga efecto, el comando *Programmer->Manual Device Select* debe estar desactivado. Para el PIC16F628A (Midrange) se verá el siguiente mensaje:

PICkit 2 found and connected.  
[Parts in this family are not auto-detect.]

Si el dispositivo no soporta autodetección, debe ser seleccionado manualmente, para lo cual el comando  *Programmer->Manual Device Select* debe estar activado. Por ejemplo, para el PIC16F628A se debe seleccionar el comando *Device Family->Midrange->Standard*, y a continuación seleccionar el PIC de la lista desplegable *Device*:



5. Con el comando *File->Import Hex* abra el archivo ejecutable (\*.hex) que va a ser grabado en el dispositivo. Deberá observar el siguiente mensaje:

Hex file sucessfully imported.



6. Programe el dispositivo haciendo clic en el botón **Write**. Espere hasta que aparezca el mensaje de programación exitosa:

Programming Successful.

[www.programarpicenc.com](http://www.programarpicenc.com)

7. Desconecte el cable USB del computador y extraiga el dispositivo programado.  
8. Si va a programar otro dispositivo, conéctelo al programador, conecte nuevamente el cable USB al computador y luego seleccione el comando *Tools->Check Communication*. Repita los pasos 4 a 7.

**Nota:** Los dispositivos se encuentran agrupados por familias, así que si no logra encontrarlo en una de las familias vaya al menú *Device Family* y búsqüelo en las otras familias.

## APÉNDICE E: CONSIDERACIONES PRÁCTICAS

Las condiciones de funcionamiento en un ambiente real (no simulado) pueden ocasionar resultados inesperados e incontables dolores de cabeza, fundamentalmente debido al ruido eléctrico. Con el fin de disminuir al mínimo estos problemas de operación, se sugiere tomar muy en cuenta las siguientes recomendaciones y aplicarlas desde el inicio en el diseño de circuitos prácticos:

### PINES NO UTILIZADOS

Si se deja un pin sin utilizar, puede dejarse desconectado pero configurado como SALIDA y programado en cualquier estado (ALTO o BAJO), o puede configurarse como ENTRADA con un resistor externo de 10k a V<sub>DD</sub> o V<sub>SS</sub>.

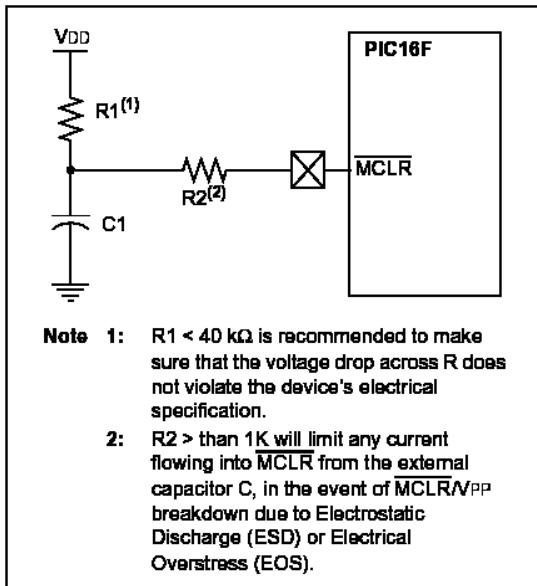
Las dos opciones permitirán que el pin sea empleado en lo posterior ya sea como entrada o salida sin realizar modificaciones importantes en el hardware.

### RESET INDESEADO #MCLR

La hoja de especificaciones de los PICs 16F88, 16F628A y 16F877A muestra que el reset #MCLR se producirá siempre y cuando se aplique un pulso negativo que tenga una duración mínima de 2us en este pin (T<sub>MCL</sub>).

Param No.	Symbol	Characteristic	Min	Typt	Max	Units	Conditions
30	TMCL	MCLR Pulse Width (low)	2	—	—	μs	V <sub>DD</sub> = 5V, -40°C to +85°C
31*	TWDT	Watchdog Timer Time-out Period (no prescaler)	7	18	33	ms	V <sub>DD</sub> = 5V, -40°C to +85°C
32	TOST	Oscillation Start-up Timer Period	—	1024 T <sub>Osc</sub>	—	—	T <sub>Osc</sub> = OSC1 period
33*	TPWRT	Power-up Timer Period	28	72	132	ms	V <sub>DD</sub> = 5V, -40°C to +85°C
34	TIOZ	I/O High-Impedance from MCLR Low or Watchdog Timer Reset	—	—	2.1	μs	
35	TBOR	Brown-out Reset Pulse Width	100	—	—	μs	V <sub>DD</sub> ≤ V <sub>BOR</sub> (D005)

Debido a que este tiempo es muy corto, es muy probable que se produzca un reset indeseado debido al ruido eléctrico (EMI, ESD o picos de voltaje) en el pin #MCLR. Para evitar este problema, que suele presentarse en ambientes industriales altamente contaminados eléctricamente, el fabricante sugiere emplear una red RCR, la cual puede tener los siguientes valores: R1=10k, R2=1.5k y C1=0,1 uF. El pin RA5/#MCLR debe estar configurado como #MCLR y no como E/S digital.

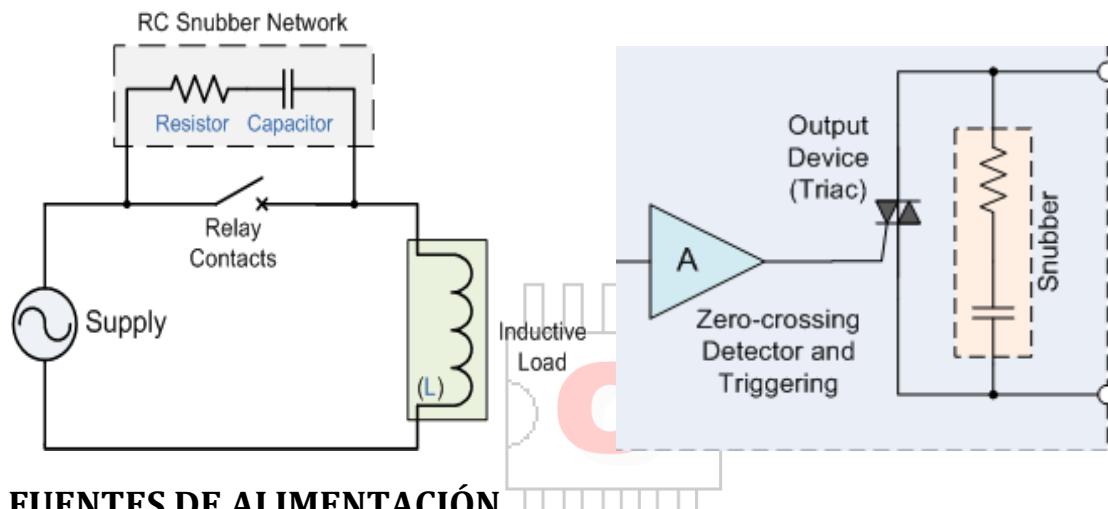


## RESET POR DESVANECIMIENTO (BOR)

Para tener la posibilidad de habilitar y deshabilitar el reset por desvanecimiento (*Brown-out Reset BOR*) a voluntad, se debe conectar siempre un capacitor de desacoplo de 100 nF (0.1 uF) lo más cerca posible de los pines de alimentación del PIC ( $V_{DD}$ - $V_{SS}$ ), para evitar que se produzca un reset BOR indeseado (si está habilitado) cuando cualquiera de las salidas del microcontrolador cambia de estado.

## FUNCIONAMIENTO ERRÁTICO DEL PIC

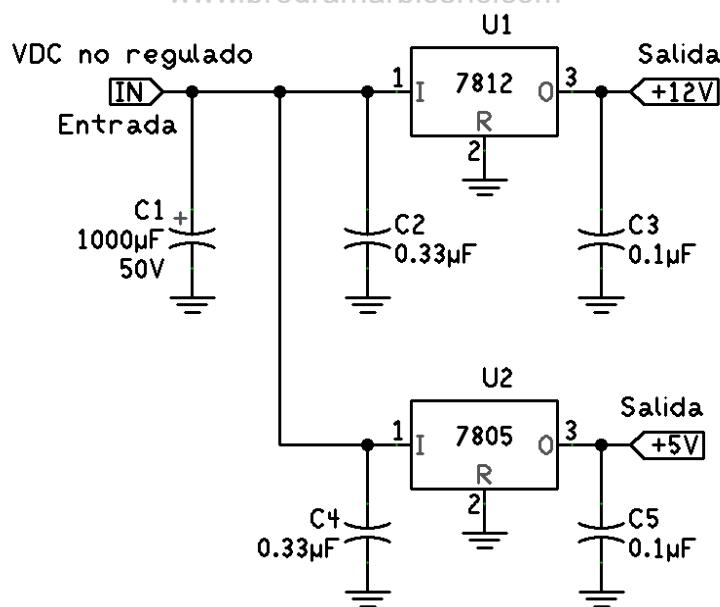
Para evitar que el PIC opere de manera inesperada (por ejemplo RESET indeseado, ejecución incorrecta del programa) se recomienda colocar una red *snubber* entre los contactos de los relés electromecánicos y de estado sólido (TRIAC), esto ayuda a atenuar los transitorios que se producen en el momento de la conexión y desconexión del relé, especialmente cuando hay cargas inductivas (motores, solenoides). El *snubber* está formado por una conexión RC en serie, que puede ser  $R=100\Omega/0,5W$  y  $C=0,1\mu F/250V$  (el capacitor debe ser de alto voltaje ya que soportará normalmente los voltajes pico de CA).



## FUENTES DE ALIMENTACIÓN

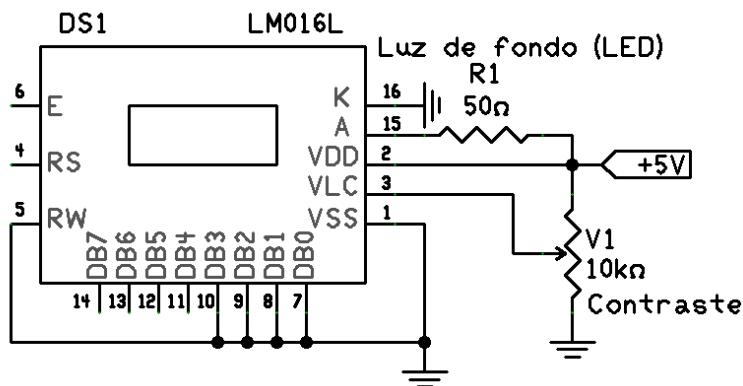
Otra cuestión de extrema importancia es la regulación de la fuente o de las fuentes de alimentación, tanto del PIC como de los demás elementos (por ejemplo, relés de 12V). La regulación correcta de las fuentes puede ahorrar incontables quebraderos de cabeza a la hora de construir una aplicación real.

[www.programarpicenc.com](http://www.programarpicenc.com)



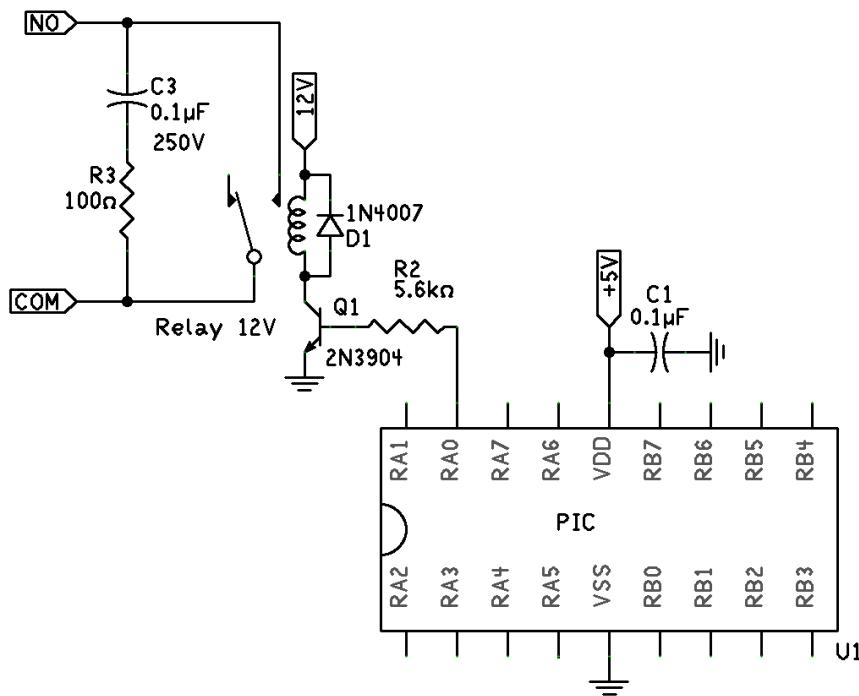
## LCD ALFANUMÉRICO (CON EL CONTROLADOR HD44780 O COMPATIBLE)

Este LCD posee un terminal para el control de contraste (VLC) por medio de un potenciómetro, y dos terminales adicionales correspondientes al ánodo (A) y cátodo (K) de un diodo emisor de luz (LED) que se utiliza para el alumbrado de fondo del módulo LCD. El LED se polariza por medio de un resistor de 50 ohm / 0,25 W, lo que da como resultado una corriente de 25 mA aproximadamente.



## CONEXIÓN DE UN RELÉ ELECTROMECÁNICO AL PIC

Los relés electromecánicos (de contactos metálicos) son muy útiles para el control de un sinnúmero de cargas eléctricas, tanto de CA como de CD. Son ampliamente utilizados por sus prestaciones y bajo costo. Se pueden conectar a un microcontrolador por medio de un transistor de propósito general operando como interruptor de estado sólido y que actúa como amplificador de corriente.



## ÍNDICE ALFABÉTICO

- Archivos HEX, **12**  
 Banderas de interrupción, **70**  
 Base de tiempo, **14**  
 Baudios, **102**  
 bit, tipo de dato, **21**  
 Bits de configuración, **10**  
*BOR.* Véase Reset por desvanecimiento  
*Button*, función, **37**  
*ByteToStr*, función, **41**  
 Capacitor de desacople, **183**  
 Carácter NULO (0x00), **117**  
 Cargas inductivas, **123**  
 Cátodo común, **26**  
 Cerradura electrónica-Ejemplo, **91**  
 char, tipo de dato, **21**  
 Ciclo de instrucción, **33**  
 Ciclo de trabajo, **93**  
 Ciclo interno de escritura-24LC256, **115**  
 Ciclos de reloj, **33**  
 Código de máquina, **7**  
 Código ejecutable, **176**  
 Código ensamblador-Insertar, **67**  
 Código fuente, **15**  
 Códigos ASCII, **89**  
 Comentarios, **21**  
 Computadoras personales-RS232, **102**  
 Conector DB-25/DB-9, **102**  
**CONFIG**, **10**  
**CONFIG1**, **10**  
**CONFIG2**, **10**  
 Control de contraste (VLC), **184**  
 Control de potencia-PWM, **93**  
 Control de velocidad de un motor DC, **95**  
 Conversor AD/DA (PCF8591), **114**  
 CPU, **21**  
 Cristal externo, **2**  
 Cristal líquido (LC), **135**  
 Dado electrónico-Ejemplo, **39**  
*Dec2Bcd*, función, **30**  
 Decimales en LCD, **45**  
 Delta, **13**  
 Desbordamiento del registro TMR0, **70**  
 Diodo emisor de luz-LED, **74**  
*dip switch*, **27**  
 Dirección de interrupción, **60**  
 Diseño de caracteres personalizados-LCD, **46**  
 Display de 7 segmentos, **26**  
 do...while, instrucción, **18**  
 Driver para Windows NT/2000/XP (icprog.sys), **170**  
 Duty cycle. Véase Ciclo de trabajo  
 EEPROM  
     Grabar el valor inicial, **48**  
 Efecto de rebote (*bouncing*), **37**  
 Ejecución condicional, **16**  
 Ejecución de una instrucción, **21**  
 Ejecución paso a paso, **14**  
 Entrada de datos, **16**  
 Errores de compilación, **21**  
 FALSO-Lenguaje C, **18**  
*Film compensated STN (FSTN)*, **136**  
 Firmware, **180**  
 Flanco ascendente o descendente, **70**  
 float, tipo de dato, **21**  
*FloatToStr*, función, **45**  
 for, instrucción, **17**  
 Forma de onda, **166**  
*Full Step-Motor PAP*, **127**  
 Función de transferencia-A/D, **97**  
 Función desconocida, **21**  
 Funciones, **16**  
 Funciones nulas, **20**  
 Funciones numéricas, **20**  
*Half Step-Motor PAP*, **127**  
 Hitachi HD44780, **40**  
 IDE, **12**  
 if...else, instrucción, **16**  
 int, tipo de dato, **21**  
**interrupt**-Función, **71**  
 Intervalo de temporización *T*, **51**  
 INTOSC, **6**  
 IOFS, **6**  
 ISR. Véase Rutina de servicio a la interrupción  
 Lazo infinito, **18**  
 Lazos, **16**  
 Lectura continua, **71**  
 Leer-modificar-escribir, **22**  
 Llaves { }, **16**  
 Lógica negativa-RS232, **102**  
 long, tipo de dato, **21**  
**main**-Función principal, **21**  
 Manejador de Librerías (*Library Manager*), **30**  
 Mapa de bits (\*.bmp), **161**  
 Máquinas de tipo “Su turno”-Ejemplo, **49**  
 Memoria no volátil-EEPROM, **48**  
 Memorias EEPROM serie, **114**  
 Mensaje largo en LCD, **44**  
 Menú *Edit*, **13**  
 Menú *File*, **13**  
 Menú *Help*, **13**  
 Menú *Project*, **13**  
 Menú *Run*, **13**  
 Menú *Tools*, **13**  
 Menú *View*, **13**  
 Microprocesador, **1**  
 Microsoft® Paint, **160**  
 Nibble, **88**  
 Números binarios, **21**  
 Números decimales, **21**  
 Números hexadecimales, **21**  
 Onda cuadrada-Ejemplo, **82**  
 Operación NOT (con bits), **29**  
 Operación OR (con bits), **28**  
 Operación SHIFT LEFT, **29**  
 Operación XOR ^, **29**  
 Operaciones, **16**

- Operador de asignación (=), **21**  
 Operador de comparación (==), **21**  
 Operador de negación !, **18**  
 Operadores de relación, **17**  
 Operadores lógicos, **17**  
 OSCCON, **6**  
 Oscilador externo, **7**  
 Oscilador interno de 4MHz-PIC16F88, **37**  
 Oscilador interno de 8MHz-PIC16F88, **79**  
 Osciloscopio básico-Ejemplo, **166**  
 Periféricos, **1**  
 PIC16F877A, Polarización básica, **27**  
 PICkit2 v2.61-Aplicación, **179**  
 Polling. Véase Lectura continua  
 Problema (de programación), **16**  
 PROTEUS VSM-Simulador, **172**  
 Puerto A-PIC16F628A, **23**  
 Puerto A-PIC16F88, **23**  
 Puerto B-PIC16F628A, **24**  
 Puerto B-PIC16F88, **24**  
 Puerto paralelo (puerto de impresora LPT1), **168**  
 Puertos COM, **102**  
 Pull up del puerto B, **24**  
 Pulsador normalmente abierto (NO), **37**  
 Punto y coma (;), **21**  
*Push-pull*, **121**  
 PWRT, **6**  
*Q. Véase* Valor de carga del TMR0  
 Registro ANSEL, **98**  
 Registro INTCON, **69**  
 Registro OPTION\_REG, **50**  
 Relés electromecánicos, **183**  
 Reloj calendario (DS1307), **114**  
 Reloj digital de 24 horas-Ejemplo, **86**  
 Reset por desvanecimiento, **6**  
 Resistencia de *pull-up*, **88**  
 Resolución-PWM, **93**  
 RMW. Véase Leer-Modificar-Escribir  
 ROM del Generador de Caracteres (CGROM), **45**  
 Rutina de servicio a la interrupción, **69**  
 Salida de datos, **16**  
 signed char, tipo de dato, **21**  
 Simulador, **13**  
 Sirena-Ejemplo, **84**  
 Snubber, **183**  
 Stopwatch, **13**  
*Strings* o cadenas de caracteres, **117**  
*Super Twisted Nematic (STN)*, **136**  
 switch, instrucción, **19**  
 Tabla de verdad-Ejemplo, **31**  
 Tecla de borrado, **91**  
 Temporizador de vigilancia, **57**  
 Termómetro (DS1624), **114**  
 Tipos de datos, **16, 21**  
 TMR0, **50**  
 TOCS, **50**  
 TRIAC, **183**  
*Twisted Nematic (TN)*, **135**  
 unsigned long, tipo de dato, **21**  
 unsigned, tipo de dato, **21**  
 Valor de carga del TMR0, **51**  
 Valor instantáneo, **165**  
 V<sub>DD</sub>, **6**  
 Vector de interrupción. Véase Dirección de interrupción  
 Velocidad de transmisión-RS232, **102**  
 VERDADERO-Lenguaje C, **18**  
 Voltímetro digital-Ejemplo, **100**  
 V<sub>ss</sub>, **6**  
*Watch Clock*, **13**  
*Watch Values*, **13**  
 WDT. Véase Temporizador de Vigilancia  
 while, instrucción, **18**



www.programarpicenc.com

©Ing. Juan Ricardo Penagos Plazas