USERS

¡INCLUYE PROYECTOS EXPLICADOS PASO A PASO!

MICRO CONTROLADORES

FUNCIONAMIENTO, PROGRAMACIÓN Y USOS PRÁCTICOS

- MICROCONTROLADORES PIC 16F Y 18F
- CONSTRUCCIÓN DE GRABADOR DE PICS Y PLACA EXPERIMENTAL
- MANEJO DE DISPLAYS LCD GRÁFICOS
- CONTROL REMOTO POR INFRARROJOS, RF Y TCP/IP
- CONECTIVIDAD ZIGBEE, BLUETOOTH Y WI-FI
- NETWORKING CON TCP/IP, RABBIT Y PIC



USERS MANUALES USERS MANUALES USERS MANUALES USERS MANUALES USER

ALCANCE UN NUEVO NIVEL EN EL MUNDO DE LA ELECTRÓNICA

CONÉCTESE
CON LOS MEJORES LIBROS DE COMPUTACIÓN

LLEGAMOS A TODO EL MUNDO
VÍA OCA* Y DHL**

usershop.redusers.com
usershop@redusers.com

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA



CONOZCA LAS MÁS NOVEDOSAS SOLUCIONES EN REPARACIÓN DE PC

» HARDWARE
» 320 PÁGINAS
» ISBN 978-987-1773-14-5



ARME, ACTUALICE Y MEJORE SU PC CON SUS PROPIAS HERRAMIENTAS

» HARDWARE / HOME
» 320 PÁGINAS
» ISBN 978-987-663-034-4



DISEÑO, ARMADO Y PROGRAMACIÓN EN C# DE ROBOTS COMPLEJOS

» DESARROLLO / .NET
» 352 PÁGINAS
» ISBN 978-987-663-020-7



ARME, CONFIGURE, DIAGNOSTIQUE Y REPARE SU PC COMO UN EXPERTO

» HARDWARE / HOME
» 320 PÁGINAS
» ISBN 978-987-663-001-6

MICROCONTROLADORES

Funcionamiento, programación
y usos prácticos

RedUSERS



TÍTULO: Microcontroladores

COLECCIÓN: Manuales USERS

FORMATO: 17 x 24 cm

PÁGINAS: 320

Copyright © MMX. Es una publicación de Fox Andina en coedición con DALAGA S.A. Hecho el depósito que marca la ley 11723. Todos los derechos reservados. Esta publicación no puede ser reproducida ni en todo ni en parte, por ningún medio actual o futuro sin el permiso previo y por escrito de Fox Andina S.A. Su infracción está penada por las leyes 11723 y 25446. La editorial no asume responsabilidad alguna por cualquier consecuencia derivada de la fabricación, funcionamiento y/o utilización de los servicios y productos que se describen y/o analizan. Todas las marcas mencionadas en este libro son propiedad exclusiva de sus respectivos dueños. Impreso en Argentina. Libro de edición argentina. Primera impresión realizada en Sevagraf, Costa Rica 5226, Grand Bourg, Malvinas Argentinas, Pcia. de Buenos Aires en VI, de MMXI.

ISBN 978-987-1773-56-5

Microcontroladores / coordinado por Daniel Benchimol. - 1a ed. - Buenos Aires : Fox Andina; Dalaga, 2011.
v. 213, 320 p. ; 24x17 cm. - (Manual users)

ISBN 978-987-1773-56-5

1. Informática. I. Daniel Benchimol, coord.
CDD 005.3



ANTES DE COMPRAR

EN NUESTRO SITIO PUEDE OBTENER, DE FORMA GRATUITA, UN CAPÍTULO DE CADA UNO DE LOS LIBROS EN VERSIÓN PDF Y PREVIEW DIGITAL. ADEMÁS, PODRÁ ACCEDER AL SUMARIO COMPLETO, LIBRO DE UN VISTAZO, IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA Y MATERIAL ADICIONAL.

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios, glosarios, atajos de teclado y todos los elementos necesarios para asegurar un aprendizaje exitoso y estar conectado con el mundo de la tecnología.



LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com // usershop@redusers.com



El libro de un vistazo

Esta obra nos entrega una serie de conceptos y prácticas que nos ayudarán a conocer en profundidad el funcionamiento de los microcontroladores, a través de sus diversos capítulos desarrollaremos diversos temas que se relacionan en forma directa con ellos. Interesantes temas, tales como Conectividad inalámbrica y Módulos Rabbit componen a este imperdible libro.

01 LÓGICA DIGITAL, MEMORIAS

En el primer capítulo de este libro nos dedicaremos a conocer y analizar los sistemas digitales y binarios, además conoceremos las compuertas lógicas y aprenderemos a realizar distintas operaciones de aritmética binaria. Finalmente conoceremos los circuitos secuenciales y las celdas de memoria, además de la aplicación de memorias PROM.

02 MICROCONTROLADORES

A través de este capítulo podremos conocer las características y funcionamiento de los microcontroladores, analizaremos el funcionamiento de las unidades de entrada y salida y conoceremos las historias de algunos microprocesadores famosos. Además veremos el funcionamiento de las microcomputadoras y también las interrupciones y los conversores analógico-digitales.

03 MICROCONTROLADORES PIC 16F

En este capítulo podremos conocer el funcionamiento y características de los microcontroladores PICF16, aprenderemos a crear un contador de un dígito con u display de siete segmentos y también analizaremos el funcionamiento de un PICF16 por dentro. Finalmente utilizaremos la aplicación denominada MPLABSIM.

04 PIC18F

Aquí podremos conocer las características que hacen destacar a la familia PIC18F, aprenderemos a trabajar con una placa experimental y enumeraremos las consideraciones que debemos tener en cuenta para su armado. Veremos la forma en que es necesario manejar un LCD inteligente y realizaremos un proyecto de alarma térmica.

05 CONECTIVIDAD (NO INALÁMBRICA)

En este capítulo conoceremos las distintas alternativas de conectividad no inalámbrica que tenemos a nuestra disposición, entre otras opciones analizaremos los protocolos de conectividad, serial peripheral interface y universal serial bus.

06 CONECTIVIDAD INALÁMBRICA

La conectividad inalámbrica será analizada a través de las secciones que componen a este capítulo, conoceremos diversas alternativas e conectividad, veremos el funcionamiento de ZigBee y XBee. También realizaremos un control por RF y por infrarrojos; finalmente trataremos la tecnología bluetooth y los módulos prearmados.



07 MÓDULOS RABBIT

En este capítulo analizaremos el funcionamiento de los módulos Rabbit, analizaremos Dyamic C y conoceremos los distintos elementos que conforman el mundo Rabbit.

08 DISPLAYS GRÁFICOS LCD

Los display LCD son tratados en profundidad a través de las secciones que componen a este capítulo, aquí conoceremos el funcionamiento de estos display y también su arquitectura, además aprenderemos a dibujar letras y gráficos en ellos.

09 NETWORKING CON TCP/IP

En este capítulo podremos conocer las opciones que nos ofrece el networking con TCP/IP, aquí podremos analizar el modelo OSI y también los alcances de TCP/IP. Trataremos la resolución de nombre con DNS y analizaremos la forma de realizar una conexión a un servidor web vía Telnet.

10 NETWORKING CON RABBIT

Aquí podremos conocer los alcances del networking utilizando Rabbit, analizaremos sus alcances y podremos profundizar en conceptos tales como RabbitWeb, Multitarea cooperativo y Multitarea cooperativo en Rabbit, entre otros.

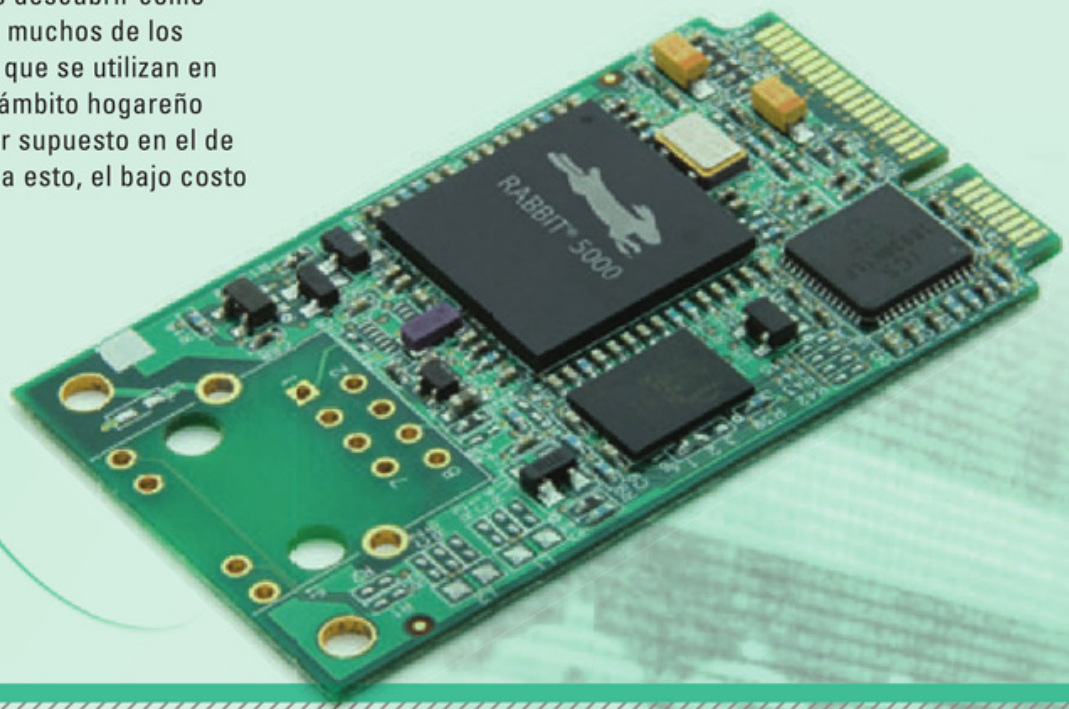
11 NETWORKING CON PIC

En el capítulo fina de este libro podremos conocer las alternativas que nos ofrece el networking con PIC. Analizaremos la forma en que debemos agregar servicios al stack, enviar mails, acceder desde Internet y monitorear la temperatura a través de Ethernet. También conoceremos el módulo ZeroG y configuraremos un router.



Prólogo

Para los expertos en microcontroladores, el alcance del mercado laboral hoy en día es inabarcable, ya que siempre existirá la industria que necesite digitalizar sus máquinas, desarrollar nuevas creaciones electrónicas o adaptar distintas tecnologías entre sí.
Cada día es más frecuente encontrarnos en la industria o en el hogar con máquinas y dispositivos conectados a Internet. Como ejemplo, en el sector agropecuario se han realizado importantes adelantos, añadiendo unidades de control electrónico para optimizar el desempeño de las máquinas de siembra y cosecha. Muchas de estas innovaciones han sido creadas por aficionados o técnicos electrónicos. Si bien el mercado actual y la masificación de los dispositivos electrónicos redujeron considerablemente sus costos y, en muchos casos, los hicieron descartables, existirán siempre aplicaciones que justifiquen la intervención de un técnico electrónico.
Por esta razón, ingresar en el universo de los microcontroladores es descubrir cómo se componen y funcionan muchos de los dispositivos tecnológicos que se utilizan en la actualidad, tanto en el ámbito hogareño como en el industrial y por supuesto en el de entretenimiento. Sumado a esto, el bajo costo de aplicación, la versatilidad y la eficiencia de los sistemas digitales hacen posible su aplicación en las computadoras, en los equipos de telefonía, en los de comunicaciones y en una larga lista de dispositivos de baja y alta complejidad. Estos aspectos son solo algunos de los motivos por los cuales el libro que tiene entre sus manos cobra especial relevancia, ya que nos acompañará a recorrer este apasionante mundo.
Tanto el aficionado a la electrónica como aquél que tiene sólo algunas nociones teóricas y prácticas en sistemas digitales encontrarán, en esta obra, las herramientas que necesitan para efectuar sus primeros proyectos. Por esto, nos encontramos ante una excelente oportunidad para que transitemos juntos este sorprendente recorrido donde se combinan aspectos vinculados con la programación, el hardware y la Robótica, entre muchos otros apasionantes temas.





Contenido

El libro de un vistazo	4
Prólogo	6
Introducción	10

Capítulo 1
Lógica Digital y Memorias

Sistemas digitales	12
Sistema binario	14
¿Por qué conocer el sistema binario?	16
Las compuertas lógicas	19
Aritmética binaria	24
Circuitos secuenciales	30
Multiplexores	35
Registros	36
Celda de memoria	38
Aplicación de memorias PROM	44
Actividades	50

Capítulo 2
Microcontroladores

¿Cómo funcionan?	52
Unidades de entrada y salida	56
Historias de microprocesadores	60
Microcomputadoras: ¿de qué se tratan?	62
Las interrupciones	66
Periféricos en microcontroladores	68
Conversores analógico-digitales	70
Actividades	76

Capítulo 3
Microcontroladores PICF16

Características del PIC16F	78
Contador de un dígito con display de 7 segmentos	85
Simulador de hogar de leña	88
PIC16F por dentro	90
MPLAB	92
Prácticas con MPLABSIM	95
El grabador de PICS	97
Actividades	98

Capítulo 4
PICF18

Características de la familia PIC18F	100
MPLAB C18	110
Placa experimental para PIC18LF4620	120
Consideraciones de armado	121
Periféricos externos	123
Contador PWM	126
Periféricos del PIC 18LF4620	131
Manejo del display LCD inteligente	135
Tecnologías de displays LCD	142
Proyecto de Alarma térmica	144
Proyecto Alarma térmica en MPLAB C18	148
Actividades	150

Contenido

Capítulo 5
Conectividad no inalámbrica

Protocolos de conectividad	152
Serial Peripheral Interface	156
Conexión de una memoria SPI	160
Bus de comunicación I2C	161
Conexión y manejo de un periférico I2C	165
Docklight	166
Universal Serial Bus	167
Actividades	170

Capítulo 6
Conectividad inalámbrica

Alternativas de conectividad	172
ZigBee y 802.15.4	174
Módulos prearmados (XBee y XBee ZB)	177
Conexión con la PC y confi guración	180
Red de sensores ZigBee	182
Control por RF	184
Control por infrarrojos	186
Módulos prearmados (Kcwirefree)	188
Actividades	190

Capítulo 7
Módulos Rabbit

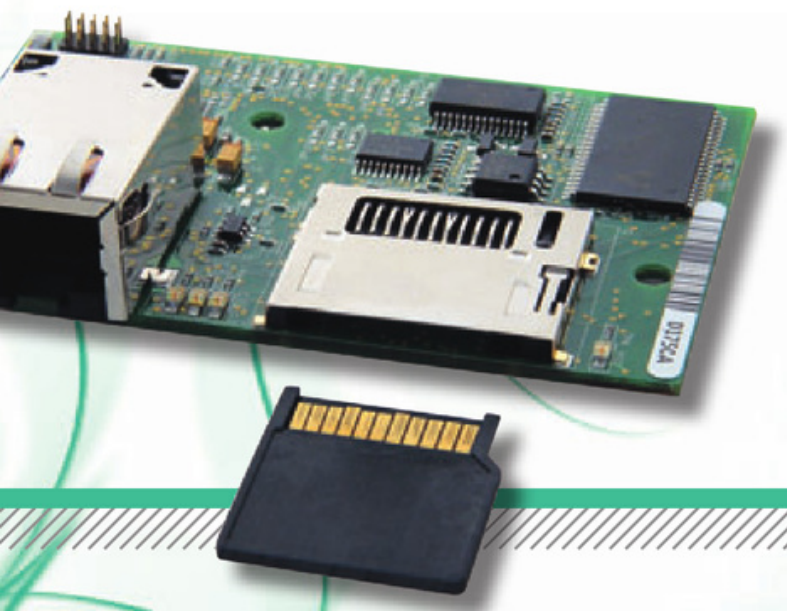
Módulos Rabbit	192
Dynamic C	195
Mundo Rabbit	200
Actividades	202

Capítulo 8
Displays gráficos LCD

Displays gráficos LCD	204
Arquitectura de un display digital	208
Cómo dibujar letras	210
Cómo dibujar gráficos y textos	211
Actividades	214

Capítulo 9
Networking con TCP/IP

Networking	216
Modelo OSI	219
TCP/IP	222
Resolución de nombres: DNS	224
TCP y UDP	226
Conexión a servidores web	228
Conexión a un web server vía Telnet	229
El lenguaje HTML	231
Ethernet con pic	232
Otras alternativas	234
Actividades	236



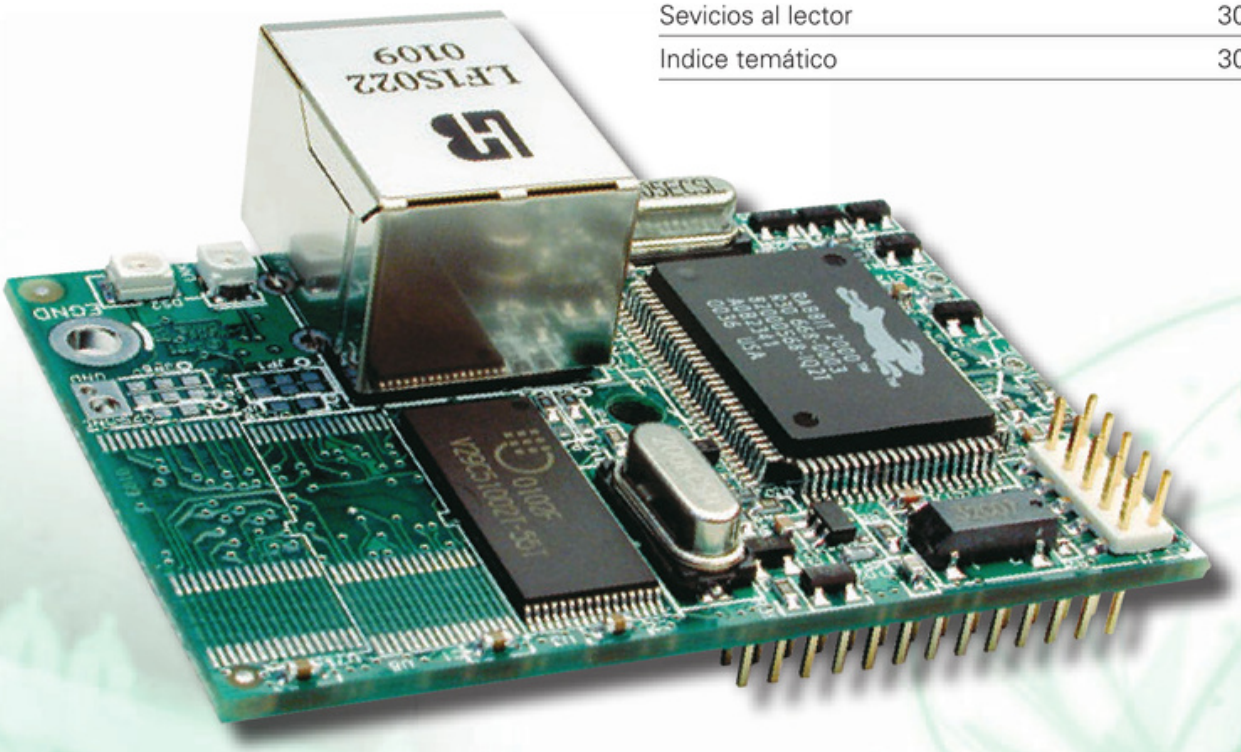
Capítulo 10
Networking con Rabbit

Las funciones de networking en Rabbit	238
Servidores web SSI y CGI	242
RabbitWeb	244
Servidor web con Rabbit	248
Conceptos integrados	250
WI-FI con Rabbit	252
Multitarea cooperativo	254
Multitarea cooperativo en Rabbit	256
Actividades	258

Capítulo 11
Networking con Rabbit

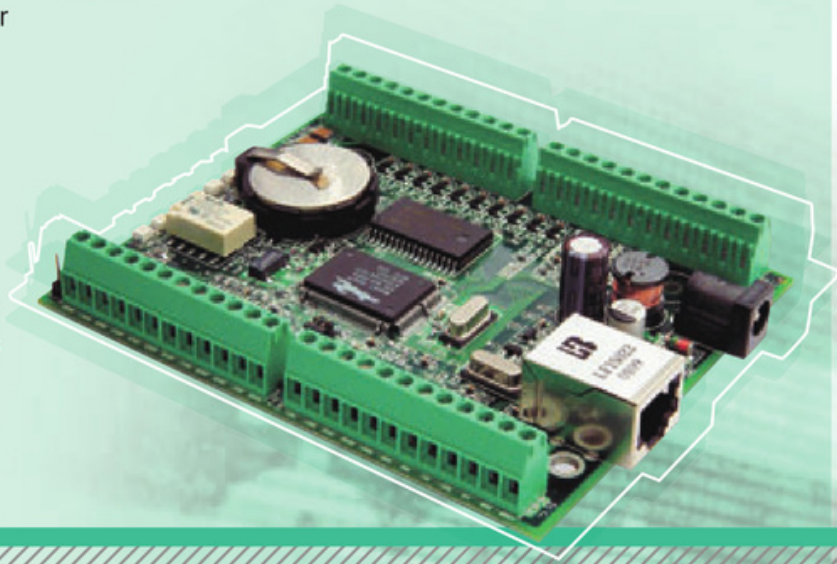
Introducción a networking	260
Agregar servicios al stack	262
Confi guración del web server	264
Envío de e-mails	275
Puente serie-Ethernet	277
Acceso desde Internet	278
Arquitectura del PIC18F97J60	280
Telemetría	282
Monitoreo de la temperatura por Ethernet	286
Servidores web dedicados	288
Encender un led y un relay por Ethernet	290
Módulo ZeroG	294
Confi guración del stack TCP/IP para Wi-Fi	301
Actividades	306

Sevicios al lector	307
Indice temático	308



Introducción

La Electrónica Analógica trata con conceptos que varían de forma continua con el tiempo, por ejemplo tensión, corriente y potencia, entre otras. Como ejemplo podemos citar una señal de audio o la tensión que proporciona un sensor de temperatura. Sabemos que todas estas señales están compuestas por múltiples valores de tensión y la Electrónica Analógica considera todos estos valores, como contrapartida, en la Electrónica Digital, las variables pueden tomar solo dos valores o estados posibles. Dado que estamos tratando el aspecto eléctrico del concepto, esos dos valores pueden ser 0 V y 5 V o, de forma binaria, 0 y 1. La elección del valor de 5 V no es caprichosa o antojadiza. Es debido a que la mayoría de los sistemas digitales trabajan con una tensión de 5 V. Sin embargo, en la actualidad, podemos encontrar a menudo sistemas que operan con tensiones de 3.3 V y aún menores, dependiendo del circuito.
En Electrónica Analógica y Digital se utilizan exactamente los mismos componentes discretos o primitivos, como las resistencias, los diodos, los capacitores y los transistores, entre otros.
Es entonces que nos preguntamos cuál es la diferencia, ya que se utilizan los mismos componentes, tanto para Electrónica Analógica como Digital. La diferencia se halla en la manera de hacer funcionar estos componentes. Observemos lo siguiente: un transistor puede funcionar como una resistencia variable que opera como un interruptor eléctrico que anula el paso por completo de la corriente o la deja pasar. Estos dos estados del transistor se los conoce como estado de bloqueo (cuando está abierto) y estado de saturación (cuando se encuentra cerrado).
Cuando un transistor funciona solo con los estados de bloqueo y saturación (como si fuese un interruptor) podremos decir que está operando de manera digital, ya que no considera los valores intermedios. Un ejemplo de ello puede ser un interruptor, que conecta o desconecta una bombilla de luz. Si, en cambio, hacemos funcionar un transistor de manera tal que tenga en cuenta los estados o valores intermedios, el transistor estará operando de manera lineal o analógica. Como ejemplo, podemos citar el regulador de luminosidad o dimmer con que cuentan algunos hogares y que permiten graduar la intensidad de luz deseada emitida por la bombilla.
Pensando en eso es interesante analizar que la Electrónica Digital hizo su aparición al mismo tiempo que la Electrónica Analógica, pero se encargó de tomar una significación especial desde el momento en que se empleó para el diseño y la construcción de sistemas básicos de cómputo, el control industrial, los autómatas programables, los microcontroladores, los microprocesadores y los computadores personales. Gracias a que la Electrónica Digital considera solo dos valores posible (0 y 1), es mucho más simple procesar y manipular las señales de forma digital que hacerlo de manera analógica. Esta ventaja se produce a partir de que ésta última resulta inexacta.
A lo largo de este libro, se estudiarán los microcontroladores y las diversas tareas que podemos llevar a cabo gracias a su uso, de esta forma estaremos preparados para hacer frente a problemáticas relacionadas con la electrónica digital.





CAPÍTULO 1

LÓGICA DIGITAL Y MEMORIAS

Sistemas digitales
Sistema binario
¿Por qué conocer el sistema binario?
Las compuertas lógicas
Aritmética binaria
Circuitos secuenciales
Multiplexores
Registros
Celda de memoria
Aplicación de memorias PROM
Actividades

01

SISTEMAS DIGITALES

En electrónica, todo sistema digital trabaja con señales digitales de naturaleza binaria. Esto implica la presencia de solo dos estados posibles: conducción (ON) y corte (OFF), en equivalencia a verdadero y falso, nociones que maneja todo sistema lógico.
Los transistores de los circuitos integrados actúan como llaves de conmutación, al permitir la conducción o interrupción de un circuito eléctrico modificando el estado del sistema. Es decir, gobiernan el comportamiento lógico del circuito. Hablaremos entonces de transistores y tecnologías que se ajustarán a niveles de tensión representativos de dichos estados (ver tabla "Valores y estados", pág. 13).

Lógica combinacional y lógica secuencial

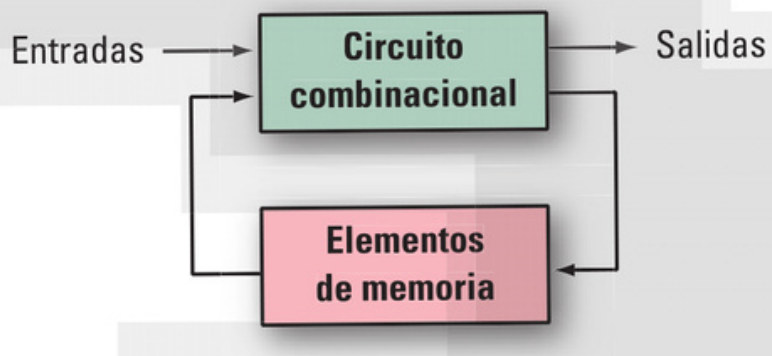
En este punto, debemos hacer una distinción entre lógica combinacional y lógica secuencial.
La **combinacional** se refiere a un sistema que reacciona siempre de la misma manera frente al mismo juego o combinación de entradas. Es decir, cada vez que se aplica una combinación de entradas determinada, el sistema devuelve el juego de salidas correspondiente. Esta es una operación sin memoria.
Un sistema **secuencial**, en cambio, tiene memoria. Responde no solo a una determinada combinación de entradas, sino que también coteja con algún resultado anterior para realizar una acción. Es decir, depende del orden y de la secuencia con que se ejecutan las combinaciones a su entrada.
Es importante aclarar que todo circuito digital puede desglosarse en dos grandes bloques: uno **combinacional** y otro **secuencial**, que actúan en conjunto.



Los sistemas de numeración

La necesidad de representar cantidades de un determinado objeto mediante símbolos ha llevado a desarrollar diversos métodos de representación.
Analizaremos a continuación los sistemas de nume-ración actuales. Históricamente, los esfuerzos se centraron en encontrar un sistema que precisara de la menor cantidad de símbolos para representar grandes cifras, y que facilitara las operaciones y cálculos. Los sistemas posicionales de numeración surgieron en forma independiente, tanto en Oriente como en América. En ambos casos, hay un símbolo representante de la **ausencia de cantidad: el cero.**
Los sistemas posicionales utilizan un conjunto limitado y constante de símbolos, donde cada uno representa una cierta cantidad de unidades. Pero, además, dependiendo de la posición que ocupe en el grupo de caracteres de representación, este símbolo tendrá mayor o menor peso. Nuestro sistema decimal, por ejemplo, es un caso típico de un sistema de **representación posicio-nal.**
Lo llamamos decimal pues, con la combinación de 10 dígitos, es posible representar cualquier cantidad: 0,1, 2, 3, 4, 5, 6, 7, 8 y 9. Veamos un

Diagrama de bloque de un circuito secuencial



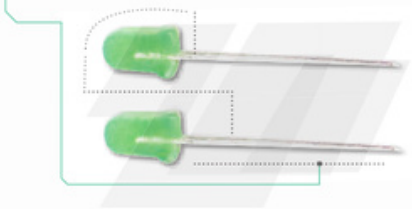
Vemos aquí la conformación de un sistema digital que integra módulos combinacionales y secuenciales.

ejemplo del peso de los símbolos de acuerdo con su posición. Podemos, entonces, descomponer la cantidad **18.127** de la siguiente forma:

18.127 = 1 x 10.000 + 8 x 1.000 + 1 x 100 + 2 x 10 + 7

Observemos que cada dígito que conforma el número **18.127** tiene un peso propio por la posición que ocupa en la cadena de caracteres. El peso de cada dígito en el sistema decimal es claramente múltiplo de 10. Luego, la descomposición que sigue también puede lograrse:

18.127 = 1 x 104 + 8 x 103 + 1 x 102 + 2 x 101 + 7 x 10^0



De aquí viene la denominación de sistema en base 10, sinónimo de sistema decimal. Este tipo de descomposición puede extenderse a los demás sistemas de numeración, como el binario que desarrollaremos más adelante.

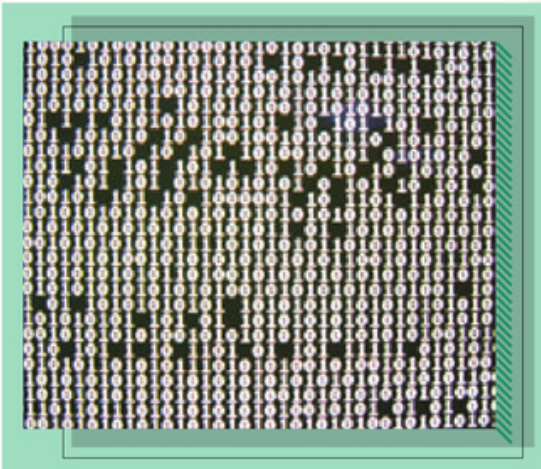
VALORES Y ESTADOS

VALORES	ESTADOS
5 V / 3,3 V / 1,8 V	ON (1 en binario o estado alto)
0 (cero) V	OFF (0 en binario o estado bajo)

Todas las operaciones lógicas podrán efectuarse en estas condiciones.



El decimal es un sistema de representación posicional de base 10; el binario es de base 2, y el hexadecimal, de base 16.



> REGLA POSICIONAL

Para todo sistema entero de numeración posicional se cumple que el último dígito de la derecha representa unidades (peso de valor 1). El peso de cada posición se incrementa de derecha a izquierda en potencias de la base.

123 = 1 x 100 + 2 x 10 + 3 = 1 x 102 + 2 x 101 + 3 x 100

mcdu = m x base3 + c x base2 + d x base + u (y así sucesivamente).

SISTEMA BINARIO

Se denomina sistema binario al sistema de numeración de base 2, que, como se desprende, consta de solo 2 dígitos para representar cantidades: 0 y 1. Con ellos podemos representar $2^2 = 4$ valores; con 3 dígitos, $2^3 = 8$ valores; con 4 dígitos, $2^4 = 16$ valores y, así sucesivamente; por lo tanto, con N dígitos se podrá representar hasta 2^N valores. Es importante destacar que los ceros a la izquierda no cuentan, como es lógico, a la manera en que estamos acostumbrados en el sistema decimal. Para operar con números binarios, lo haremos intuitivamente del mismo modo que cuando utilizamos el sistema decimal (ver tabla "Valores binarios", pág. 15).

Operaciones con números binarios

En las operaciones típicas de suma, resta y multiplicación, se aplican las técnicas de acarreo, adaptadas en este caso a la operación con solo 2 símbolos. Esto es: **01 + 01**, en binario, arrojará el valor 10, ya que **1 + 1** no puede representarse con un solo símbolo. Se deja entonces un 0 en la posición menos significativa y se acarrea un 1 hacia la más significativa.
El resultado es 10 binario (2 en decimal). La multiplicación es todavía más intuitiva y se realiza de manera habitual. La única posibilidad de que una multiplicación entre 2 bits arroje 1 como resultado es que ambos sean 1.

Es importante destacar que agregar un cero por derecha a un número binario tiene como resultado duplicar su valor. En efecto, dado el número 110 binario (6 decimal), el número 1100 (binario) corresponde al doble de su valor (12 en decimal).

Representación con signo

Es posible representar números binarios signados utilizando el bit más significativo como bit de signo: un 1 indicará negativo, y un 0, positivo. Por ejemplo, como se ve en la tabla "Representación con signo", el número **1010** –10 decimal en binario **no signado**– representaría el número **-6 decimal en binario signado**.

Multiplicación

```
   11001
 x 1101
 -------
   11001
+  11001
  11001
 --------
 101000101
```

Sumas

```
111  1  → Acarreo
 10010011
+ 1111010
 ---------
 100001101

  +  | 01
  0  | 01
  1  | 10  → Acarreo

   11
 1011001
+  1110
 -------
 1100111

  1111
 110000110
+ 100011111
 ----------
 1010100101
```

Restas

```
 1
 1000000000
-  100000000
 ----------
 0100000000

  -  | 10
  1  | 01  → Acarreo
  0  | 10

  10111010
- 10000000
 --------
  00111010

 1110111010
- 1000000000
 ----------
 0110111010
```

Observamos aquí las operaciones de suma, resta y multiplicación entre números binarios.

REPRESENTACIÓN CON SIGNO

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0111

Representación de números binarios con signo.

```
  10000    (Módulo 24 = 16 en decimal)
- 0110     (Representación positiva: 6 decimal)
  ----
  1010     (Representación negativa: -6 decimal)
```

> Cálculo de la representación negativa del número 0110 (6 decimal). Para módulo 10000 (24 = 16 decimal).

Debido a que un bit se utiliza como signo, el número máximo que es posible representar para los positivos es 0111 (7 decimal). El mínimo negativo será el 1000 (-8 decimal), y el máximo negativo, el 1111 (-1 decimal).

Concepto de módulo

En representaciones de números binarios signados, se llama módulo al resultado de sumar la representación negativa de un número con su representación positiva.
Por ejemplo, si sumamos **1111 (-1 decimal)** a **0001 (1 decimal)**, obtendremos el módulo: **10000**. Se dice entonces que estos números son complementarios con respecto a su módulo.

Trabajo con binarios signados

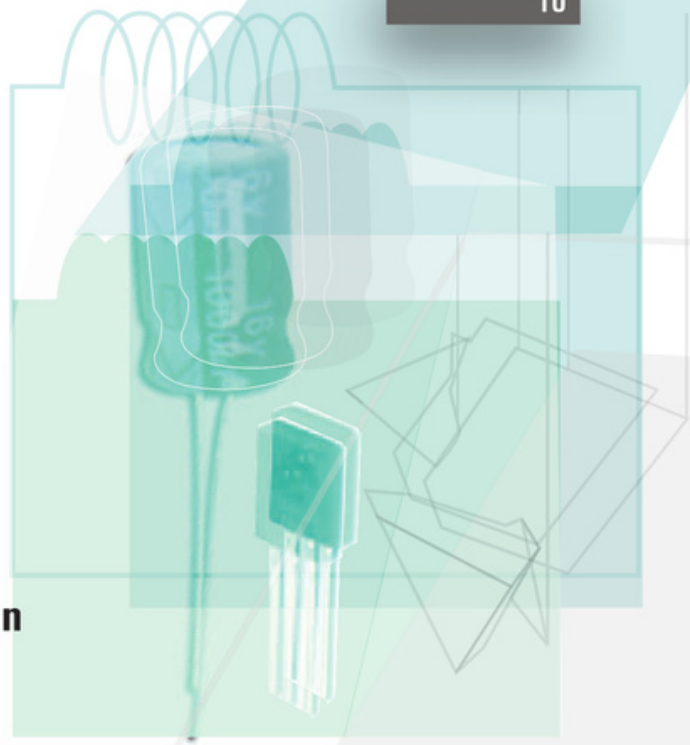
Cuando trabajamos con binarios signados, calculamos el módulo como el número binario correspondiente a la cantidad de combinaciones que se pueden lograr con la cantidad de bits utilizada incluyendo el signo.
Por ejemplo, para el caso de utilizar 4 bits (incluido el signo), se puede lograr $2^4 = 16$ combinaciones (de -8 a 7 pasando por el cero). Por lo tanto, el módulo de este arreglo de bits signados es 16; en binario: 10000.
Conociendo el módulo, es posible determinar la representación de números negativos, simplemente operando sobre la representación positiva, sin necesidad de tener a la vista todas las combinaciones posibles. Restando entonces al módulo la representación positiva, obtendremos la representación negativa de ese número con esa cantidad de bits. ■

VALORES BINARIOS

2 BITS	3 BITS	4 BITS	VALOR DECIMAL
00	000	0000	0
01	001	0001	1
10	010	0010	2
11	011	0011	3
	100	0100	4
	101	0101	5
	110	0110	6
	111	0111	7
		1000	8
		1001	9
		1010	10
		1011	11
		1100	12
		1101	13
		1110	14
		1111	15

Los números binarios pueden formarse a partir de arreglos de distintas cantidad de dígitos.

¿POR QUÉ CONOCER EL SISTEMA BINARIO?



Cuando hablamos de electrónica digital, nos referimos a sistemas electrónicos que procesan, almacenan, se comunican y operan en binario. Veremos aquí su importancia.

Un sistema digital de este tipo manejará internamente solo dos estados: 1 (alto) y 0 (bajo). En consecuencia, la organización de la información estará basada en arreglos de valores binarios. Surge entonces el concepto de **bit**, que no es más que el acrónimo de ***binary digit*** (dígito binario). Con un bit podremos representar dos estados o valores.
El arreglo de 8 bits se denomina ***byte***, término que deriva de la palabra anglosajona *bite*, '**mordisco**' en castellano. Se refiere a la cantidad mínima de datos que un procesador puede "morder" a la vez, adoptándose por convención el tamaño de 8 bits. La traducción al español que toma la Real Academia es **octeto**, pero nosotros utilizaremos **byte** para evitar confusiones. De allí, las unidades de capacidad y almacenamiento más utilizadas:

- **Kilobyte = 1024 bytes**
- **Megabyte = 1024 Kb**
- **Gigabyte = 1024 MB**
- **Terabyte = 1024 GB**

Con un byte es posible manejar 256 valores. Por ejemplo, en programación, se acostumbra definir variables de tipo carácter, de 1 byte de longitud. Se trabaja, así, con valores enteros en un rango de 0 a 255 (sin signo) o de -128 a 127 (con signo).
Es común también hablar de "words" o arreglos de 16 bits, no solo en programación de sistemas con microprocesadores y microcontroladores, sino también en algunos medios de almacenamiento, como memorias RAM y ROM, que proponen direccionamiento y palabras de datos mínimas de 16 bits. Por lo tanto, es una unidad de trabajo estándar que identifica una palabra de 2 bytes de longitud, muy utilizada para definir variables de tipo entero, capaces de manejar 65.536 valores distintos, suficientes para muchas de las operaciones más corrientes.

SISTEMA HEXADECIMAL

DECIMAL	BINARIO	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

En la tabla se muestra una equivalencia entre valores decimales, binarios y hexadecimales.

Conversión decimal a binario

Para convertir un número de decimal a binario, hay que dividir el decimal sucesivamente por 2, hasta obtener un cociente menor que el divisor. Este cociente (que será 0 o 1, naturalmente), más los restos de las sucesivas divisiones efectuadas, constituye la representación binaria buscada. De esta forma, el cociente de la división será el dígito más significativo del número binario, y el menos significativo corresponderá al primer resto de la división.

Conversión decimal a hexadecimal

Este método puede extenderse a conversiones de decimal a cualquier otro tipo de base, por ejemplo, hexadecimal. Para esto, habrá que dividir el

```
18 | 2
 0  9 | 2
    1  4 | 2
       0  2 | 2
          0  1    = 10010b
```

> Se muestra aquí el método de divisiones sucesivas para hallar la representación binaria de un número decimal.

```
5680 | 16
 88   355 | 16
      80   35   22 | 16
       0    3    6    1    = 1630h
```

> El mismo método de divisiones sucesivas para hallar la representación ahora hexadecimal de un número decimal.

número decimal sucesivamente por la base en la que se lo quiere representar; en nuestro caso, 16.

El orden de los dígitos del número hexadecimal obtenido es el inverso del que hemos obtenido en la división.

Conversión binario a decimal

Para realizar este procedimiento, hacemos el desarrollo en potencias de 2 de un número binario y sumamos los pesos:

$$110101b = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 53d$$

Es decir, en orden ascendente, el peso de cada dígito binario queda determinado: 1, 2, 4, 8, 16, 32, 64 (etc.). Observamos cómo aumenta en potencias de 2, duplicando el peso en cada dígito. La conversión a decimal se realiza efectuando la suma de las contribuciones de peso de cada dígito que tome valor 1.
En este caso, el número es 53.

Conversiones directas entre binario y hexadecimal

En los pasajes de binario a hexadecimal, y viceversa, se da una situación particular. Como una base es potencia de la otra, en nuestro caso **16 = 24**, es posible demostrar que los dígitos de la base menor (base 2 en nuestro caso) pueden agruparse en un número igual al exponente (4 en el ejemplo) para conformar un dígito de la base mayor.
Para simplificar: partiendo un número binario en cuartetos (agregando ceros a la izquierda a la parte más significativa, si es necesario), el reemplazo de dicho cuarteto por el símbolo hexadecimal equivalente se realiza en forma directa (ver tabla "Conversión de binario a hexadecimal"). ■

SISTEMA HEXADECIMAL

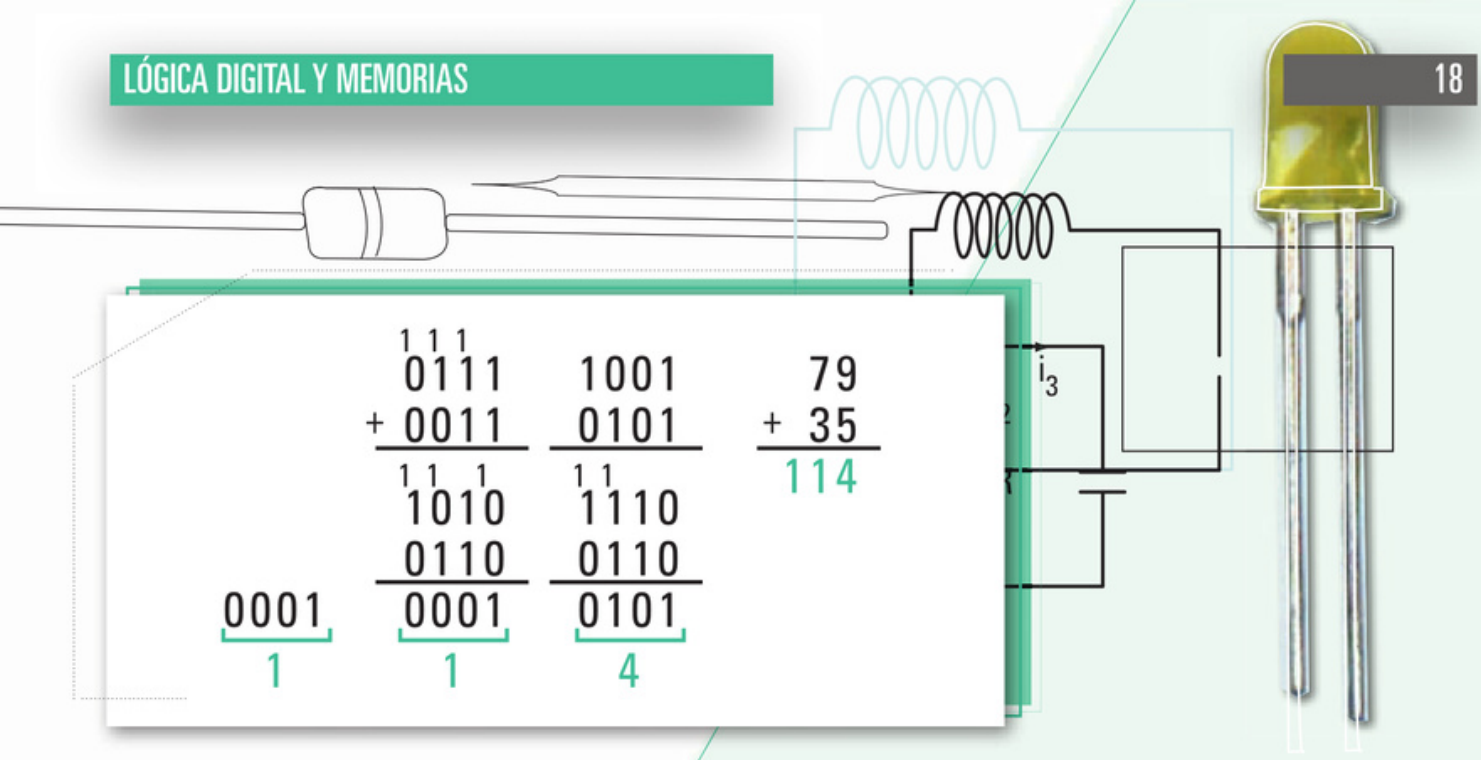
BINARIO	CUARTETOS	HEXADECIMAL
1110100101	(0011) (1010) (0101)	3A5h
101111	(0000) (0010) (1111)	02Fh
101011100001	(1010) (1110) (0001)	AE1h

Ejemplo de Conversión de binario a hexadecimal.



> ALCANCE HEXADECIMAL

El sistema hexadecimal consta de 16 símbolos para representar números. Del 0 al 9 coinciden con los símbolos del sistema decimal. Luego, se agregan los caracteres A, B, C, D, E y F para obtener el juego de 16 símbolos. Con solo dos dígitos hexadecimales, se podrán representar $16^2 = 256$ valores; con tres dígitos, $16^3 = 4096$, y con cuatro dígitos, $16^4 = 65536$.



Vemos aquí la operación de suma de números BCD.

Conversión inversa

La conversión inversa, de hexadecimal a binario, se lleva a cabo en forma directa y de la misma manera. Se reemplaza cada símbolo hexadecimal por un número binario de 4 bits. Por lo tanto, no se requiere ningún tipo de operación para los procedimientos de este tipo. El sistema hexadecimal es tan utilizado en sistemas electrónicos e informáticos. Para las personas resulta difícil trabajar en binario debido a que es fácil perderse con largas secuencias de unos y ceros para operar. El sistema hexadecimal permite representar de manera más compacta la información contenida en largas secuencias binarias que se almacenan y procesan en los equipos digitales.

El sistema hexadecimal permite representar la información comprimida.

Código BCD

Este tipo de codificación está diseñado con el objetivo de simplificar la conversión de decimal a binario, y evita la necesidad de recurrir a tediosas operaciones aritméticas, como en el método de divisiones sucesivas por 2. De este modo, el **código BCD** no es más que la representación binaria con 4 bits de cada dígito de un número decimal, en forma individual. Las conversiones de BCD a decimal y de decimal a BCD se realizarán, entonces, directamente, por simple inspección: **9723d = (1001) (0111) (0010) (0011) BCD**. Sin embargo, este tipo de representación es una equivalencia; no es una conversión matemática como el traspaso de binario a hexadecimal o de hexadecimal a binario. Es decir, no estamos representando números en ninguna base. Por lo tanto, la suma de los pesos de los bits individuales no arrojará el valor decimal representado en BCD. ■



> SUMA DE BCD

Para sumar números codificados en BCD, simplemente operamos como si trabajáramos con números naturales. Si la suma parcial de un cuarteto supera el valor 1001 (9 en decimal), se suma al resultado 0110 (6) y se acarrea 0001 al siguiente dígito BCD o cuarteto de bits.

LAS COMPUERTAS LÓGICAS

Los circuitos lógicos digitales han revolucionado la electrónica, ya que son mucho más inmunes al ruido eléctrico, más rápidos y más versátiles que su contraparte analógica.

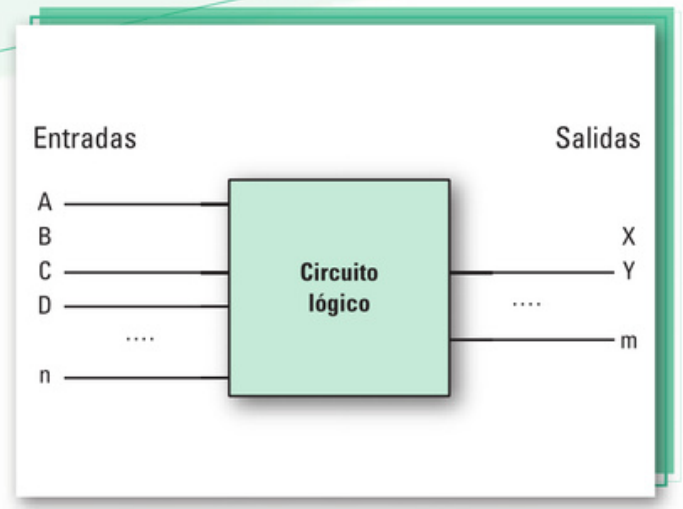


Figura 1. Representación de un circuito lógico. Las **n** entradas aplicadas producen las **m** salidas. Desde este punto de vista, no resulta necesario conocer su estructura interna.

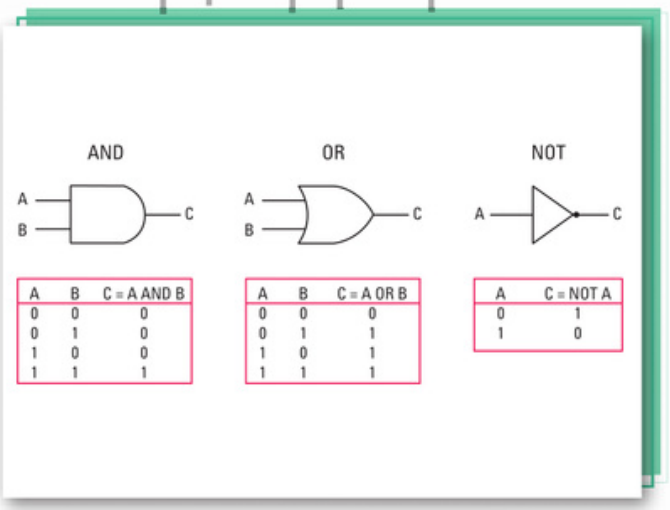


Figura 2. Compuertas lógicas elementales AND, OR, NOT y compuerta BUFFER. Su representación gráfica y su tabla de verdad, a partir de las cuales es posible construir cualquier circuito combinacional.

El continuo avance de la tecnología permite la aplicación de la electrónica digital cada vez en más áreas de especialización. Aquí veremos los bloques constitutivos de los circuitos digitales: las **compuertas lógicas**.
Un circuito lógico puede representarse con un mínimo de detalle, como una caja negra con una determinada cantidad de entradas y salidas. Entendemos por caja negra a un circuito del cual no tenemos datos sobre su composición interna, pero sobre el que sí podemos conocer cómo es su salida ante una cierta entrada. Como las entradas al circuito lógico pueden tomar solo los valores discretos **0 y 1**, la lógica llevada a cabo por él puede describirse completamente a través de una tabla que ignora su comportamiento eléctrico y define la salida con **0 y 1**.

Valores lógicos

Los valores lógicos se representan con un **0** o un **1**, y se denominan **dígitos binarios** o **bits**. Cada uno representa un rango de validez para una señal analógica. Un circuito lógico, cuya salida depende solo de sus entradas presentes, se conoce como **circuito combinacional**, y su funcionamiento se representa por medio de una **tabla de verdad**. Cualquier circuito combinacional puede construirse sobre la base de tres compuertas lógicas fundamentales, denominadas **AND, OR** y **NOT**. Existe otra compuerta particular llamada **BUFFER**, en donde la señal lógica no sufre ningún vcambio, es decir que la tensión de salida sigue a la de entrada. Estas compuertas se utilizan generalmente para regenerar señales débiles y convertirlas otra vez en señales fuertes para que puedan ser transmitidas a lo largo de cierta distancia sin pérdida de información.

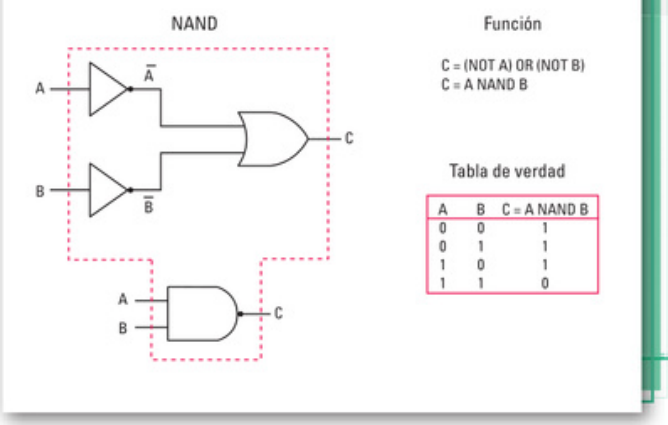


Figura 3. Compuerta lógica NAND. Implementación como compuerta OR con sus entradas negadas y su tabla de verdad. La compuerta NAND produce el resultado inverso de la compuerta AND.

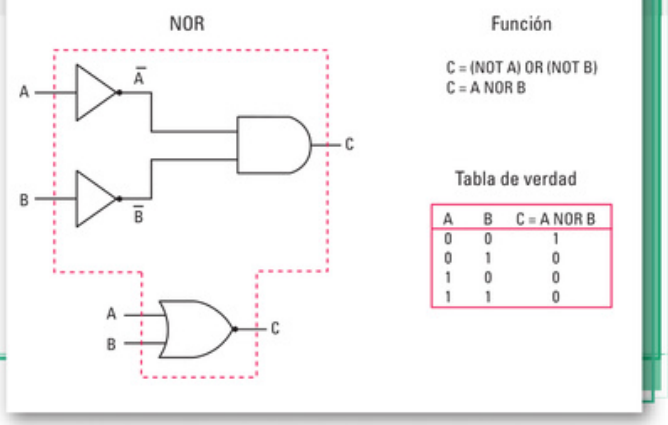


Figura 4. Compuerta lógica NOR. Implementación como compuerta AND con sus entradas negadas y su tabla de verdad. La compuerta NOR produce el resultado inverso de la compuerta OR.

En la **Figura 2** observamos la representación gráfica de la compuerta **NOT**. En la punta del triángulo se ha colocado un círculo que denota el carácter inversor de la función, e implica que el valor lógico presente en la entrada de la compuerta se invierte a la salida. De hecho, a la compuerta **NOT** también se la conoce con el nombre **inversor.** Si la entrada al inversor es **A**, entonces éste implementa la función **NOT A** y se representa con **Ā** (nombre de la entrada con una raya en la parte superior).

¿Qué sucede si en cada entrada de la función **OR** colocamos un inversor? La configuración lógica que se obtiene se conoce con el nombre **NAND** y es la implementación inversa de la función **AND**.

Ahora, veamos qué obtenemos si usamos la compuerta **AND** y colocamos inversores en sus entradas. La configuración lógica resultante se conoce como **NOR** y es la implementación inversa de la función **OR**.

Observemos la configuración lógica que se muestra en la **Figura 5**. Ésta se conoce como **OR-Exclusiva** o **XOR**, y su característica es que produce una salida lógica **1** cuando sus entradas son diferentes, mientras que arroja una salida lógica **0** cuando sus entradas son iguales.

De la electrónica a la lógica

Dijimos que un circuito lógico puede representarse como una caja negra con entradas y salidas. Esta caja negra es la que contiene un circuito electrónico que implementa la compuerta lógica representada por el bloque combinacional.
A continuación veremos algunos circuitos electrónicos muy básicos que implementan las compuertas lógicas elementales vistas anteriormente.

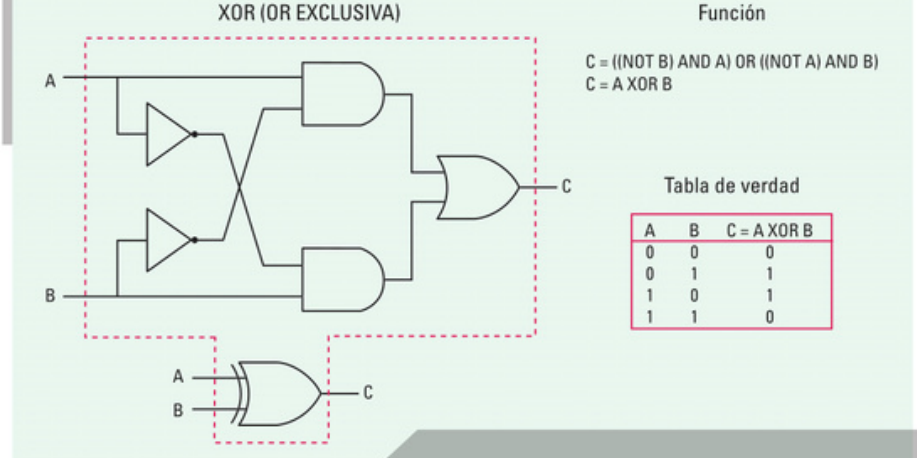


Figura 5. Compuerta lógica XOR. Implementación como secuencia AND-OR y su tabla de verdad. Se utiliza para identificar cuando dos entradas son iguales o distintas entre sí.

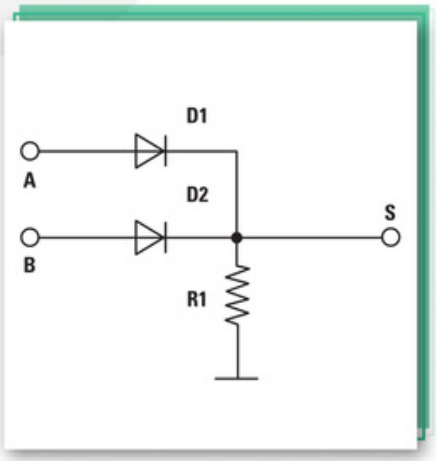


Figura 6. Una compuerta OR elemental con diodos y resistores (RDL, o Resistor Diode Logic).

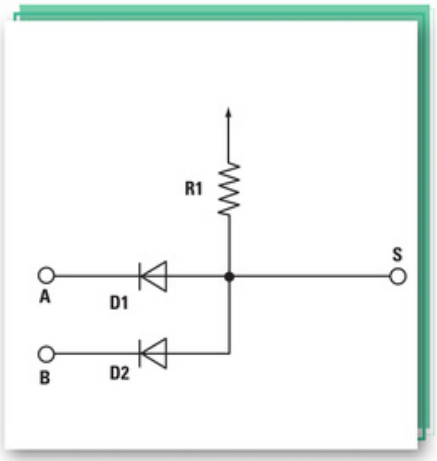


Figura 7. con diodos y resistores (RDL, o Resistor Diode Logic).

En el circuito de la **Figura 6**, si en cualquier entrada (**A**, **B**) se presenta un valor de tensión positiva que haga conducir al diodo, este valor se observa a la salida (**S**). En caso contrario, el resistor fuerza 0 V. Entonces, dado que cualquier entrada que esté en **1** ocasiona un **1** a la salida, esta es una compuerta **OR**, pues esta característica se corresponde con la tabla de verdad de dicha compuerta.

En el circuito de la **Figura 7**, si cualquier entrada (**A, B**) se conecta a 0 V, este valor se observa a la salida (**S**). En caso contrario, el resistor fuerza la tensión de alimentación. Entonces, dado que cualquier entrada que esté en **0** ocasiona un 0 a la salida, esta es una compuerta **AND**, pues esta característica se corresponde con la tabla de verdad de dicha compuerta.

Las compuertas lógicas implementan las funciones lógicas elementales.

El circuito de la **Figura 8** utiliza un transistor tipo **NPN, Q1**, para implementar la compuerta **NOT**. En este caso, con una tensión de **1** lógico en el punto **A**, el transistor entra en saturación, y el punto **C** se coloca a la tensión de **0** lógico. Cuando el transistor entra en corte a través de la aplicación de una tensión de **0** lógico en el punto **A**, el punto **C** queda a tensión de fuente menos la caída de tensión en el resistor **R1**.

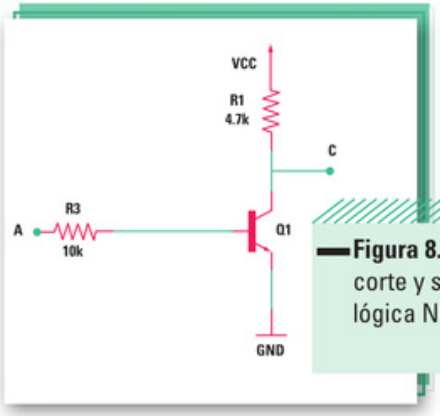
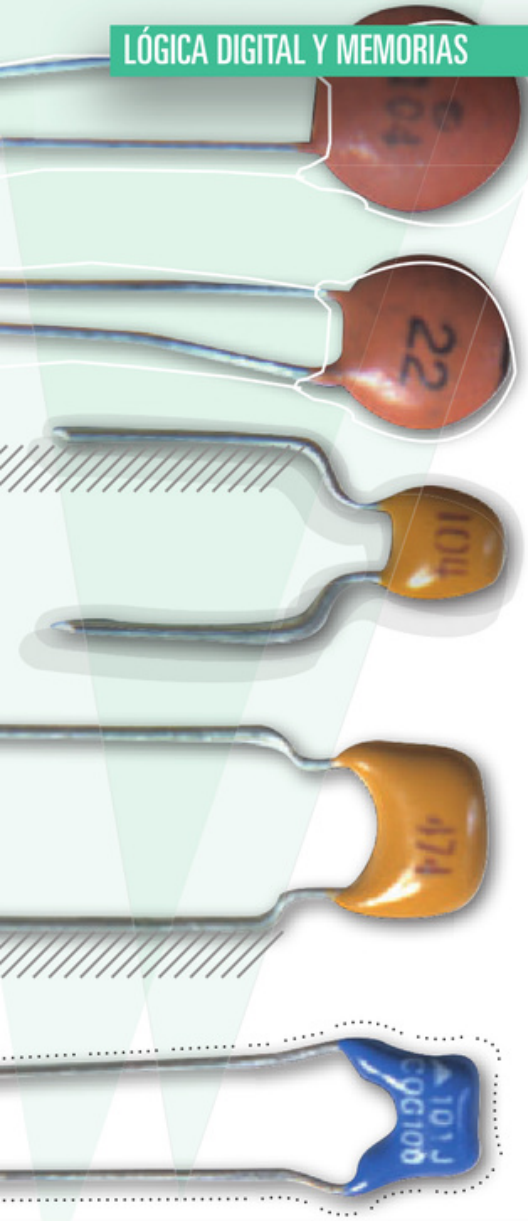
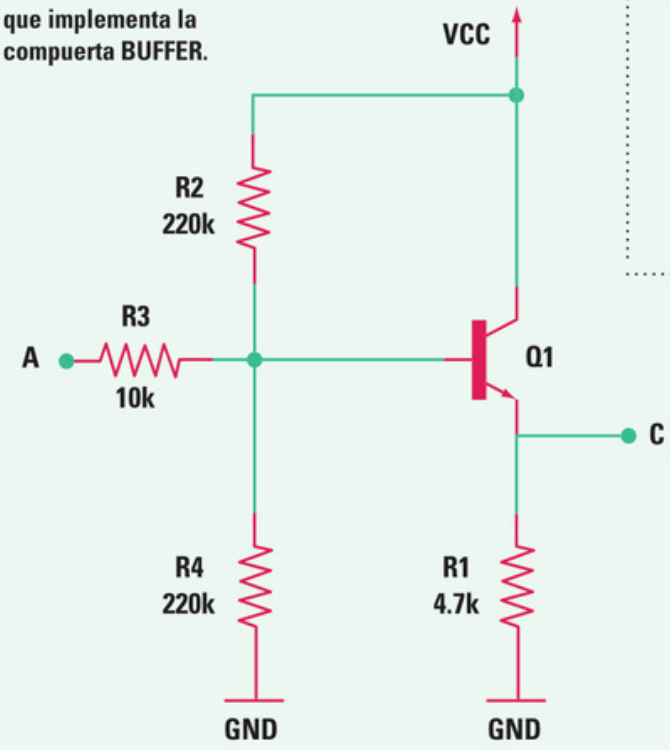


Figura 8. Circuito con un transistor trabajando a corte y saturación, que implementa la compuerta lógica NOT.



> **Figura 9. Circuito con un transistor, que implementa la compuerta BUFFER.**

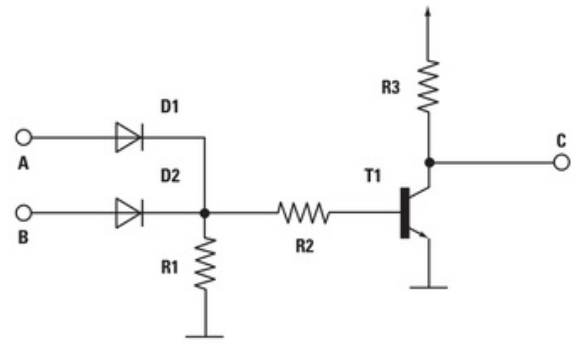


En la **Figura 9** observamos un circuito que implementa la compuerta BUFFER o seguidor de tensión. Cuando en el punto **A** tenemos una tensión correspondiente a un **0** lógico, el transistor **Q1** está en corte, y en el punto **C** tendremos una tensión equivalente a un **0** lógico. Cuando en A tenemos un **1** lógico, **Q1** conduce y coloca en el punto **C** a la tensión de fuente o **1** lógico.

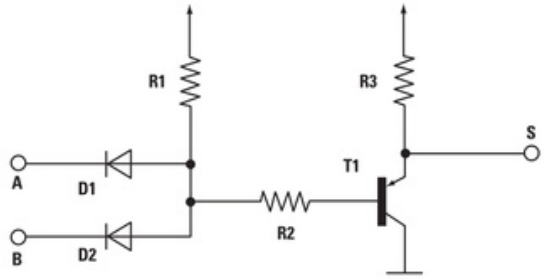
En la **Figura 10** se presenta la implementación de la compuerta **NOR** agregando un inversor (basado en un transistor PNP) en cascada con la compuerta **OR** que ya vimos en la **figura 6**.

La **Figura 11** muestra la implementación de la compuerta **NAND** agregando un inversor (esta

> **Figura 10. Circuito elemental que implementa la compuerta lógica NOR.**



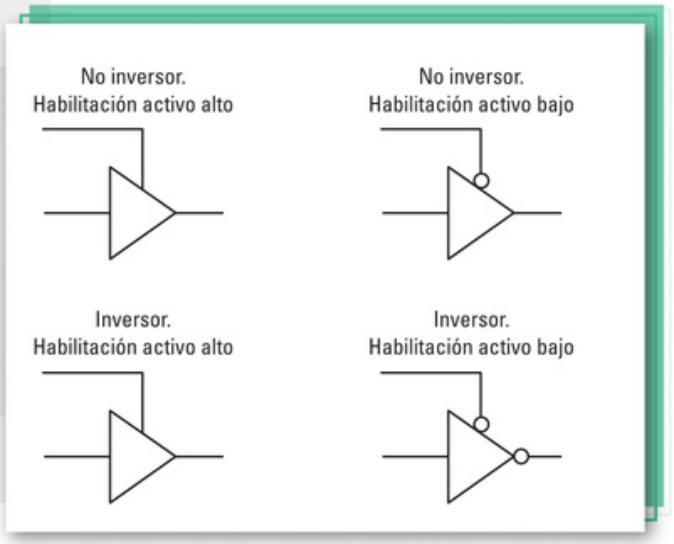
> **Figura 11. Circuito elemental que implementa la compuerta lógica NAND.**



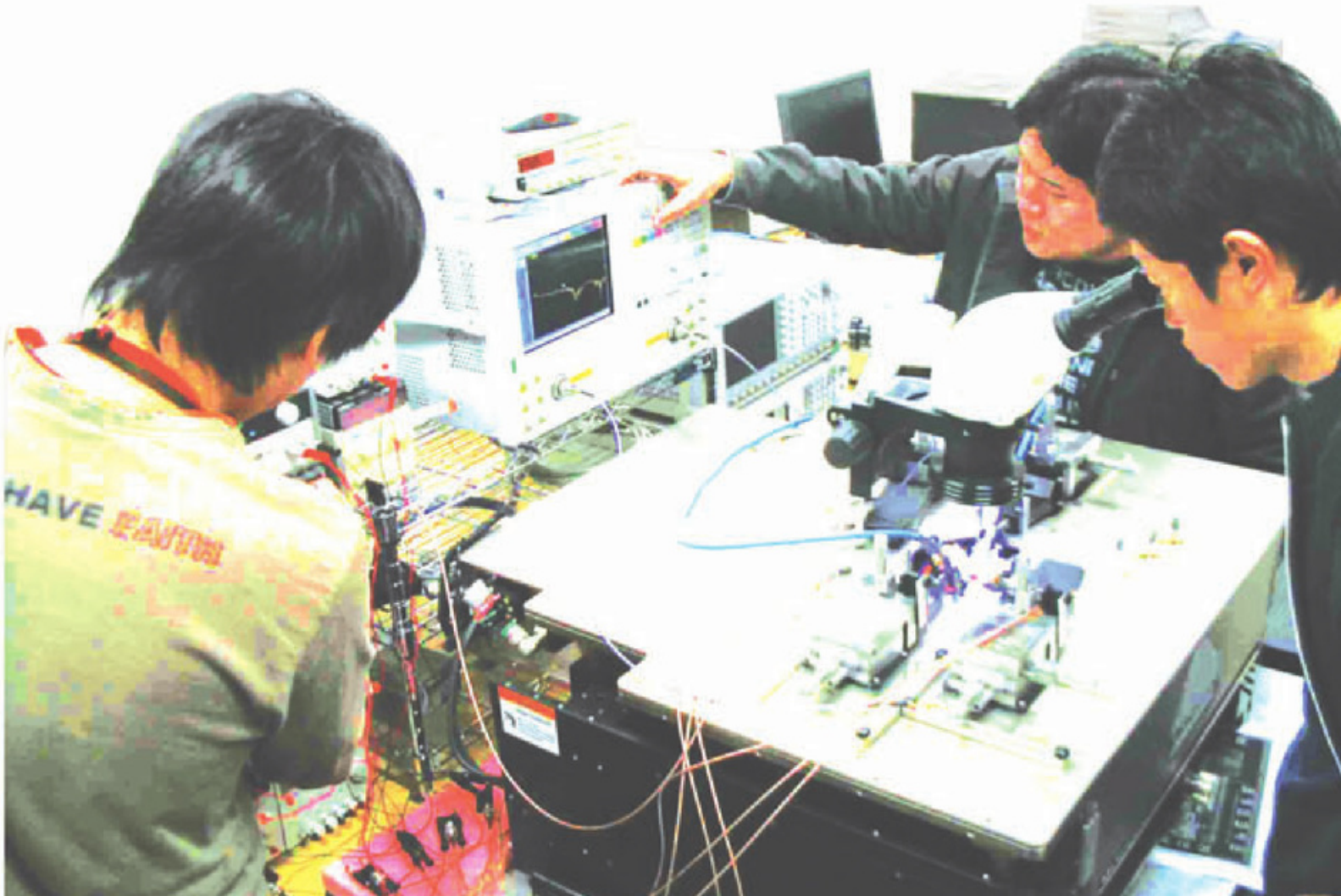
vez, basado en un transistor PNP) en cascada con la compuerta **AND** vista en la **Figura 7**.

Buffers de tres estados

El diseño electrónico de las salidas de algunos dispositivos **CMOS** o **TTL** puede estar en un estado lógico **0**, 1 o de **alta impedancia**, también llamado **Hi-Z**. En la representación gráfica de un BUFFER de tres estados (**Figura 12**), se distingue con claridad las entradas y las salidas. Adicionalmente, se representa una señal que actúa sobre el BUFFER, denominada **habilitación de salida**, que puede ser activo alto o bajo, dependiendo de si está presente o no el círculo que denota inversión. Cuando esta entrada está activa, el dispositivo se comporta como un BUFFER normal, mientras que si está negada, entonces la salida del BUFFER entra en un estado de alta impedancia y, funcionalmente, se comporta como si no estuviera allí. La utilidad de estas compuertas es que permiten a múltiples fuentes compartir una sola línea de comunicación, mientras que solo una de ellas transmite datos por vez. Es decir, cuando un dispositivo quiere colocar información en la línea, deberá salir de su estado **Hi-Z** y empezar la transmisión, pero antes de hacerlo, debemos asegurarnos de que los demás dispositivos en la línea ingresaron en su estado **Hi-Z**; de lo contrario, habrá colisión de datos.



> Figura 12. Representación de BUFFERS de tres estados, inversores y no inversores, cada uno con su respectiva habilitación activo alto o bajo.



ARITMÉTICA BINARIA

Suma, resta, multiplicación y división son operaciones aritméticas que solemos realizar con números decimales. A continuación, veremos cómo hacerlas empleando números binarios.

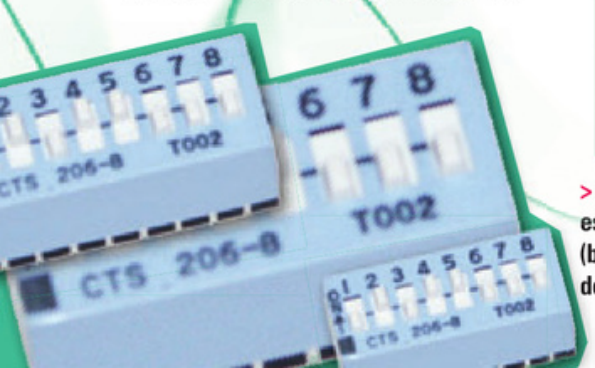
En el apartado anterior hicimos una primera aproximación a las operaciones con números binarios. A continuación, veremos las cuatro operaciones fundamentales: suma, resta, multiplicación y división. Al operar con números binarios, debemos tener en cuenta que si el resultado no está dentro del rango que va desde **0** hasta 2^n-1, siendo **n** la cantidad de bits, no cabrá dentro de la precisión que estamos usando.

Suma de números binarios

La suma se realiza bit a bit. Por ejemplo, para sumar dos números binarios, se procede sumando **dígito** a **dígito** y transportando el **acarreo** (si existe) al dígito de mayor peso, es decir, el inmediato izquierdo. Veamos un ejemplo para el caso de z:

```
  1010
+ 0100
------
  1110
```

> **Este resultado no posee acarreo (carry) final, con lo cual cabe dentro de la precisión que estamos usando.**



Podemos verificar el resultado de la operación anterior realizándola en decimal: vemos que **10 + 4 = 14**. Si hacemos la conversión de un sistema a otro, comprobaremos que ambos resultados coinciden. Veamos otro ejemplo:

```
  1110
+ 1000
------
 10110
```

> **Este resultado posee acarreo final, con lo cual no cabe dentro de la precisión que estamos usando.**

Resta de números binarios

La **resta** también se realiza bit a bit. Una forma sencilla de hacerla es sumarle al minuendo el complemento a 2 del sustraendo. Veamos un ejemplo para el caso de **4 bits**:

```
  1001
- 0100
------

Lo anterior es equivalente a

  1001
+ 1100
------
 10101
```

> **La presencia de acarreo final en este caso está indicando que no existió préstamo (borrow) en el resultado, con lo cual cabe dentro de la precisión que estamos usando.**

Multiplicación de números binarios

Para multiplicar dos números binarios, se procede de igual forma que en el sistema decimal: se multiplica el **multiplicando** por cada dígito del **multiplicador**, comenzando por el de menor peso, y finalmente se suman los resultados con su peso correspondiente. Una manera sencilla de hacerlo es tomando el **bit menos significativo del multiplicador**.
Si es **1**, tomamos el multiplicando como un primer resultado, y si es **0**, tomamos **0** como primer resultado. Luego, desplazamos el multiplicando un dígito hacia la izquierda (recordemos que desplazar una cifra binaria un lugar hacia la izquierda equivale a multiplicar por 2). A continuación, tomamos el segundo bit de la derecha y hacemos lo mismo; sumamos este nuevo resultado al anterior. Repetimos así esta operación hasta acabar con todos los dígitos del multiplicador.
Estas operaciones se simplifican con el uso de registros de desplazamiento, tema que veremos más adelante.
Veamos un ejemplo:

```
    1101
x    101
------
    1101
+  0000
+ 1101
------
 1000001
```

> **Multiplicación con números binarios.**

División de números binarios

Para la división, podemos aplicar un algoritmo sencillo. La operación comienza preparando una resta, en la que ubicamos como minuendo al dividendo, y como sustraendo, el divisor, pero colocado a la izquierda, de forma tal que el peso del bit más significativo del divisor coincida con el del bit más significativo del dividendo. Si vemos que en esa posición el minuendo es menor al sustraendo, no se realiza la resta, y colocamos un **0** como dígito más significativo del cociente. Si no, hacemos la resta y ponemos un **1** como dígito más significativo del cociente. En el siguiente paso, el resultado de la resta anterior será el nuevo minuendo (si no hubo resta, se mantendrá). Luego, el nuevo sustraendo será igual al anterior, pero desplazado un dígito hacia la derecha. Procedemos de la misma manera, anotando el nuevo resultado (si corresponde) y agregando un nuevo dígito al cociente (un dígito menos significativo que el anterior). Esta operación se lleva a cabo hasta que el nuevo sustraendo coincida con el dividendo. Veamos un ejemplo:

```
Dividendo: 10110
Divisor: 10

    10110
  - 10000
    00110
```

> **Se realizó una resta. El dígito más significativo del cociente es un 1.**

```
    00110
  - 00100
    00010
```

> **Se realizó una resta. El dígito inmediato derecho del cociente es un 1.**

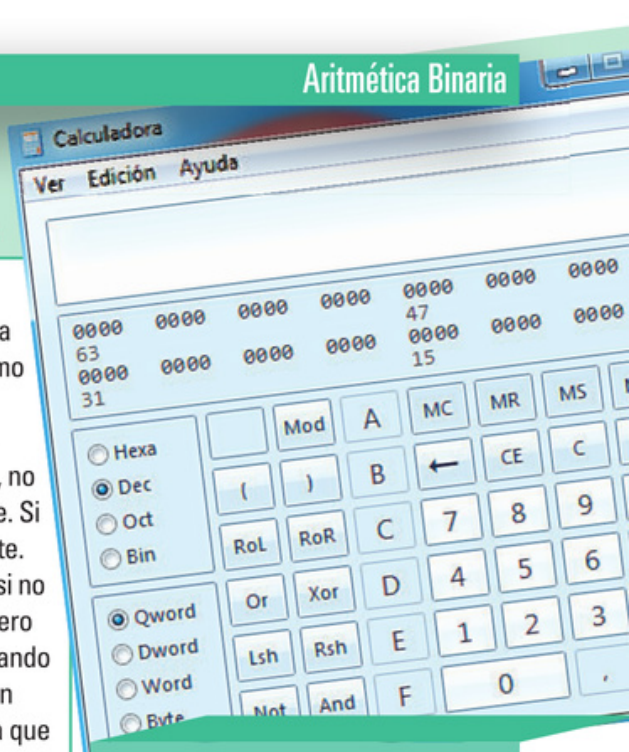
```
    00110
  - 01000
    00110
```

> **No se realizó la resta. El dígito inmediato derecho del cociente es un 0.**

```
    00010
  - 00010
        0
```

> **Se realizó una resta. El dígito menos significativo del cociente es un 1 y el resto, 0.**

Finalmente, en el ejemplo obtuvimos:
Cociente: **1011** Resto: **0**



> CON LA CALCULADORA DE WINDOWS

Para convertir un sistema de numeración en otro, primero seleccionamos el sistema de origen (Hexa, Dec, Oct o Bin), escribimos el número por convertir y, luego, hacemos clic sobre el sistema de numeración al que deseamos llevarlo. Tenemos que seleccionar el tamaño de presentación que queremos utilizar (Qword, Dword, Word o Byte). Esta calculadora no nos informa acerca de la existencia de acarreo.

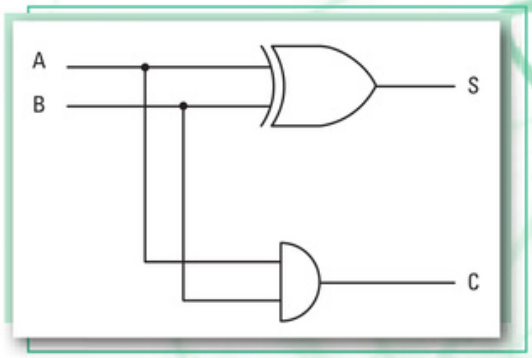
Circuito sumador binario de 1 bit con acarreo

Para realizar una suma binaria de 1 bit, debemos tener en cuenta el resultado para la suma de **1 + 1** (caso especial), en la cual se obtiene 10 (resultado 0 y acarreo 1). Este circuito constará de dos entradas que llamaremos **A** y **B**; y de dos salidas, las cuales serán **S** para el resultado (suma) y **C** para el acarreo (carry). Así, podemos confeccionar la **Tabla 1**:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

> Tabla 1

Vemos que la compuerta necesaria para obtener **S** es una **XOR**, y la requerida para obtener **C** es una **AND**. En ambas, ingresan las entradas **A** y **B**. Este sencillo circuito se denomina medio-sumador, y solo puede sumar cifras de un dígito. Si queremos sumar cifras de n bits, necesitamos un sumador completo, como se observa en la figura. ¿Cómo funciona?



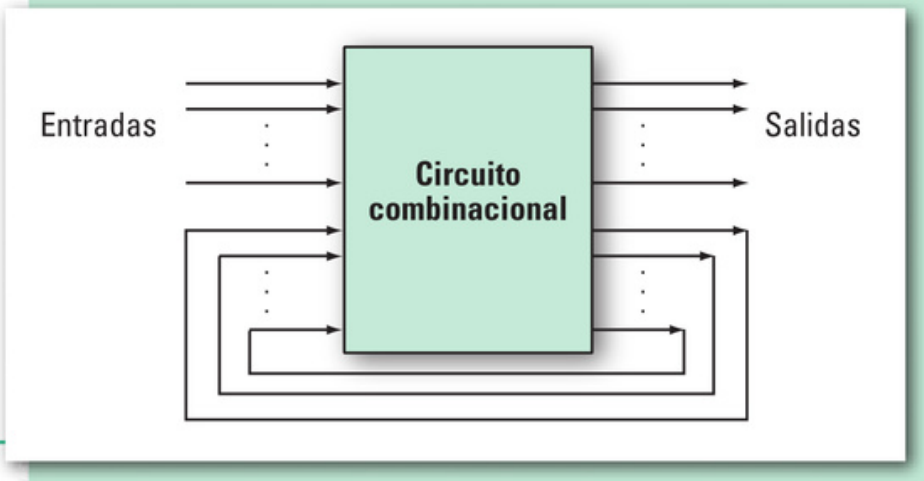
> **A la izquierda vemos un circuito medio-sumador binario de 1 bit con acarreo. A la derecha, un sumador completo de 1 bit con acarreo.**

Lógica combinacional y secuencial

Los circuitos lógicos pueden tener una o más entradas y una o más salidas, las cuales presentarán dos niveles de tensión posibles: el correspondiente para el **0 lógico** y el correspondiente para el **1 lógico**. Además, todo circuito lógico puede clasificarse en dos tipos: **combinacional** o **secuencial**.

Circuitos lógicos combinacionales

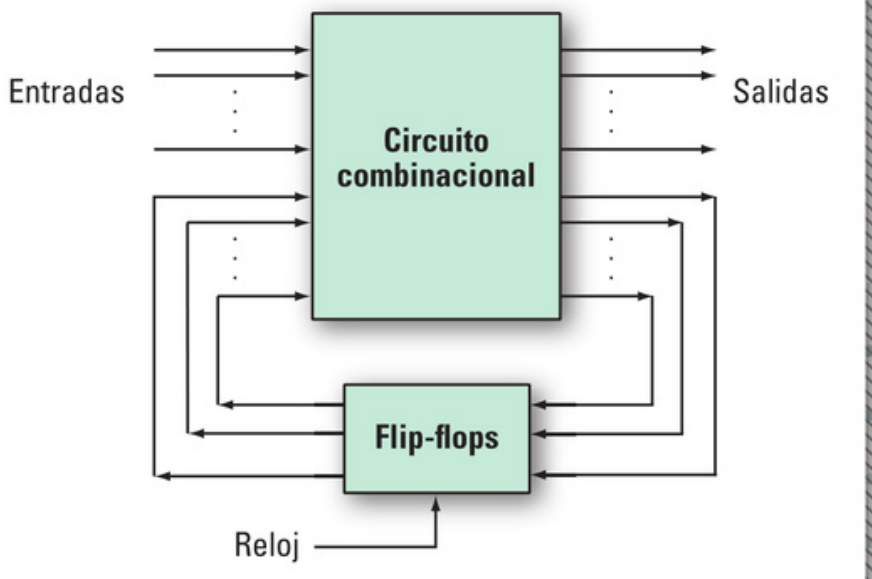
Ya hemos aprendido que, en los circuitos lógicos combinacionales, el estado de las salidas es función exclusiva del estado de las entradas en un instante determinado. También estudiamos las compuertas: **OR**, **AND**, **NOR**, **NAND**, **XOR**, **XNOR**, **inversor** y **buffer**. Pertenecen también a la categoría de circuitos lógicos combinacionales los dispositivos integrados, tales como: **comparadores de magnitud**, **sumadores**, **codificadores**, **decodificadores**, **multiplexores** y **demultiplexores**, entre otros.



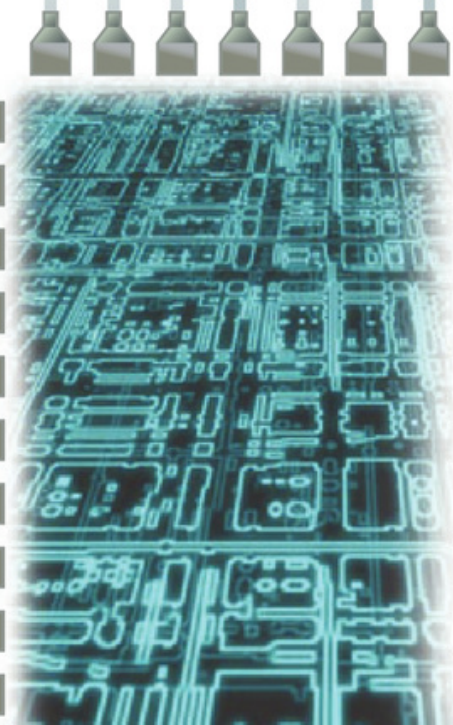
Un circuito secuencial asincrónico está compuesto por un circuito combinacional, que cuenta con salidas realimentadas hacia la entrada.

Circuitos lógicos secuenciales

En los circuitos lógicos **secuenciales**, el estado de las salidas es función no solo del estado de las entradas en ese mismo instante, sino también del estado previo de ellas, con lo cual almacenan (memorizan) los estados anteriores. Existen dos tipos de circuitos lógicos secuenciales. Por un lado, los **asincrónicos**, que poseen salidas realimentadas hacia las entradas. Esto permite que los estados de las salidas sean función no solo de los estados de las entradas en ese instante, sino también de los estados anteriores de estas. Por otro lado, están los **sincrónicos**, en los cuales los cambios de estado se producen en sincronismo mediante una señal de reloj (**clock**).

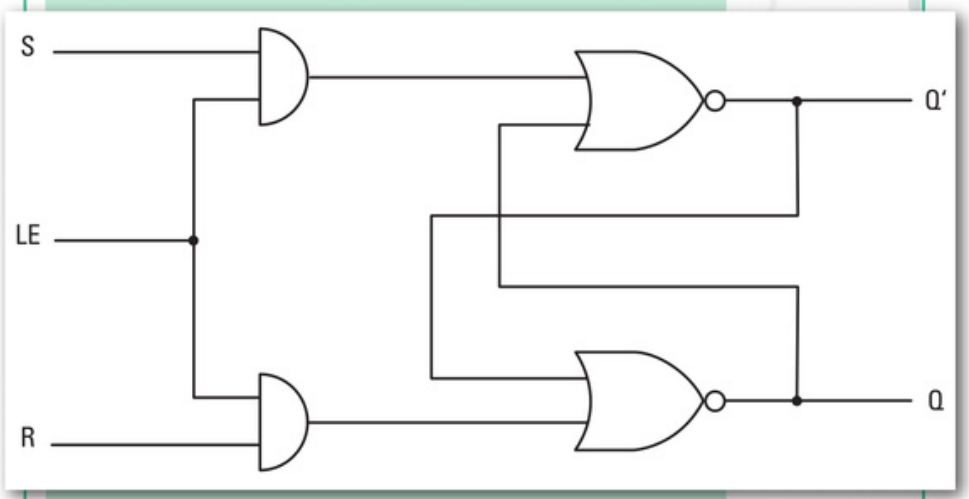


En un circuito secuencial sincrónico los cambios de estado se producen en sincronismo mediante la señal de reloj.



Circuitos con realimentación

Analizaremos los diferentes circuitos con realimentación; en particular, veremos los **biestables** o **flip-flops**. Hay flip-flops en los que los cambios de estado se producen en forma sincrónica con una señal de reloj. Existen los disparados por nivel y los disparados por flanco. Los segundos pueden dividirse en dos tipos: por flanco positivo y por flanco negativo.



> **Al poner un 1 en la entrada S, la salida Q cambiará a 1, mantendrá dicho estado solo hasta que se le aplique un 1 a la entrada R, con lo cual la salida Q cambiará a estado 0.**

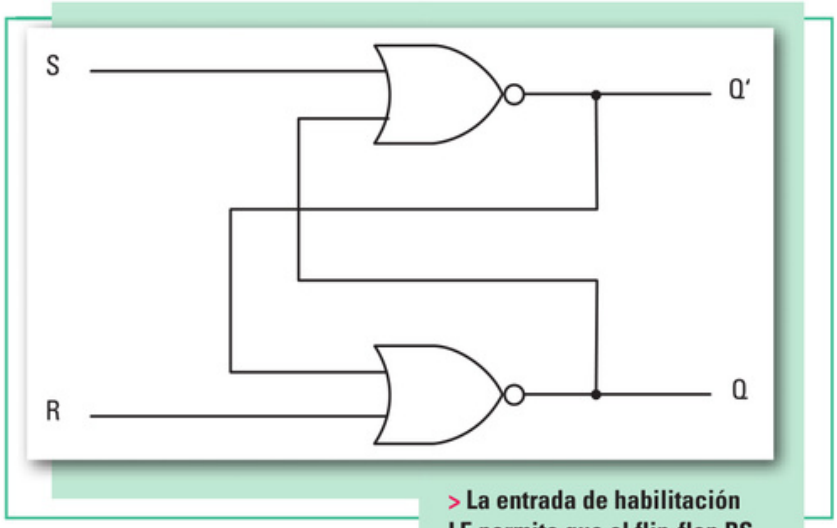
En los flip-flops sincrónicos, los cambios de estado se producen en forma sincrónica con una señal de reloj.

Flip-flop RS (o SR)

Este circuito secuencial posee dos entradas llamadas **S** (set) y **R** (reset); y dos salidas llamadas **Q** y **Q**, siendo **Q** el complemento o negación lógica de **Q**. Al poner un **1** a la entrada **S**, el estado de la salida **Q** también se pondrá en **1**. En cambio, al poner un **1** en la entrada **R**, el estado de esta salida se pondrá en **0**. Cuando ambas entradas tienen nivel lógico **0**, el estado de las salidas podrá ser **0** o **1**, dependiendo del estado anterior de las entradas. En caso de que el estado de **S** y **R** sea **1**, producirá una indeterminación en la salida, con lo cual queda como estado no deseado. Veamos la tabla característica de este flip-flop (**Tabla 2**).

S	R	Q
0	0	no varía
0	1	0
1	0	1
1	1	no deseado

> Tabla 2

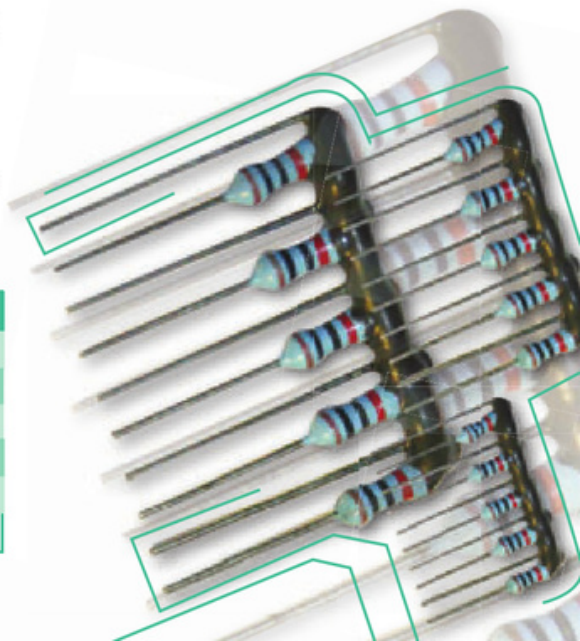


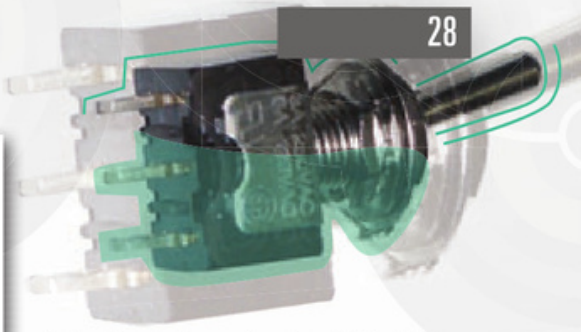
> **La entrada de habilitación LE permite que el flip-flop RS funcione normalmente, o que mantenga el último valor de su salida sin importar el estado de las entradas.**

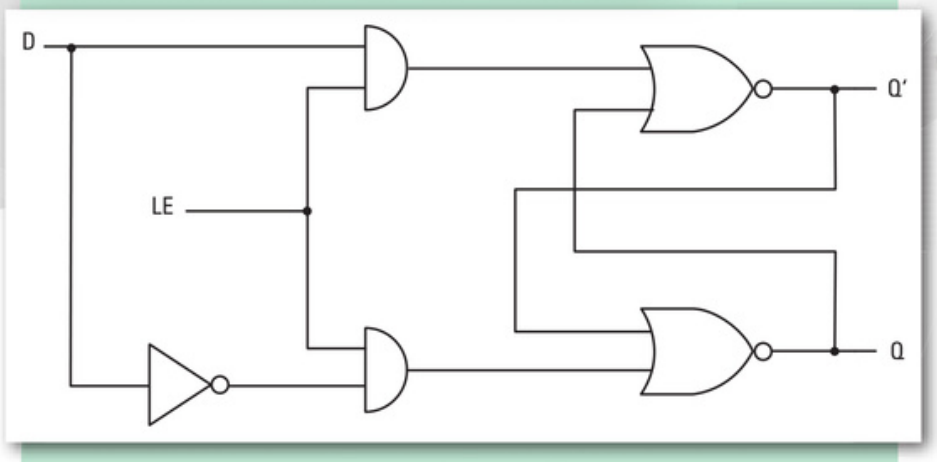
Si modificamos un poco este circuito agregándole dos compuertas **AND**, podemos añadirle una entrada de habilitación **LE** (Latch Enable).
De este modo, cuando esta entrada **LE** esté en estado lógico **1**, el flip-flop funcionará normalmente. Cuando esté en estado **0**, el flip-flop mantendrá el último valor de su salida, sin importar los de las entradas **S** y **R** aplicados posteriormente. La tabla característica de este flip-flop con el agregado de la entrada de habilitación es la **Tabla 3**.

LE	S	R	Q
0	X	X	no varía
1	0	0	no varía
1	0	1	0
1	1	0	1
1	1	1	no deseado

> Tabla 3







> Este circuito es, básicamente, un flip-flop RS, con una única entrada D conectada directamente a la entrada S, y a través de una compuerta inversora, a la entrada R.

Latch (Flip-flop D asincrónico)

Este circuito secuencial permite que la salida **Q** siga los cambios de la entrada **D**, siempre y cuando esté habilitada la entrada **LE**. La tabla característica de este flip-flop es la **Tabla 4**:

LE	D	Q
0	X	no varía
1	0	0
1	1	1

> Tabla 4

Flip-flop RS sincrónico

El funcionamiento de este **flip-flop RS** es análogo al del asincrónico, con la diferencia de que los cambios de estado se producen en forma sincrónica con una señal de reloj, que es ingresada por la entrada **CLK** del flip-flop. La tabla característica de este flip-flop es la **Tabla 5**:

S	R	Q*
0	0	Q
0	1	0
1	0	1
1	1	X

>Tabla 5

Se denomina **Q*** al valor que tomará la salida del flip-flop en el siguiente flanco correspondiente de la señal de reloj. Aquí también vemos que si el estado de las entradas **S** y **R** es **1**, se produce una indeterminación en la salida, con lo cual, al igual que en el **flip-flop RS asincrónico**, queda como estado no deseado.

Además de las entradas **S**, **R** y **CLK**, existen dos más llamadas **PRE** (preset) y **CLR** (clear). La primera fuerza la salida **Q** al estado **1**, mientras que la segunda la fuerza al estado **0**.

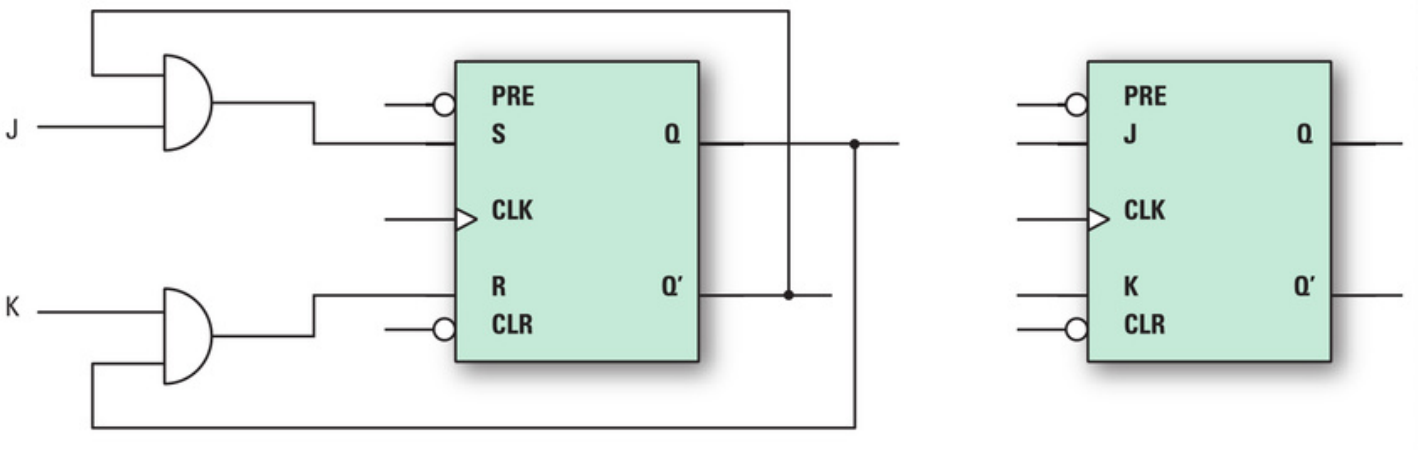
Flip-flop JK

Se comporta de manera similar al **RS**, con la excepción de que en este último se permite que ambas entradas, **J** y **K**, estén en estado **1** al mismo tiempo. La tabla característica de este flip-flop es la **Tabla 6**.

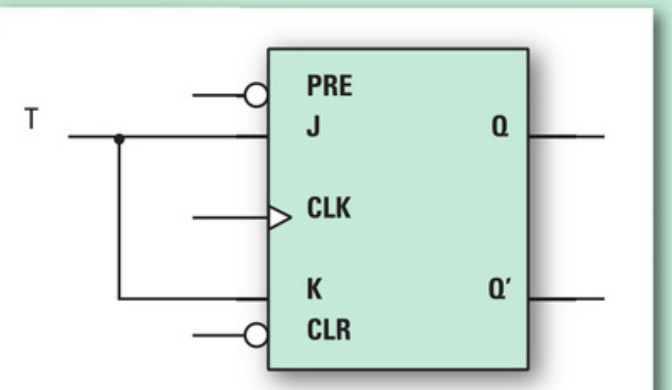
J	K	Q*
0	0	Q
0	1	0
1	0	1
1	1	\Q (Q negado)

> Tabla 6

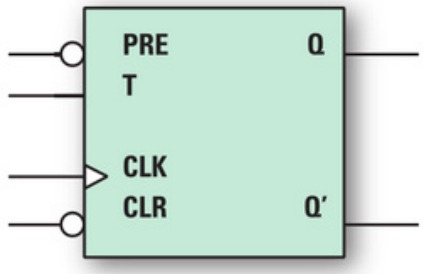
Sobre la base de esta tabla, podemos ver que el **flip-flop JK** cubre todas las características del **RS**. Cuando las entradas **J** y **K** están en estado **1** al mismo tiempo, el estado futuro **Q*** será el opuesto al anterior, con lo cual será **\Q (Q negado)**. Las entradas de **CLK**, **PRE** y **CLR** son las mismas que para el **flip-flop RS**.



> A partir de un flip-flop RS, podemos obtener uno JK, simplemente agregando dos compuertas AND y conectándolas como muestra la figura.



> Uniendo ambas entradas de un flip-flop JK, obtenemos un flip-flop T. Cuenta también con las entradas PRE y CLR para forzar el estado de la salida.

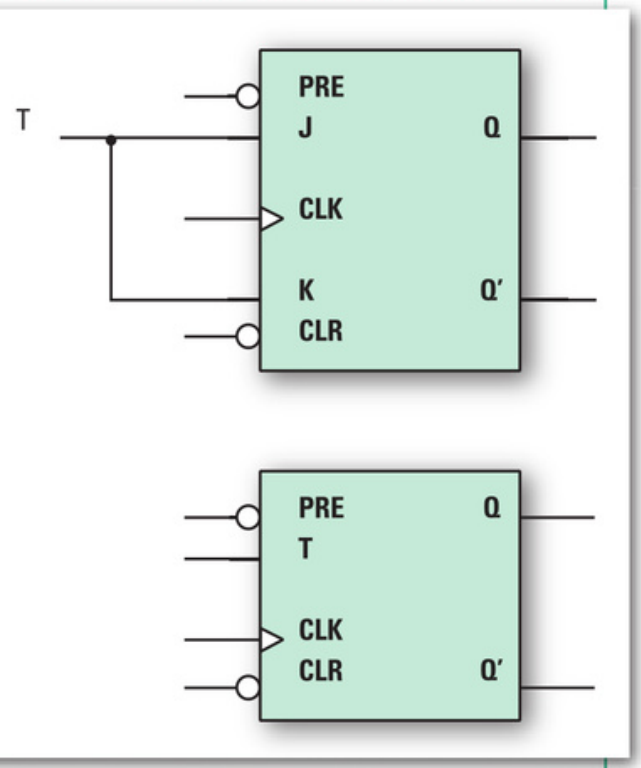


Flip-flop T (toggle)

Fácilmente, podemos obtener un **flip-flop T** con solo unir las entradas **J** y **K** de un **flip-flop JK**, y llamando **T** a esta única entrada. La tabla característica de este flip-flop es la **Tabla 7**.

T	Q*
0	Q
1	\Q (Qnegado)

> Tabla 7



> Este es un flip-flop RS, con una entrada D conectada a la entrada S, y a través de una compuerta inversora, a la entrada R.

El flip-flop T se utiliza como divisor de frecuencia. El D, como demora de un clock.

Sobre la base de esta tabla, podemos describir el funcionamiento del **flip-flop T** de la siguiente manera: cuando la entrada **T** tiene nivel lógico **0**, el estado de la salida **Q**, en el próximo flanco correspondiente de la señal de reloj –el estado de **Q***–, no se modifica. En cambio, si la entrada **T** tiene nivel lógico **1**, el estado de la salida **Q** en el próximo flanco correspondiente de la señal de reloj se invierte.

Una de las aplicaciones que puede tener el **flip-flop T** es como divisor de frecuencia, ya que si aplicamos un **1** a la entrada **T**, la salida **Q** irá alternando su estado en cada flanco correspondiente de la señal de reloj. Con esto, estaríamos dividiendo la frecuencia del reloj a la mitad.

Flip-flop D (delay)

Este flip-flop permite que la salida **Q** siga los cambios de la entrada **D**, siempre en cada flanco correspondiente de la señal de reloj. La tabla característica de este flip-flop es la **Tabla 8**.

Se logra a partir de un flip-flop **JK** uniendo sus entradas con un inversor, de modo que **J = D y K = \D (D negado)**.

A su vez, si en un flip-flop **D** se conecta **D = \Q (Q negado)**, se lo transforma en un flip-flop **T** ■

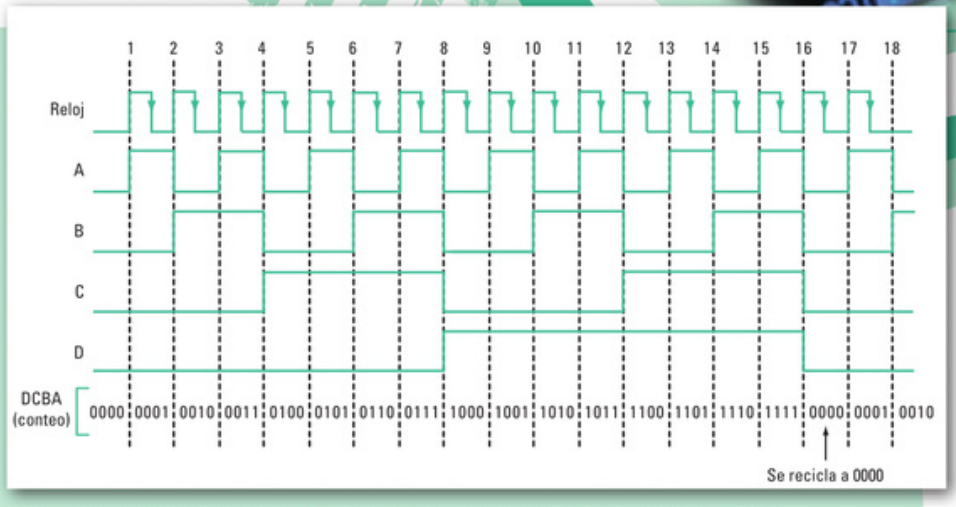
D	Q*
0	0
1	1

> Tabla 8

CIRCUITOS SECUENCIALES

Ya sabemos qué son los flip-flops y cómo funcionan. Ahora veremos de qué manera se utilizan para constituir la base de los circuitos secuenciales, contadores sincrónicos y asincrónicos.

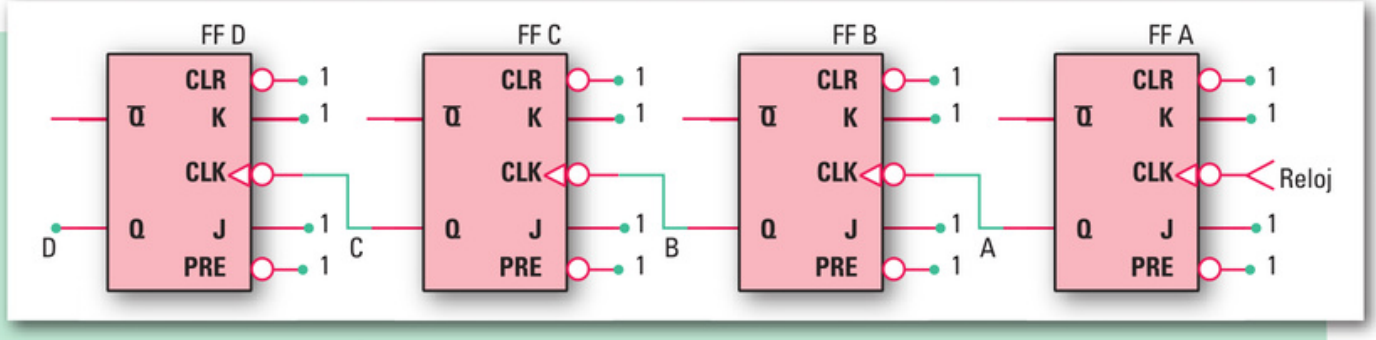
Los contadores son circuitos lógicos secuenciales que llevan la cuenta de una serie de pulsos de entrada. Se forman a partir de **flip-flops** conectados entre sí y combinados muchas veces con compuertas lógicas. En los sistemas digitales, se utilizan variaciones de estos circuitos en forma de circuitos integrados. Podemos clasificar los contadores como **binarios** o **no binarios** y **síncronos/sincrónicos** o **asíncronos/asincrónicos**. Aquí nos ocuparemos de los binarios, llamados así porque el conteo se realiza en código binario; además, veremos los del tipo síncrono y asíncrono. Analizaremos ahora la estructura de ambos.



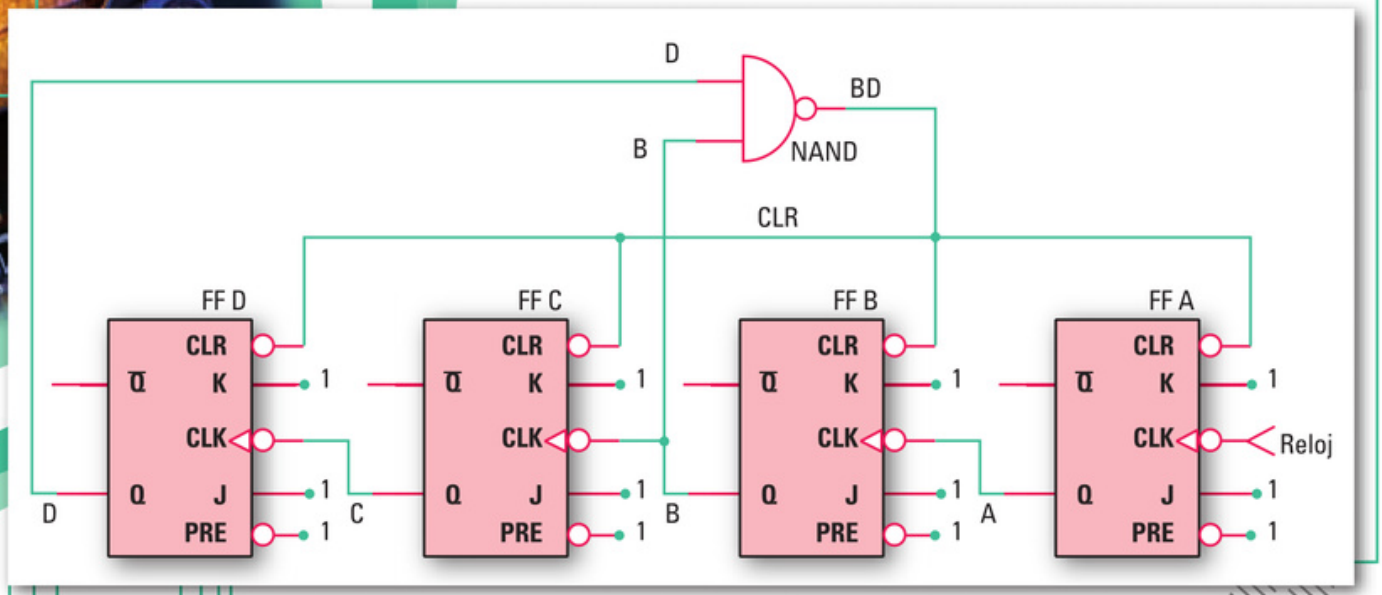
> **Las formas de onda de salida de cada flip-flop y de la señal de reloj. Vemos una secuencia de conteo de 0000 a 1111. En la decimosexta transición negativa de reloj se inicia un nuevo ciclo de conteo.**

Contador binario asíncrono

Para entender el funcionamiento de este tipo de contador veamos el circuito **Contador asíncrono de 4 bits**. Se trata de un contador binario de 4 bits formado por cuatro flip-flops **JK** conectados en serie. Los pulsos de reloj se aplican únicamente a la entrada de reloj (**CLK**) del primer flip-flop. Como las entradas **JK** del primer flip-flop (**flip flop A**) tienen un valor igual a **1** (nivel alto), cada vez que los pulsos de reloj hagan una transición negativa (de 1 a 0), el flip-flop pasará a su estado opuesto. Este modo de funcionamiento corresponde al de un **flip-flop T** (toggle). La salida normal del **flip-flop A** es la entrada de CLK para el **flip-flop B**. Por lo tanto, el flip-flop B cambiará su estado cada vez que la salida de A pase de **1** a **0**. De esta misma manera, el **flip-flop C** cambiará de estado cuando **B** pase de **1** a **0**, y así sucesivamente. Si ahora observamos las señales del circuito, notaremos que las salidas D, C, B y A del flip-flop representan un número binario de 4 bits.



> **Contador asíncrono de 4 bits. Se lo llama de este modo porque los flip-flops no modifican su estado en sincronía exacta con los pulsos de reloj.**



Un contador asíncrono de módulo 10. La acción de la compuerta NAND permite omitir estados en la secuencia. Podríamos implementarlo utilizando un integrado 74LS293 TTL.

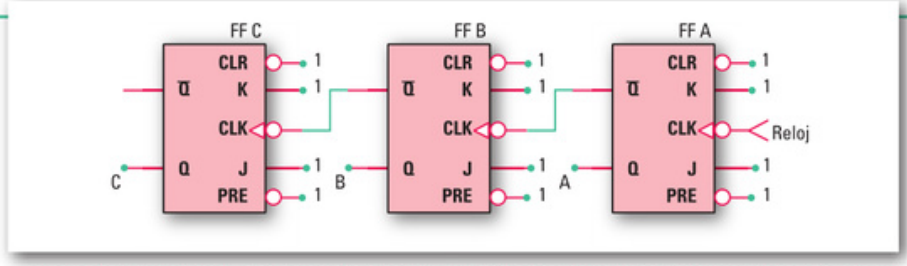
En el contador asíncrono, cada salida del flip-flop excita la entrada CLK del siguiente flip-flop. Por lo tanto, es un contador **asíncrono**, porque los flip-flops no modifican su estado en sincronía exacta con los pulsos de reloj. Concluimos, entonces, que hay un retardo entre las respuestas de los flip-flops, generalmente, de entre 5 y 20 **ns** (nanosegundos) por cada uno.

Los contadores son circuitos lógicos secuenciales que llevan la cuenta de una serie de pulsos de entrada.

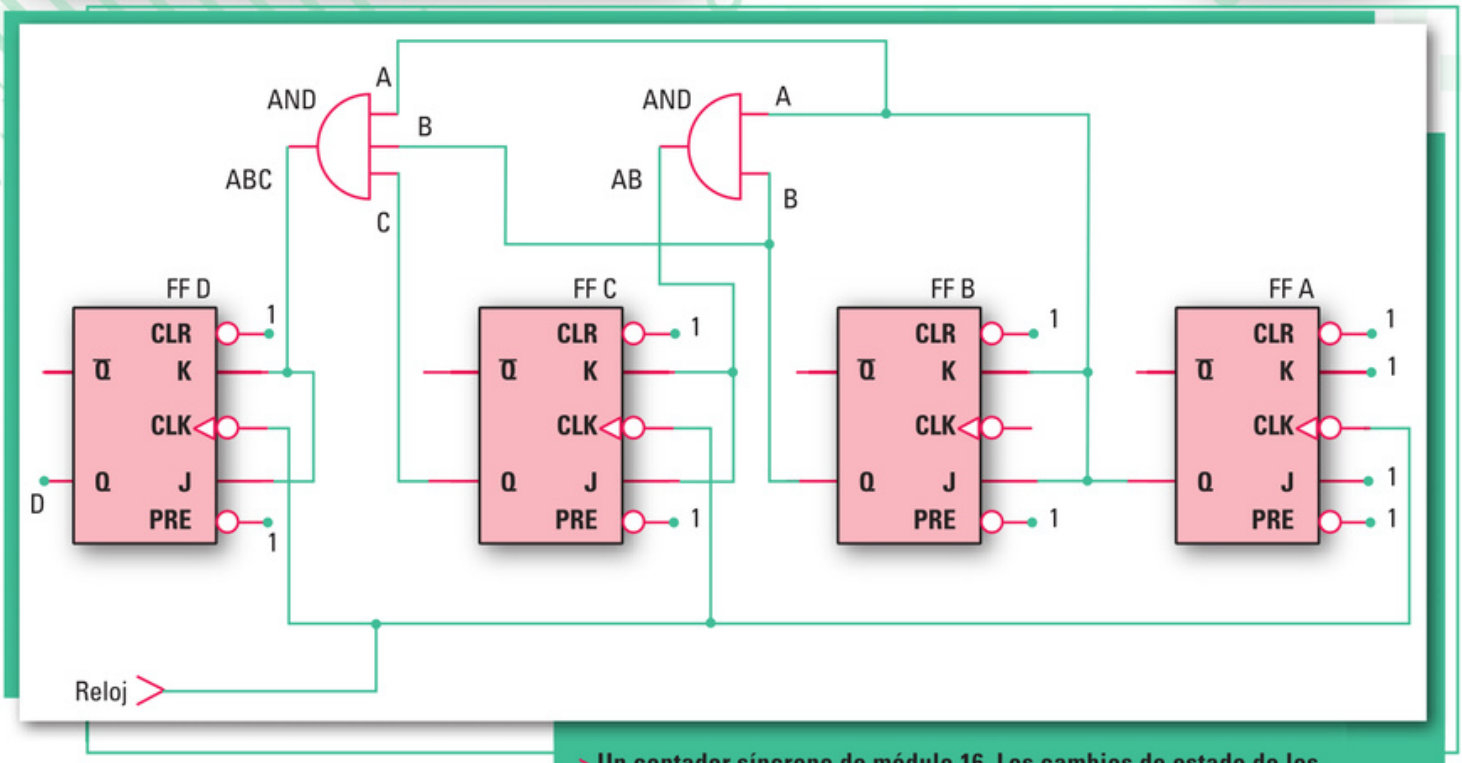
Módulo de un contador

El contador que acabamos de ver tiene 16 estados diferentes (0000 a 1111); su módulo es 16 (MOD-16). Si queremos aumentarlo, simplemente agregamos flip-flops. Entonces, **MOD=2^N**, donde **N** es el número de flip-flops conectados. Un contador actúa como un divisor de frecuencia, ya que la forma de onda de salida del último flip-flop tendrá una frecuencia que será igual a la frecuencia de entrada de reloj dividida por el módulo del contador.
El contador visto anteriormente posee un valor de módulo sujeto a la expresión **2^N**. Por ejemplo, si deseamos construir un contador con módulo igual a **10**, que no es una potencia de **2**, debemos buscar la manera de omitir estados en la secuencia de conteo. Para hacerlo, se utilizan compuertas lógicas.

Tomamos un contador de 4 bits, es decir, de cuatro flip-flops. Este sería, entonces, de módulo **16** si omitimos la compuerta **NAND**.
Si ahora tomamos en cuenta su función, vemos que cuando sus entradas **B** y **D** sean iguales a **1**, su salida será igual a **0** y, al estar conectada a la entrada **CLR** de los flip-flops, estos se borrarán y el contador pasará al estado **0000**. Esto ocurrirá justamente cuando el contador pase del estado **1001** al estado **1011** en la transición negativa del décimo pulso de entrada. El contador pasa por diez estados diferentes antes de reciclar su cuenta; se trata de un contador de módulo **10**.



Hasta ahora analizamos contadores de 0 hacia arriba (ascendentes). Para un contador descendente, los flip-flops se conectan a través de su salida negada. Vemos aquí un contador de módulo 8.



> Un contador síncrono de módulo 16. Los cambios de estado de los flip-flops están sincronizados con la entrada de reloj. Un circuito integrado comercial de este tipo es el 74HC163 CMOS.

Contador binario síncrono

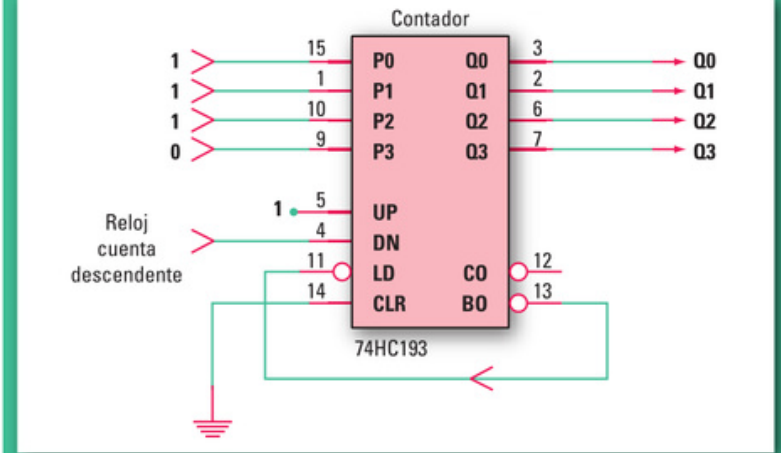
El contador asíncrono tiene el inconveniente de poseer retardos de propagación en su funcionamiento, por lo que no recomendamos su uso a frecuencias altas. Podemos superar este inconveniente utilizando contadores **síncronos**. Veamos el caso de un **contador síncrono de módulo 16**. Para un conteo adecuado, solo los flip-flops que deben cambiar de estado necesitan tener sus entradas **J** y **K** iguales a **1** cuando ocurra una transición negativa en el pulso de reloj. El flip-flop **A** tiene sus entradas permanentemente en **1**, por lo que cambia su estado con cada transición negativa de la entrada de reloj.

El **flip-flop B** debe cambiar estados en cada transición negativa de reloj que ocurra mientras **A = 1**. Esto se logra conectando la salida **A** a las entradas **J** y **K** del flip-flop B. El **flip-flop C** debe cambiar estados en cada transición negativa de reloj que ocurra mientras **A = B = 1**.

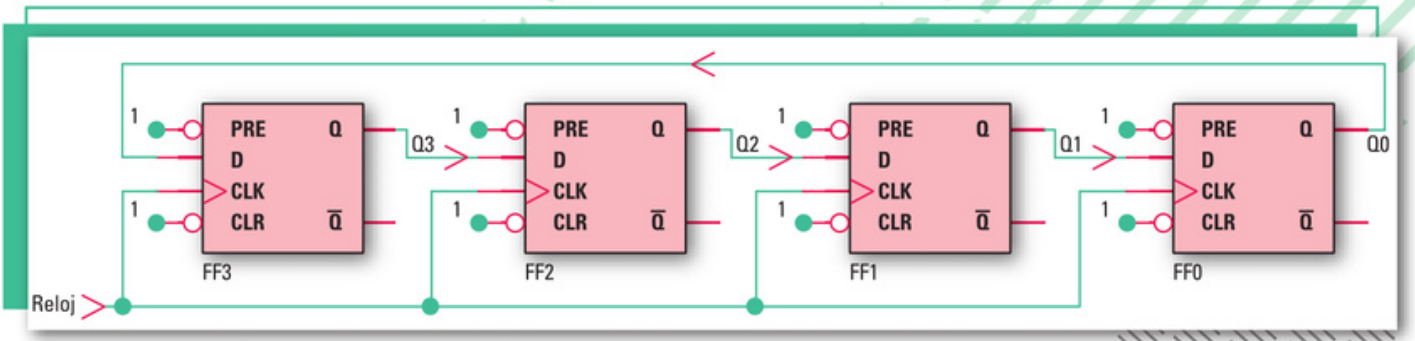
De la misma forma, el **flip-flop D** tiene que cambiar estados en cada transición negativa de reloj que ocurra mientras **A = B = C = 1**.

Para realizar todas estas operaciones se utilizan compuertas **AND**.

La mayoría de los circuitos integrados comerciales que funcionan como contadores síncronos se diseñan de modo tal que se pueda preestablecer el valor inicial del conteo de forma **asíncrona** (independiente de la señal de reloj) o de forma **síncrona** (en la transición activa de la señal de reloj). El contador posee entradas exclusivas para el preestablecimiento que se realiza de manera paralela. Además, por lo general tiene una entrada de control que establece si la cuenta será ascendente o descendente, y salidas de acarreo y préstamo para conectar más de un contador y aumentar el módulo.



> Un contador síncrono comercial, el 74HC193, en este caso usado como contador descendente de módulo 5 con sus entradas de carga paralela permanentemente conectadas a 0101 (5 en decimal).



Flip-flops del tipo D. Un contador en anillo de 4 bits formado por flip-flops del tipo D. La señal de reloj se aplica simultáneamente a todos los flip-flops, por lo que es un contador síncrono.

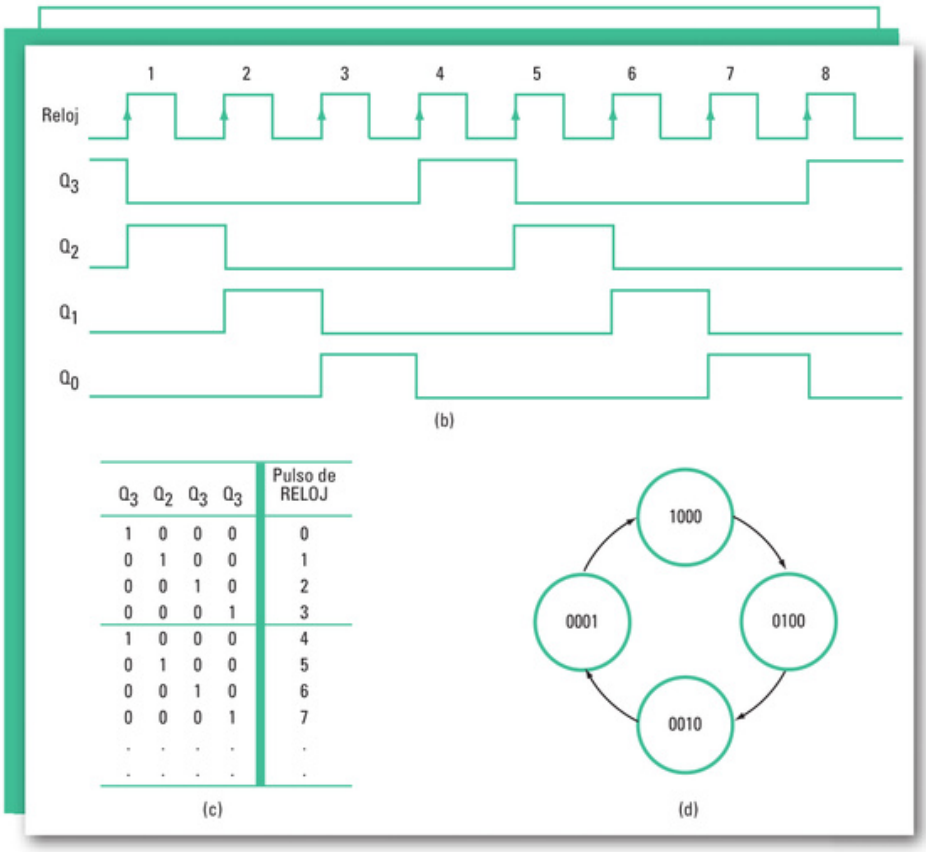
Contador en anillo y contador Johnson

Analizaremos en este apartado los contadores que emplean realimentación en su funcionamiento, ya que la salida del último flip-flop se conecta a la entrada del primero.

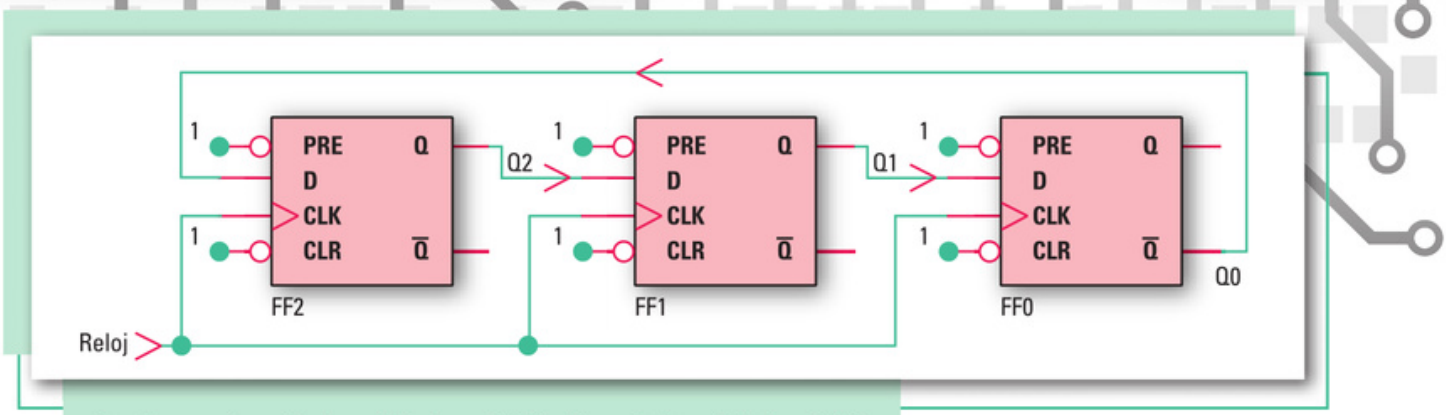
Contador en anillo

Su estructura se basa en flip-flops de tipo **D**, como se observa en la imagen correspondiente. La salida normal del último flip-flop se conecta directamente a la entrada **D** del primero. De esta manera, la información se desplaza de izquierda a derecha, y de regreso, de **Q0** a **Q3**. Un valor lógico **1** se hace circular por todo el arreglo de flip-flops cuando se aplican pulsos de reloj. Por esta razón se lo llama **contador en anillo**. Tomemos como ejemplo un contador en anillo de 4 bits. Si suponemos un estado inicial de **Q3 = 1**, **Q2 = Q1 = Q0 = 0**, vemos en las formas de onda que, después del primer pulso de reloj, el **1** se desplaza de **Q3** a **Q2**, y el estado resultante es **0100**. Con cada pulso, el **1** se desplaza, y en el cuarto pulso, se vuelve al estado inicial. Es un contador de módulo 4 porque atraviesa por cuatro estados diferentes antes de repetir la cuenta. El módulo del contador de este tipo es igual a la cantidad de flip-flops que utiliza; por lo tanto, es posible construir contadores en anillo para cualquier valor de módulo. Las formas de onda de los flip-flops están desplazadas un período de reloj una con respecto a la otra, y en cada una de ellas el estado lógico **1** tiene una duración de un período de reloj.

El contador en anillo necesita iniciar su secuencia de conteo con solo un flip-flop en estado lógico **1** y todos los demás, en **0**. Para esto, se aplica un pulso momentáneo a la entrada asíncrona **PRESET** de uno de los flip-flops, y a la entrada **CLR** de todos los otros flip-flops. También pueden utilizarse circuitos externos para asegurar este inicio.



Las formas de onda de salida de cada flip-flop y de la señal de reloj, la tabla de secuencia y el diagrama de estados para un contador en anillo. Como el conteo tiene cuatro estados diferentes, su módulo es 4.



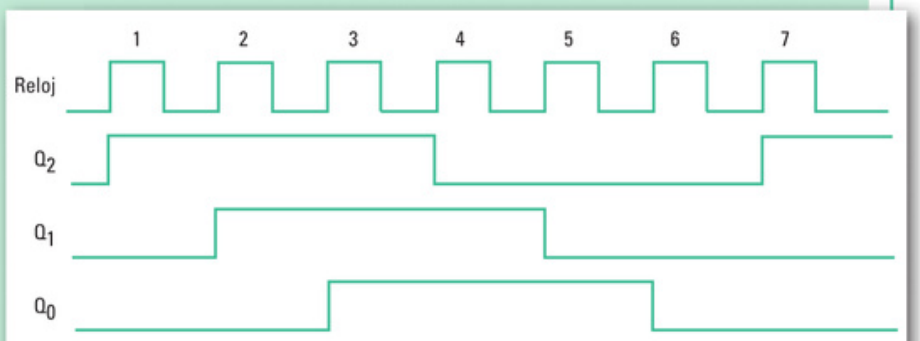
> **Las formas de onda de salida de cada flip-flop y de la señal de reloj, la tabla de secuencia y el diagrama de estados para un contador Johnson.**

Contador Johnson

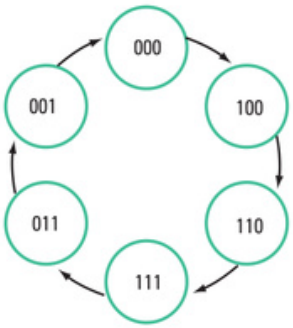
Para realizar este contador, debemos hacer una pequeña modificación al contador en anillo básico. En este caso, la salida invertida del último flip-flop se conecta a la entrada del primero. Analicemos un contador **Johnson de 3 bits**. Existe una realimentación debido a que la salida invertida de **Q0** se conecta a la entrada **D** de **Q2**.

Esta configuración hace que el nivel inverso del almacenado en **Q0** se transfiera a **Q2** en el pulso de reloj. Observando las formas de onda y la tabla de secuencia, deducimos que este contador tiene seis estados diferentes antes de repetir la secuencia. Tarda **3 pulsos** de reloj en propagar el **1** lógico hacia abajo en el arreglo de flip-flops, y otros **3 pulsos** de reloj en regresar

En un contador Johnson la salida invertida del último flip-flop se conecta a la entrada D del primero.



Q_2	Q_1	Q_0	Pulso de RELOJ
0	0	0	0
1	0	0	1
1	1	0	2
1	1	1	3
0	1	1	4
0	0	1	5
0	0	0	6
1	0	0	7
1	1	0	8
.	.	.	.
.	.	.	.
.	.	.	.



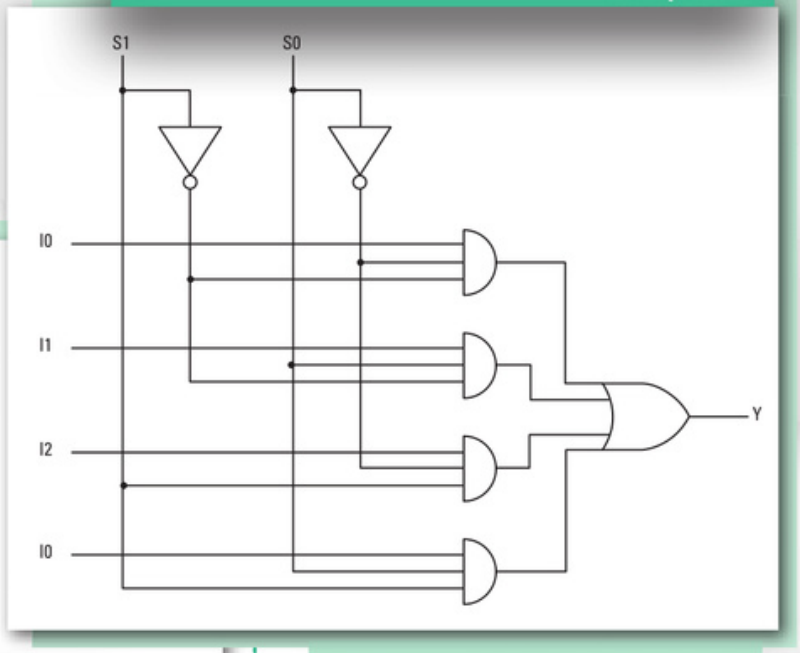
todos los flip-flops al estado **nulo**. Concluimos que el valor del módulo de un contador Johnson es igual al doble del número de flip-flops. La forma de onda de cada flip-flop es cuadrada (ciclo de trabajo del 50%) y de una frecuencia igual a la frecuencia de entrada dividida por el módulo del contador. Además, las formas de onda de los flip-flops se desplazan un período de reloj una con respecto a la otra.

Una ventaja del contador Johnson respecto del contador en anillo es que no necesita una rutina de inicialización, siempre y cuando el estado inicial de todos los flip-flops sea nulo.

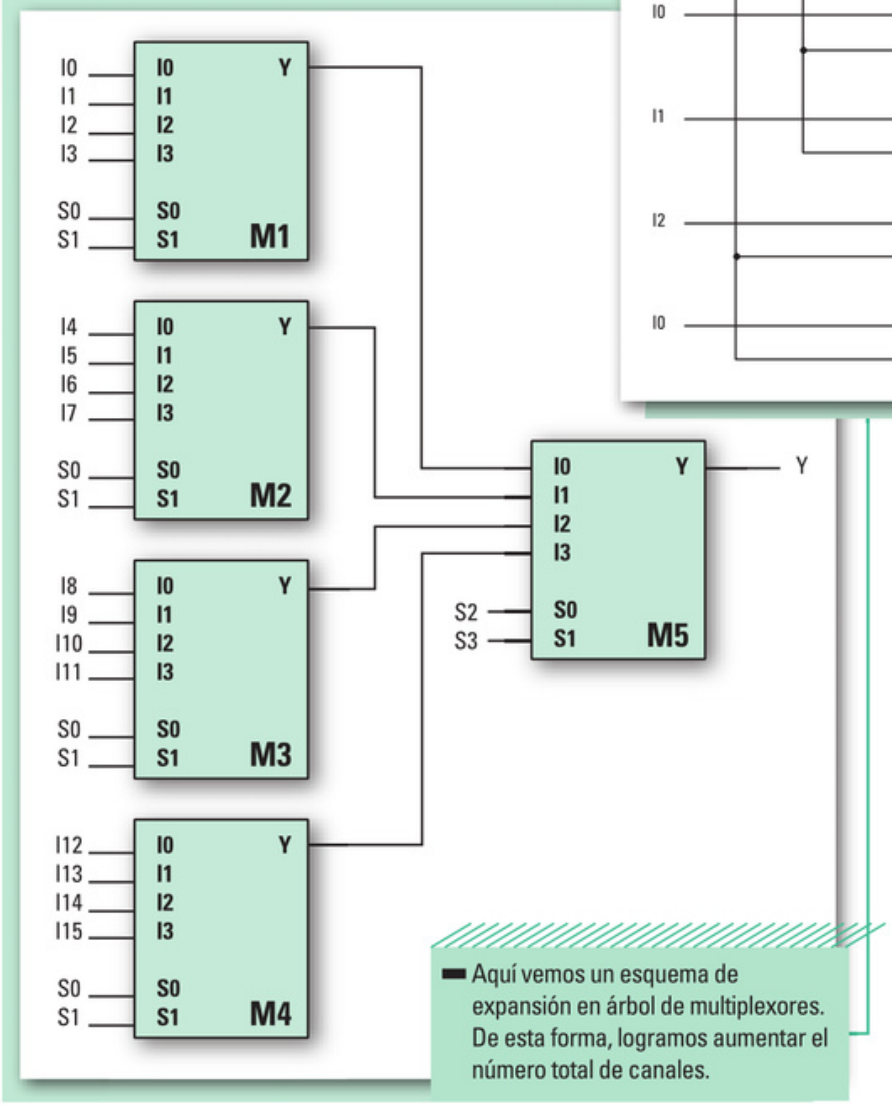
Los códigos de Johnson son la base de los sistemas de comunicación con corrección de errores **FEC** (*Forward Error Correction*). Su potencia radica en que solo un bit cambia a la vez, lo que permite detectar cambios por efectos del ruido ■

> **Un contador Johnson de 3 bits. Es una variante del contador en anillo, y también se lo conoce como contador en anillo trenzado.**

MULTIPLEXORES



Podemos ver el diagrama básico de un multiplexor de cuatro canales. La salida deseada se controla mediante las dos entradas de control.



Aquí vemos un esquema de expansión en árbol de multiplexores. De esta forma, logramos aumentar el número total de canales.

Multiplexores

A continuación, explicaremos qué son y para qué sirven los multiplexores, y veremos también la forma de realizar expansiones con estos dispositivos.

Los multiplexores son dispositivos que nos permiten mostrar una única señal en su salida, entre varias señales de entrada. La selección se realiza mediante una señal de control.

Teoría de funcionamiento

En electrónica digital, los multiplexores son dispositivos que poseen 2^n entradas (**I**), una sola salida (**Y**) y n entradas de control (**S**). Así, por ejemplo, un multiplexor de cuatro canales (4 entradas) tendrá dos entradas de control (**S0** y **S1**). El valor de la salida corresponderá al valor de la entrada seleccionada mediante las entradas de control. En la **Tabla 9** podemos ver el ejemplo mencionado:

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

> Tabla 9

Expansión

La expansión de multiplexores nos permite obtener un nuevo multiplexor con mayor cantidad de canales. Esto se logra mediante un esquema de expansión en árbol. Supongamos que contamos con cinco multiplexores (M1, M2, M3, M4 y M5) de cuatro canales cada uno. Con ellos, podremos conseguir un multiplexor de 16 canales con sus cuatro entradas de control correspondientes, de la siguiente forma: conectamos las salidas **Y** de M1, M2, M3 y M4 a las entradas **I0**, I1, I2 e I3 de M5. Las entradas **I** de M1, M2, M3 y M4 (16 en total) serán las nuevas entradas **I**. La salida **Y** de M5 será la nueva salida **Y**. La nueva señal de control será de 4 bits, donde las dos entradas de control de M5 serán los dos bits más significativos (**S3** y **S2**), y las entradas de control del resto corresponderán a los dos bits menos significativos (**S1** y **S0**).

Demultiplexores

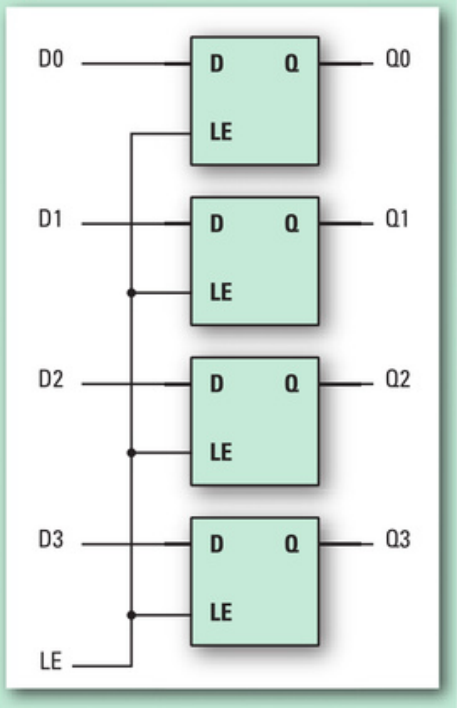
El funcionamiento de los **demultiplexores** es exactamente **inverso** al de los multiplexores. Son dispositivos que poseen una sola entrada, **2n** salidas y **n** entradas de control. De este modo, mediante las entradas de control, seleccionamos la salida por la cual se transferirá la señal que posee a la entrada ■

REGISTROS

En este apartado explicaremos qué son y para qué sirven estos dispositivos tan importantes utilizados en la electrónica digital, llamados registros.

Los registros son dispositivos lógicos compuestos, principalmente, por **biestables** o **flip-flops**, que permiten almacenar información a manera de bits. La cantidad de bits del registro queda determinada por la cantidad de biestables o flip-flops que posea. Los registros se clasifican según cómo se carguen los datos a la entrada y a la salida. Por un lado, están los que poseen carga paralela, tanto a la entrada como a la salida. Por otro lado, aquellos en que al menos la entrada o la salida es de carga serie. Estos últimos se denominan registros de desplazamiento.

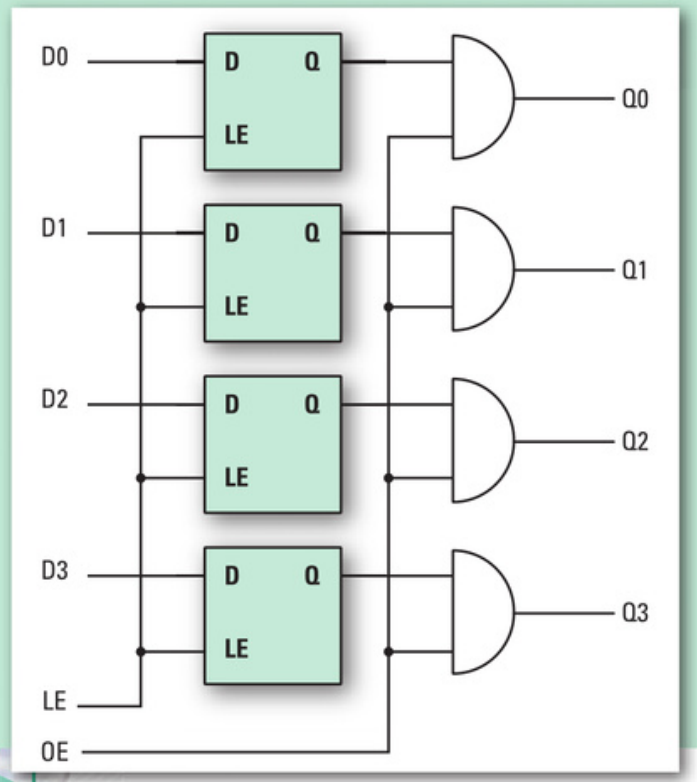
Al deshabilitar la entrada de habilitación general, la salida **Q** de cada biestable mantendrá el estado correspondiente, sin importar que se modifique el estado de la entrada **D**. En caso de que necesitemos colocar una habilitación para mostrar los datos a la salida, podemos hacerlo fácilmente agregando una compuerta **AND** de dos entradas para cada una de las salidas. Una de las entradas de cada compuerta **AND** será la salida **Q** del flip-flop **D** correspondiente. La segunda entrada de cada **AND** será la misma para todas, y corresponderá a la entrada **Output Enable** (OE).



> **Vemos un registro de entrada y salida paralelo de 4 bits. La entrada de habilitación general permite mantener los estados de la salida.**

Registros de entrada y salida paralelo

Una forma muy sencilla de obtener un registro de entrada y salida **paralelo** es empleando **flip-flops D** asincrónicos (latches) con entrada de habilitación (LE), que se unirán de modo tal de quedar una entrada de habilitación general. La cantidad de biestables que utilicemos determinará la cantidad de bits que almacenará el registro. El funcionamiento es de la siguiente manera: cuando la entrada de habilitación general posee nivel **lógico 1** (habilitado), se habilitarán todos los biestables. Así, el bit que se encuentre en cada entrada **D** será cargado en el biestable correspondiente.



> **La entrada de habilitación Output Enable (OE) permite mostrar los datos a la salida solo cuando esté habilitada dicha entrada.**

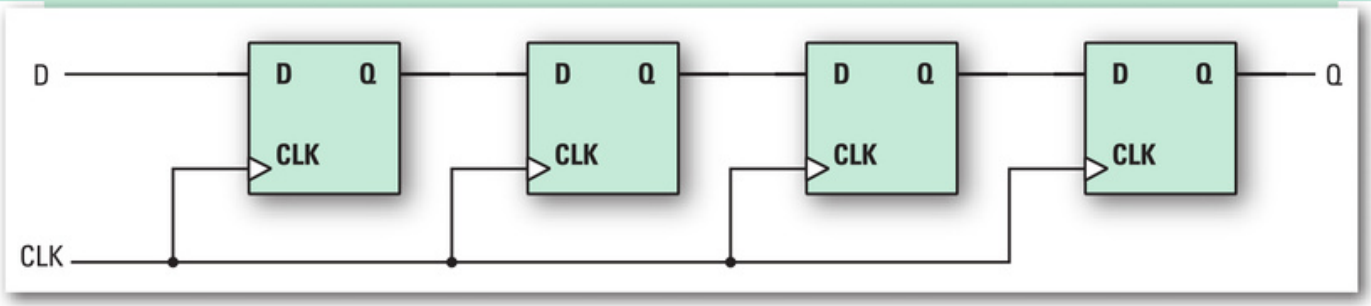


Registros de desplazamiento

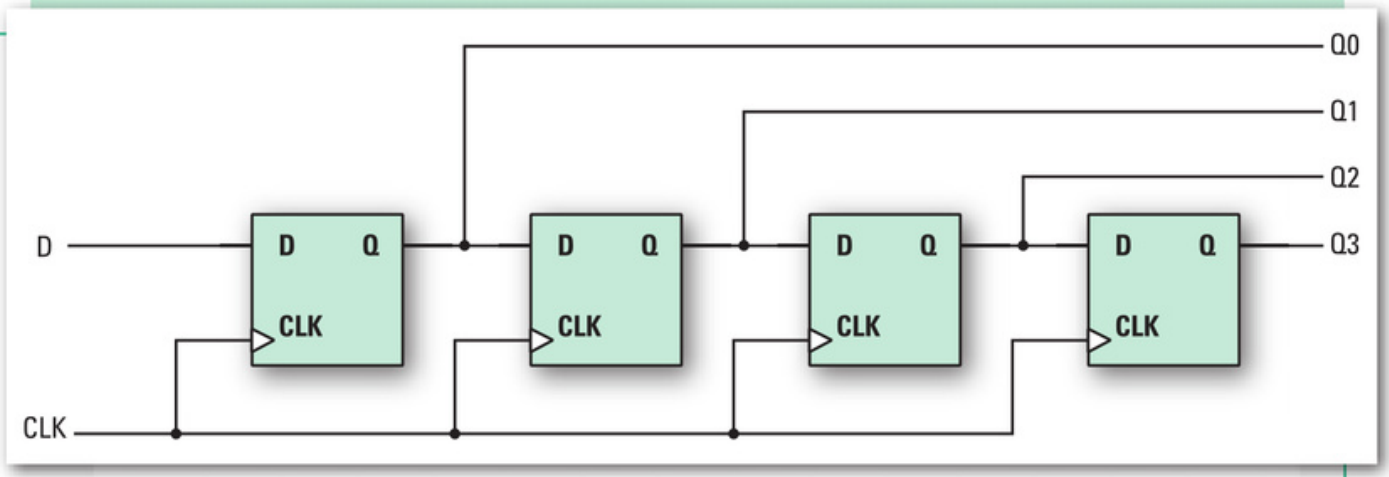
Los registros de desplazamiento deben su nombre a que cada bit de dato se irá desplazando o corriendo en cada ciclo de la señal de reloj. Uno de los más sencillos corresponde al registro de desplazamiento con entrada y salida serie. Está compuesto por **n flip-flops D**, donde **n** será la cantidad de bits que almacene el registro. La señal de reloj es única y se conecta a cada entrada de reloj de los flip-flops. Luego, la conexión de los flip-flops es en cadena, es decir, la salida **Q** del primero se conectará a la entrada **D** del segundo; después la salida **Q** del segundo se conectará a la entrada **D** del tercero, y así sucesivamente. La entrada D del primer **flip-flop** es la entrada serie del registro. Los bits ingresados por ella se irán desplazando en cada ciclo de la señal de reloj por cada uno de los flip-flops, hasta llegar al último de ellos, siendo la salida **Q** de este, la salida serie del registro. Existen registros de desplazamiento **unidireccionales**, que permiten que los bits se desplacen en una única dirección. También están los bidireccionales, los cuales permiten que los bits se desplacen en ambos sentidos. El registro de nuestro ejemplo es unidireccional.

Si en vez de tomar solo la salida **Q** del último flip-flop tomamos la salida **Q** de cada uno, vemos que el mismo registro de desplazamiento con entrada y salida serie se convierte en un registro de desplazamiento con entrada serie y salida paralelo. Existen numerosos dispositivos que requieren de registros para su funcionamiento. Más adelante veremos usos prácticos de estos elementos ■

Existen registros de desplazamiento unidireccionales y bidireccionales, según el sentido de movimiento de los bits.



Podemos observar un registro de desplazamiento unidireccional, con entrada y salida serie.



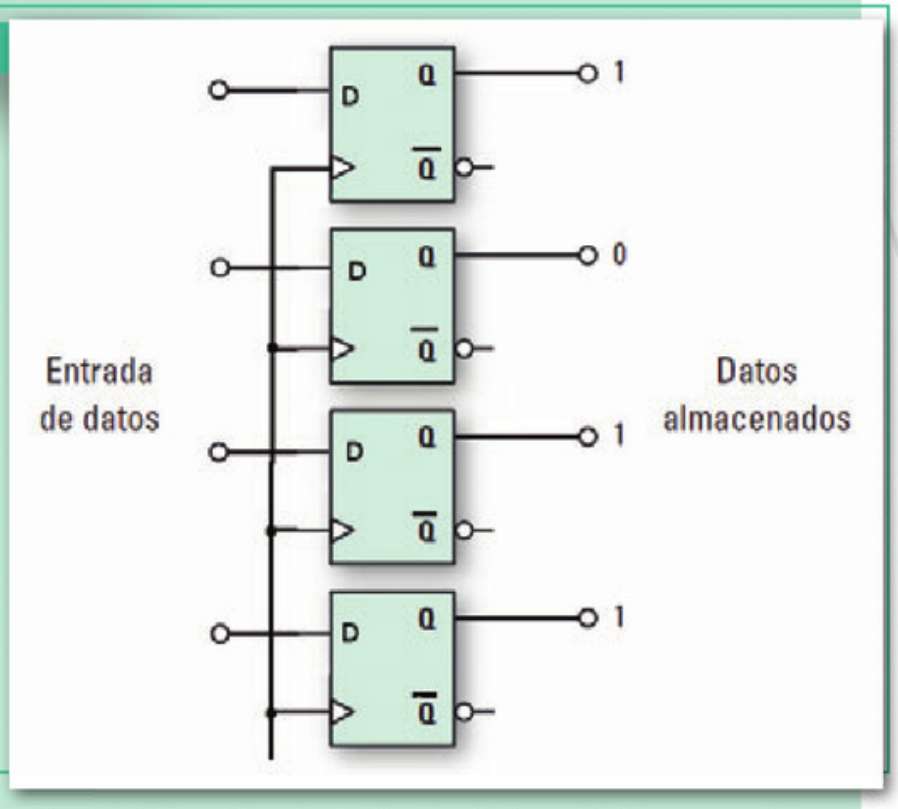
El mismo registro de desplazamiento con entrada y salida serie puede ser de entrada serie y salida paralelo, tomando los bits de cada salida Q de los flip-flops.

CELDA DE MEMORIA

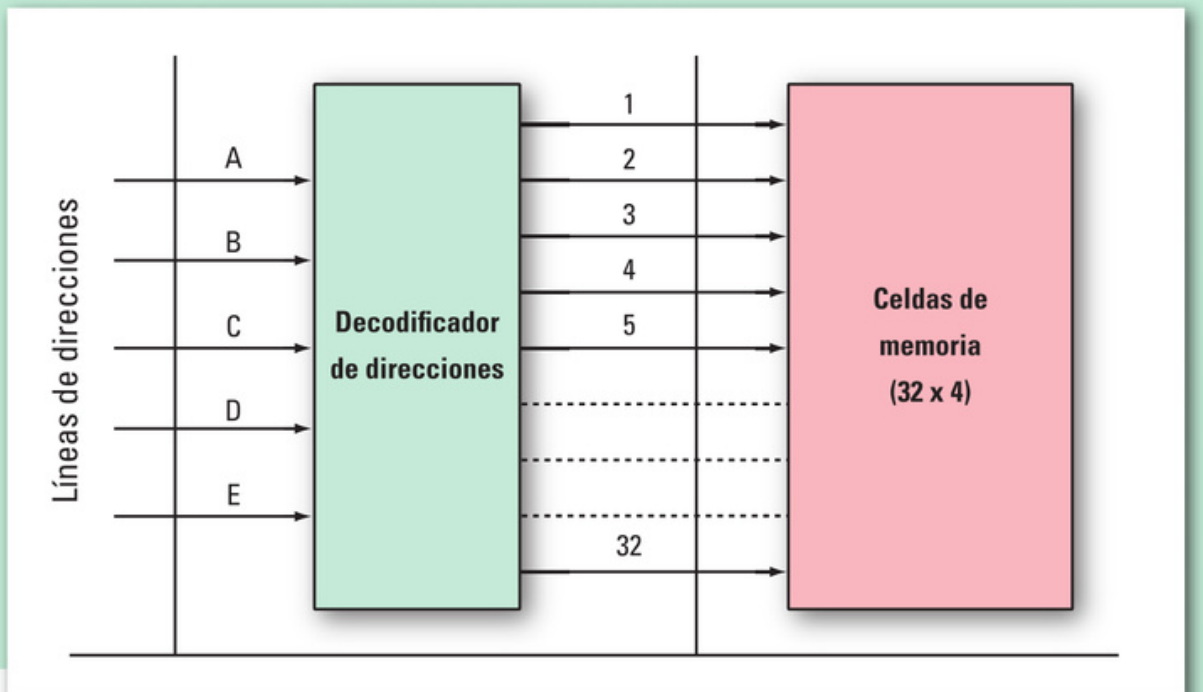
La unidad mínima de almacenamiento de una memoria es la **celda**, y permite guardar un único dato que puede adoptar solo dos valores: 1 o 0. Las celdas, en su forma genérica, están construidas alrededor de un **capacitor** o un **flip-flop**.

En el caso del capacitor, este almacena el valor **1** cuando se encuentra cargado eléctricamente, y el valor **0** cuando está descargado.

Las celdas construidas alrededor de **flip-flops** almacenan el dato **1** cuando el flip-flop presenta un nivel alto en la salida **Q**. Para más detalles sobre este tema, se recomienda revisar el funcionamiento de los flip-flops exoplicado en el apartado anterior. Además de la celda, una memoria emplea en su interior un sistema de decodificación para direccionar los datos de entrada y de salida hacia cada celda de forma lineal. Gracias al decodificador, es posible reducir el número de líneas de acceso a la memoria, ya que, a partir de **n** cantidad de líneas de entrada, obtenemos **2^n** (2 elevado a la N) líneas de salida. Por ejemplo, una memoria con una capacidad de almacenamiento de **128 bits** internamente está organizada en **32** direcciones con una capacidad de **4 bits** cada una. Para direccionar esta cantidad de datos, nuestro decodificador necesitará solo **5 líneas** de acceso, debido a que a su salida entregará **2^5 = 32** direcciones posibles. En general, la capacidad de una memoria se expresa como el resultado de multiplicar el número de posiciones que posee por la longitud de cada palabra de datos.



> **Memoria básica elaborada con flip-flops. Cada uno de ellos es una celda y almacena un único dato, que puede adoptar dos valores posibles.**



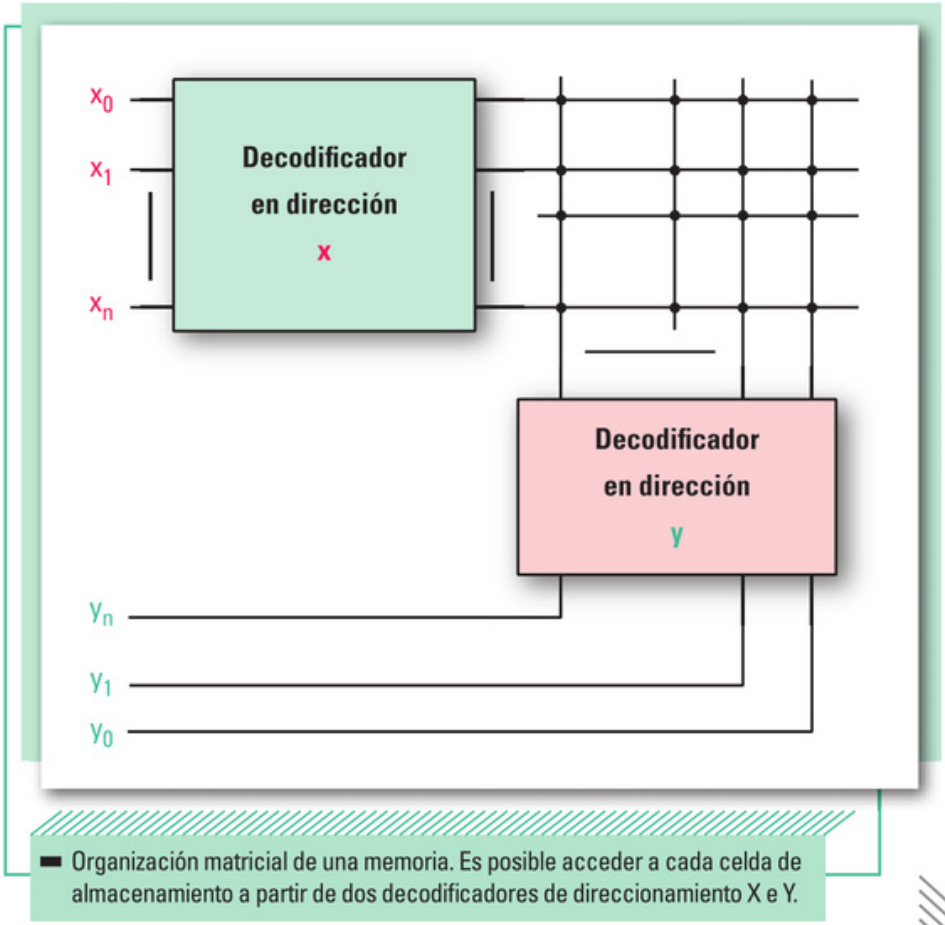
> **Estructura interna de una memoria con una capacidad de 128 bits. Dispone de un total de 32 direcciones con capacidad para palabras de 4 bits y tan solo 5 líneas de acceso.**

Organización matricial

Cuando la capacidad de la memoria es elevada, se torna difícil acceder de manera lineal a su contenido. De este modo, surge la estructura matricial para organizar internamente el dispositivo.
Este tipo de arreglo dispone de celdas situadas en filas y columnas. Las líneas de direccionamiento están divididas en dos bloques: **horizontal** (**X**) y **vertical** (**Y**).
Al utilizar esta estructura, es posible activar las celdas de memoria de manera individual e, incluso, de forma aleatoria. Esta es la principal característica de las llamadas memorias **RAM** (memoria de acceso aleatorio, del inglés *Random Access Memory*).

Lectura de una memoria

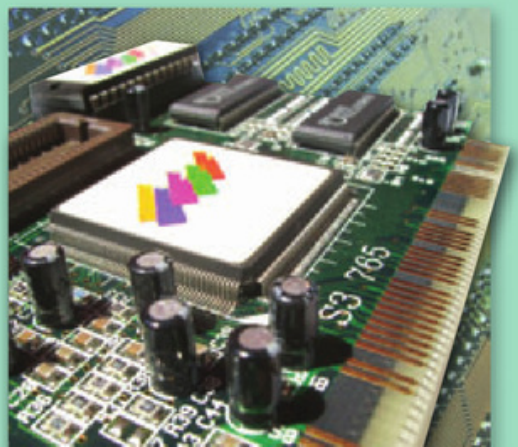
Para realizar la lectura del contenido de una memoria, es necesario seguir algunos pasos. Primero debemos colocar en las líneas de direccionamiento el número binario que represente la dirección a la cual queremos acceder. Luego, tenemos que habilitar el comando de lectura. Por lo general, las memorias disponen de un pin dedicado que permite escoger el modo de funcionamiento entre lectura y escritura. Al leer una memoria, surge un parámetro conocido como **tiempo de acceso**, que mide el tiempo transcurrido entre que se recibe una dirección válida, se habilita el modo de lectura y aparecen los datos en las líneas de salida.



Organización matricial de una memoria. Es posible acceder a cada celda de almacenamiento a partir de dos decodificadores de direccionamiento X e Y.

Escritura de una memoria

Al igual que en el proceso de lectura, para escribir datos en una memoria, es preciso realizar varios pasos. Mediante las líneas de direccionamiento, debemos elegir la dirección de la posición donde almacenaremos los datos.
A continuación, colocamos los datos en las líneas de entrada. Por último, habilitamos el pin de **escritura/lectura** en el modo correspondiente.
Cada vez que escribimos en la memoria, debemos respetar un parámetro conocido como **tiempo de escritura**. Éste indica el tiempo que tienen que estar presentes los datos que se van a escribir y los datos de direccionamiento mientras dure el proceso.



> PINES DE CONTROL

Las memorias disponen de varios pines especiales para seleccionar Lectura/Escritura (**R/W**, **RD**, **WR**) y la presentación de los datos a las salidas (**OE**). También es frecuente encontrar un pin de Habilitación o de Selección de Dispositivo, Chip Select (**CS**, **CE**), para minimizar el consumo.



Medidas de almacenamiento digital

Los medios de almacenamiento digital disponen de cierta capacidad para guardar datos, que puede medirse en bits y sus respectivas derivaciones. Recordemos que un **bit** es el acrónimo de dígito binario (del inglés *Binary Digit*). Un bit representa un dígito del sistema de numeración binario. En dicho sistema se utilizan solo dos dígitos, el **0** y el **1**, a diferencia de lo que ocurre en el sistema decimal, en el cual se utilizan diez dígitos. Un bit puede adoptar uno u otro valor exclusivamente.

Como el bit es una unidad de almacenamiento muy pequeña para los tiempos que corren, suele utilizarse asociado con los denominados prefijos de cantidad. Según el estándar **IEEE 1541**, los tradicionales prefijos del **SI** (Sistema Internacional) no pueden usarse para representar múltiplos binarios.

Por lo tanto, todas las unidades empleadas para indicar cantidades de almacenamiento en el campo de la electrónica digital y la informática poseen una nueva denominación: se introduce la parte **bi** (de binario) al prefijo. Por ejemplo, lo que antiguamente era un **kilobyte** ahora es equivalente a un **kibibyte**, que significa un **kilobinario byte**, que son **1024 bytes**.

UNIDADES		
bit	b	Un dígito binario.
byte	B	Submúltiplo del tamaño de una palabra, generalmente de ocho bits.
octeto	o	Grupo de ocho bits.

>En esta tabla se muestran las unidades correctas para representar cantidades en electrónica digital.

PREFIJOS		
kibi	Ki	210 = 1024
mebi	Mi	220 = 1.048.576
gibi	Gi	230 = 1.073.741.824
tebi	Ti	240 = 1.099.511.627.776
pebi	Pi	250 = 1.125.899.906.842.624
exbi	Ei	260 = 1.152.921.504.606.846.976

> En esta tabla se muestran los prefijos correctos para indicar los múltiplos binarios de las unidades de la tabla anterior.

Prefijos en el uso convencional de la informática

Nombre	Símbolo	Potencias binarias y valores decimales	Valores en el SI	Hexa.	Nombre	Diferencia
Unidad		$2^0 = 1$	$10^0 = 1$	16^0	un(o)	0%
kilo	K	$2^{10} = 1\,024$	$10^3 = 1\,000$	$16^{2,5}$	mil	2%
mega	M	$2^{20} = 1\,048\,576$	$10^6 = 1\,000\,000$	16^5	millón	5%
giga	G	$2^{30} = 1\,073\,741\,824$	$10^9 = 1\,000\,000\,000$	$16^{7,5}$	millardo	7%
tera	T	$2^{40} = 1\,099\,511\,627\,776$	$10^{12} = 1\,000\,000\,000\,000$	16^{10}	billón	10%
peta	P	$2^{50} = 1\,125\,899\,906\,842\,624$	$10^{15} = 1\,000\,000\,000\,000\,000$	$16^{12,5}$	billardo	13%
exa	E	$2^{60} = 1\,152\,921\,504\,606\,846\,976$	$10^{18} = 1\,000\,000\,000\,000\,000\,000$	16^{15}	trillón	15%
zetta	Z	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$	$10^{21} = 1\,000\,000\,000\,000\,000\,000\,000$	$16^{17,5}$	trillardo	18%
yotta	Y	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$	$10^{24} = 1\,000\,000\,000\,000\,000\,000\,000\,000$	16^{20}	cuatrillón	21%

> Prefijos de uso convencional de la informática. A partir del año 2002, se realizaron cambios importantes para no confundir los múltiplos binarios con los del SI.

Clasificación de los sistemas de almacenamiento

Los sistemas de almacenamiento se clasifican según varios aspectos. A continuación, describiremos uno a uno los **principales grupos**.

Las memorias volátiles pierden los datos cuando se interrumpe la energía que las alimenta.

Según su método de acceso

Según el modo de acceso a los datos almacenados en un dispositivo de memoria, podemos clasificarlos en dos grupos: **aleatorio** y **secuencial**. En el primero, se puede acceder a cualquier dato, sin importar su posición física, y con el mismo tiempo de acceso para todas las direcciones. Todas las memorias semiconductoras, como las **RAM** y las **ROM**, pertenecen a este grupo. El segundo grupo, en cambio, obliga a leer o a escribir todas las posiciones físicas previas antes de acceder al dato deseado. En este caso, el tiempo de acceso cambia de acuerdo con el sector físico al que debamos acceder.

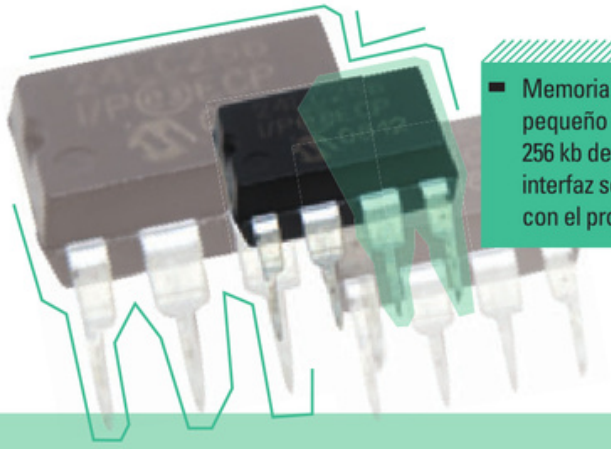


Módulo de memoria RAM en formato SO-DIMM. Se utiliza, principalmente, en computadores portátiles, y es una miniaturización del formato DIMM.

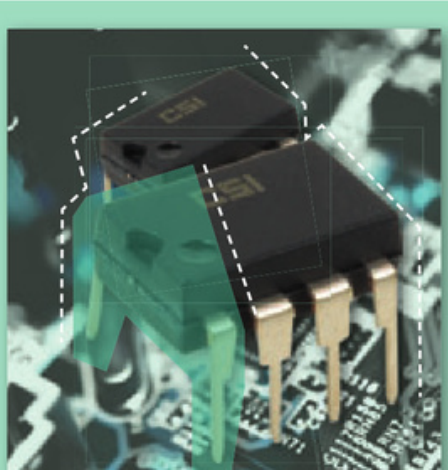
Según su volatilidad

Dependiendo de la capacidad de mantener los datos almacenados, podemos encuadrarlas en dos grupos que vemos a continuación: Por un lado, están las **no volátiles**, en las que la información se mantiene en la memoria a pesar de que el dispositivo no cuente con energía de alimentación. Teóricamente, el tiempo de almacenamiento es indefinido.

Todos los medios del tipo magnético y óptico pertenecen a este grupo. Por el otro lado, están las **volátiles**, que pierden los datos al interrumpir la energía que las alimenta. Son memorias llamadas de trabajo o temporales. Por ejemplo, podemos citar en este grupo todos los tipos de memorias **RAM** utilizadas en las computadoras.



Memoria EEPROM 24LC256. En este pequeño chip podemos guardar hasta 256 kb de datos. Cuenta con una interfaz serial de dos hilos compatible con el protocolo I2C™.



> EEPROM FLASH

Son una variante de la memoria EEPROM que permite escribir o borrar múltiples posiciones de memoria simultáneamente y de forma eléctrica. Se trata de dispositivos rápidos y compactos que almacenan la información de forma no volátil. En la actualidad, son muy utilizados en dispositivos electrónicos, como cámaras fotográficas y reproductores de música portátil, por su tamaño diminuto y su gran capacidad de almacenamiento.

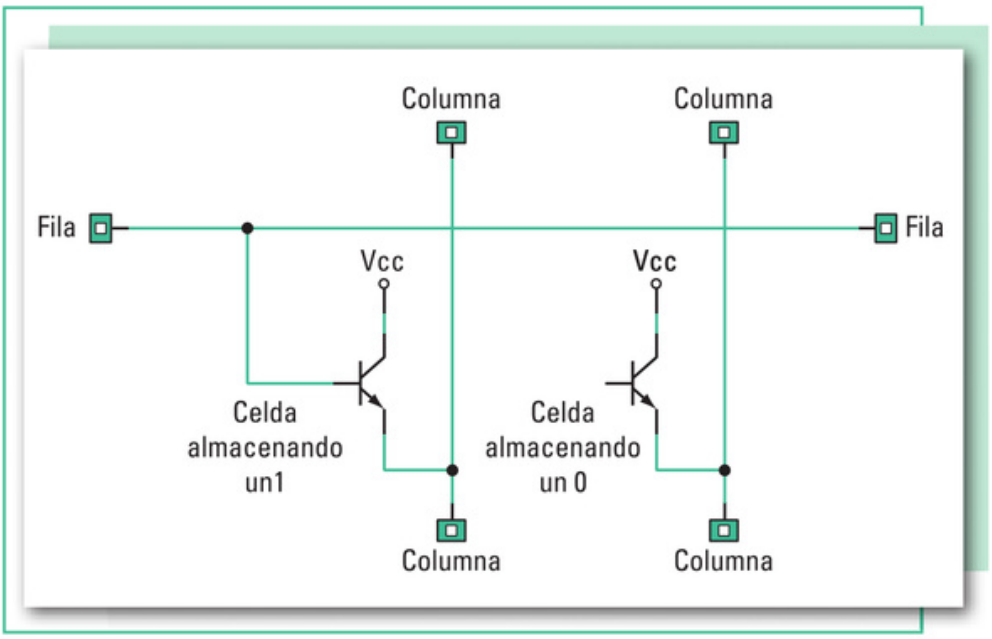
Según su método de escritura

Bajo esta denominación encontramos dos grupos de memorias de **lectura** y de **escritura**. Las más utilizadas son las **RAM**, tanto las **estáticas** (**SRAM**) como las **dinámicas** (**DRAM**).
Una memoria no volátil es el BIOS de una PC, que almacena la información básica para iniciar el sistema.
Las memorias **RAM dinámicas** están construidas sobre diminutos capacitores a modo de celdas. Debido al empleo de esta tecnología, es necesario utilizar una circuitería de refresco para mantener cargados los capacitores y no perder los datos almacenados.
En cambio, las memorias **RAM estáticas** están basadas en **flip-flops**, que no necesitan ningún tipo de refresco para trabajar.

Memoria de solo lectura

En este tipo de memoria, el acceso a los datos es rápido, pero el proceso de escritura es todo lo contrario. Son dispositivos diseñados para mantener la información sin cambios. Dentro de este grupo, encontramos una gran variedad de dispositivos:

- **ROM (Read Only Memory)**: memorias de solo lectura programadas por máscara. Los datos almacenados se escriben en fábrica y no es posible modificarlos ni borrarlos.
- **PROM (Programmable ROM)**: memoria **ROM** programable una sola vez. Al realizar el proceso de grabación, el chip queda inalterable.
- **EPROM (Erasable PROM)**: memoria **PROM** que puede borrarse mediante luz ultravioleta para regrabarla.



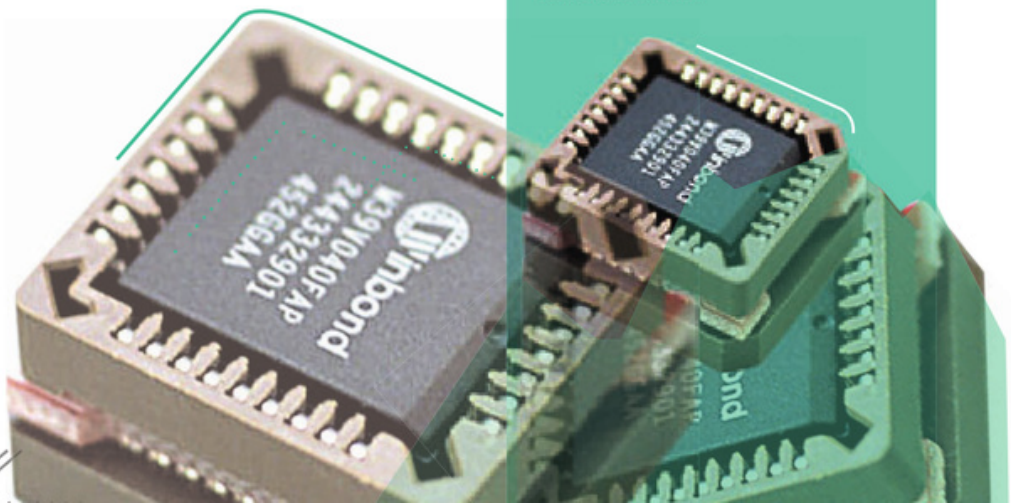
- **OPT (One Time Programmable)**: son memorias **EPROM** sin ventana, por lo que no pueden borrarse.
- **EEPROM (Electrically Erasable PROM)**: memorias que pueden borrarse y regrabarse eléctricamente.
- **FLASH**: es una variación de las memorias **EEPROM**. Son más veloces que ellas, de mayor capacidad y menor consumo.

Memorias ROM

Si realizamos un análisis funcional de la memoria ROM, podemos interpretarla como un **bloque de formato matricial** compuesto por tantas filas como direcciones

> APLICACIONES DE LAS MEMORIAS ROM

Las memorias ROM generalmente se utilizan para contener pequeñas porciones de código, conocidas como firmware, dedicado a manejar hardware a bajo nivel en sistemas microcontrolados. También es usual que se utilicen para almacenar patrones aplicados en gran variedad de aplicaciones, desde tablas de datos o números, hasta la construcción de decodificadores.



disponga, y por tantas columnas como la longitud de la palabra de datos. Para acceder a este bloque se utiliza una combinación de números binarios que se aplican en la entrada de un circuito decodificador de **n** líneas de entradas y **2n** líneas de salida. En cada unión entre fila y columna, se ubican las celdas de memoria formadas por elementos semiconductores, como pueden ser diodos, transistores bipolares o MOS. De acuerdo con el conexionado de dichos elementos, la celda almacenará un **1** o un **0**. Por ejemplo, en una memoria con celdas construidas con transistores bipolares para que el contenido sea un 1, la base del transistor deberá conectarse a la línea de la fila. En cambio, si se desea almacenar un 0, la base quedará sin conexión.

Las celdas de una memoria PROM se construyen alrededor de fusibles, diodos y transistores.

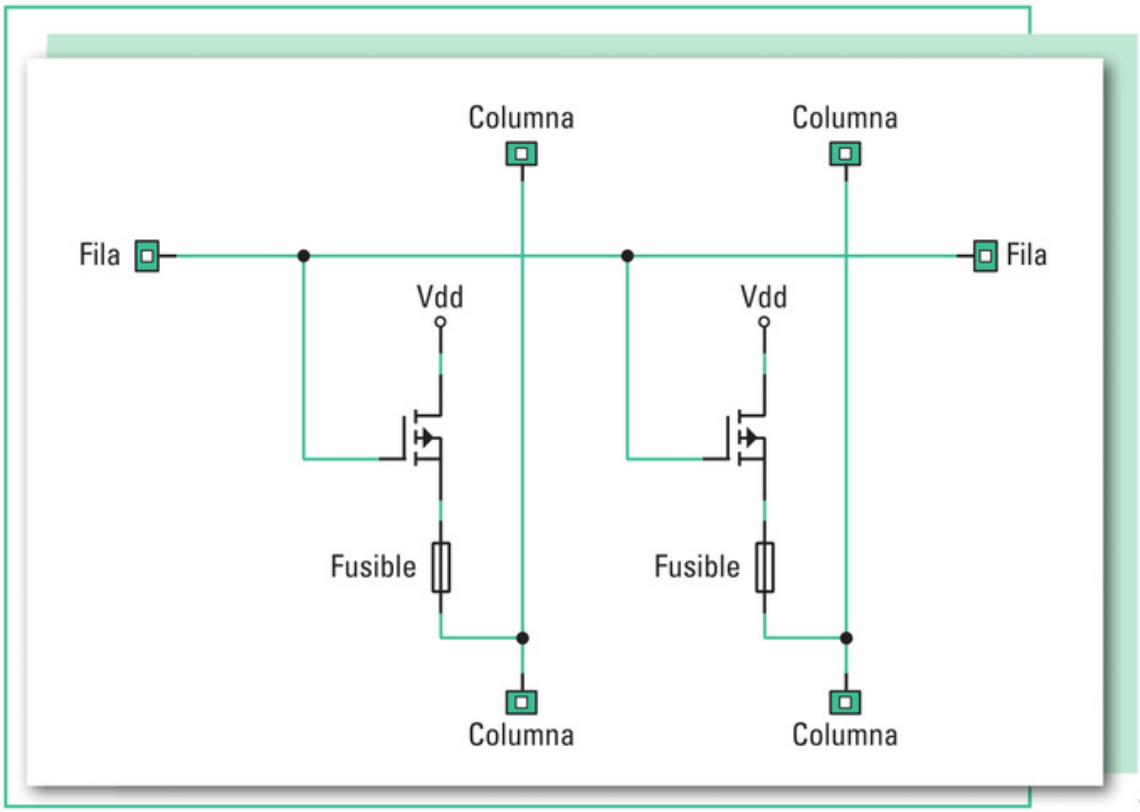
Memorias PROM/PROM con diodos

Una memoria **PROM** es un tipo de **ROM** que puede programarse una única vez. Está construida con una especie de fusibles o antifusibles, que se queman de manera irreversible haciendo circular por ellos una corriente excesiva. Los chips nuevos tienen todos los fusibles intactos, por lo cual todo su contenido se encuentra en estado alto.

Las celdas de memoria, además del fusible, poseen, cada una, un elemento semiconductor asociado –en este caso, un diodo–, encargado de definir el valor almacenado según se encuentre conectado o abierto.

Al realizar el proceso de grabación, se aplica una tensión elevada a la celda deseada, que logra fundir el fusible, con lo cual se abre el circuito y se deja al diodo sin conexión. De esta forma, la celda pasa a almacenar el valor **0**.

El proceso de quemado se efectúa con una herramienta llamada **programador de PROM**, que posee la capacidad de elegir una a una las direcciones de la memoria, y asignarles el valor que queremos almacenar. ■



Celdas de una memoria PROM formada a partir de fusibles y transistores de tecnología MOS. Con este arreglo es posible almacenar una palabra de datos de 2 bits de longitud.

APLICACIÓN DE LAS MEMORIAS PROM

Una memoria PROM puede programarse para almacenar distintos códigos. En este ejemplo veremos la manera de utilizarla como un decodificador de BCD a 7 segmentos.

Un decodificador es un circuito digital de lógica combinacional que posee una n cantidad de pines de entrada y de salida. Cada combinación de datos en su entrada representa un único valor de salida. La función principal de estos dispositivos es realizar direccionamientos de espacios de memoria.

El circuito propuesto

Es posible almacenar una serie de datos en una memoria **PROM con diodos**, para que, a partir de una entrada de datos codificados en **BCD** (palabra de 4 dígitos binarios), se pueda realizar la correspondiente decodificación. De este modo, podemos obtener a su salida **7 líneas** de datos para excitar un **display de 7 segmentos** y formar los dígitos decimales del **0** al **9**.
Nuestro circuito de ejemplo está formado por el bloque decodificador de direcciones (descripto al principio de esta página), una matriz con las celdas de memoria, una serie de diodos (en caso de querer almacenar el dato 1), un conjunto de resistencias **pull-down** (para mantener en nivel 0 cada celda que no posea conexión mediante diodo) y una serie de buffers a la salida. Estos últimos son un tipo de compuerta con una alta capacidad de corriente que permite manejar directamente pequeñas cargas como un **LED**. Utilizaremos como dirección de los datos un número BCD, y el dato de salida será el del código de 7 segmentos.
La información que vamos a almacenar en la memoria se toma de la tabla de verdad.
En nuestro caso, como solo aprovecharemos los datos correspondientes a los decimales del 0 al 9 (un total de 10 combinaciones), utilizaremos el código **BCD** desde **0000** hasta **1001**, y el resto será ignorado.

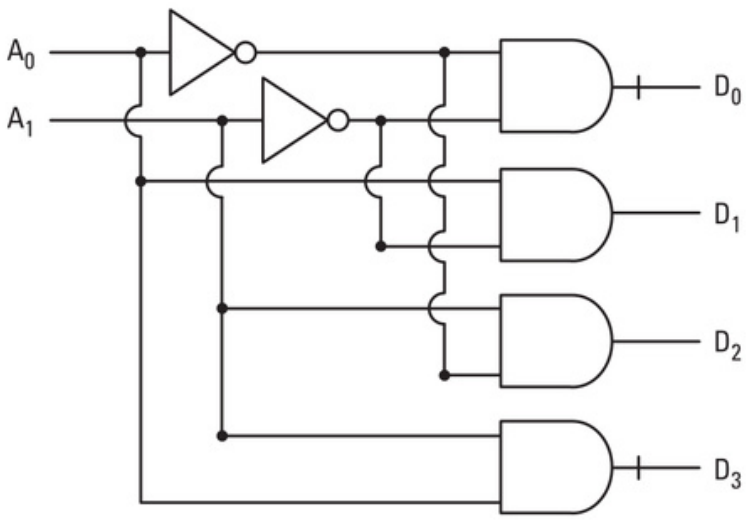
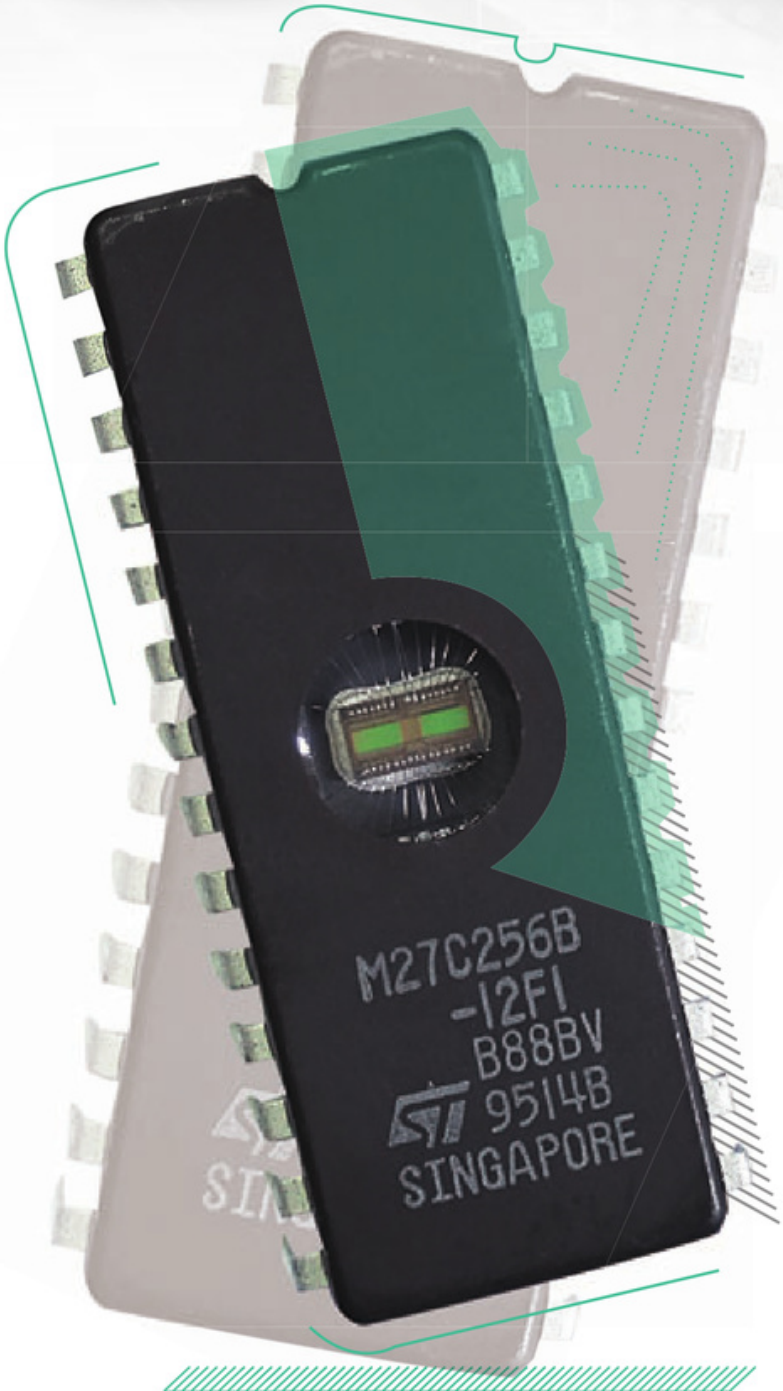


Tabla de verdad

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

> **Tabla de verdad de direccionamiento de datos para nuestro circuito. Los datos marcados con cruces son ignorados.**



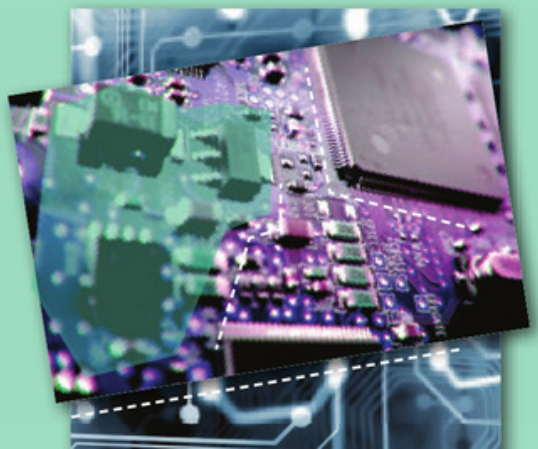
Memoria 27C256. Podemos observar la ventana de cuarzo utilizada para borrar todo el contenido de la memoria por medio de rayos UV.

Memorias EPROM

La **EPROM** (***E**rasable **P**rogrammable **R**ead **O**nly **M**emory*, o memoria de solo lectura programable y borrable) es una memoria no volátil que, como su nombre lo indica, puede programarse y borrarse por completo.
Su tecnología de fabricación permite que, por medio de una fuente de luz ultravioleta, todo su contenido sea eliminado. Por este motivo, lleva en su encapsulado una pequeña ventana de cuarzo por la cual pueden atravesar los rayos ultravioleta para lograr este fin. Para evitar que los datos almacenados en la **EPROM** sean alterados por fuentes lumínicas exteriores, especialmente el sol, la ventana se cubre con una etiqueta oscura que no permita el paso de las radiaciones UV.

Toda memoria vacía mostrará a su salida todos los 1 o, lo que es lo mismo, en hexadecimal, todas FF.

La programación de esta memoria se diferencia de la correspondiente a la **PROM** debido a que ésta no requiere que sean fundidos sus fusibles, sino que induce cargas en la compuerta aislada de un transistor **MOS** por cada **celda/bit**. Cuando una **EPROM** está vacía o borrada por completo, todas sus celdas están descargadas. Por lo tanto, la salida será siempre un **1** lógico (el drenaje del transistor MOS). Dado que todas las celdas están en **1**, solo se programan los 0, aunque deben introducirse todos los datos, incluidos los **1**. Por este motivo, se resulta indispensable borrar por completo la memoria antes de reprogramarla, dado que no es posible cambiar el valor de la celda de **0** a **1** (solo por UV), pero sí, de **1** a **0**.



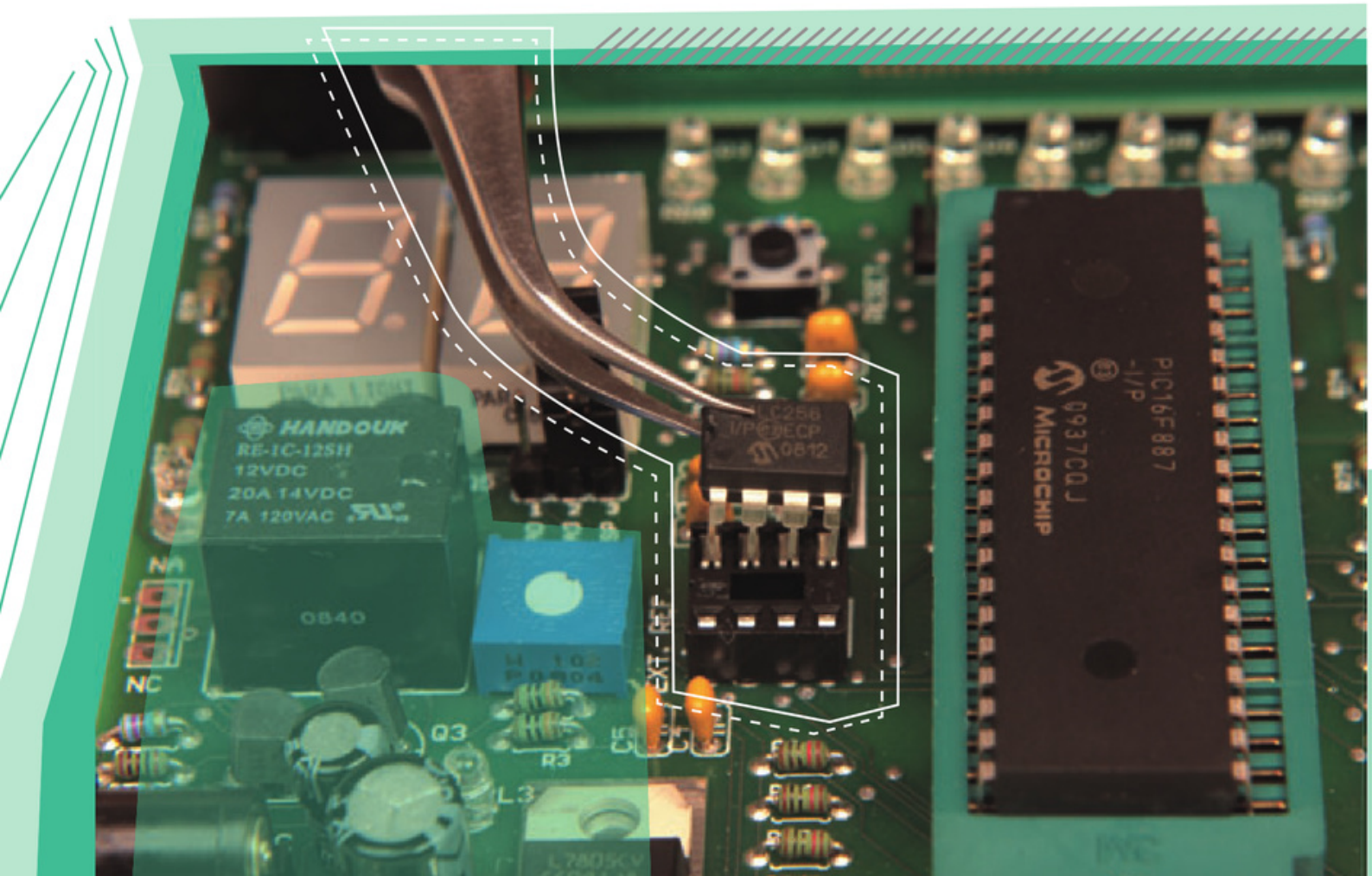
> APROVECHANDO LAS MISMAS CONEXIONES

Debemos destacar una propiedad única que existe en muchos circuitos integrados y memorias, y que se llama OE (Output Enable – Habilitar Salidas) y que permite conectar y desconectar (aislar en HI-Z – Alta Impedancia) internamente las salidas de datos. De esta forma, podemos utilizar las mismas conexiones o buses del circuito para otros propósitos.

La retención de los datos en las memorias **EPROM** puede llegar a 20 años. Debido al costo de la ventana de cuarzo, se han implementado EPROMs grabables solo una vez (*One Time Programmable*, OTP), cuyo uso está destinado a grandes volúmenes de producción.
Solo están disponibles en formato de datos paralelo; esto es, cada bit de cada posición de la matriz de memoria está asociado a un pin del encapsulado. Por ejemplo, si una memoria tiene una palabra de **8 bits**, se requerirán 8 pines para acceder a la palabra (o posición) seleccionada.
Algunas memorias EPROM representativas son:
27C16, 27C32, 27C128, y otras.

Memorias EEPROM

La memoria EEPROM, también conocida como E2PROM, es idéntica a su antecesora EPROM, con la diferencia de que es una ROM programable y borrable eléctricamente (*Electrically Erasable Programmable Read-Only Memory*). Esto significa que podemos borrarla por medios eléctricos.
Por lo general, las **EEPROM** pueden ser accedidas a través de un bus de datos paralelo o serie, dependiendo del modelo.
El formato serie permite utilizar encapsulados más pequeños, de modo que se ahorra espacio y material en los **PCB**; además, son más económicas.
La transmisión de datos en serie consiste en enviar, por medio de un solo cable, los **1** y **0** uno detrás del otro. Si comparamos la transmisión serie con el acceso paralelo, notaremos que para obtener **8 bits** será necesario enviar (o recibir) los datos uno a uno, mientras que en **paralelo**, en una sola operación, los obtenemos de forma casi instantánea en los pines de la memoria. A pesar de la supuesta ventaja que esto representa en la velocidad de las memorias paralelo, las serie son, día a día, más rápidas, y en muchos casos llegan a superar a las otras. Hoy es casi imposible pensar en un desarrollo que requiera una memoria en paralelo. Veremos la comunicación serie más adelante.
Las memorias **EEPROM** pueden **borrarse/regrabarse** entre 100.000 y 1.000.000 de veces y, normalmente, su tiempo de almacenamiento es de **40 años**.
Para borrar por completo o solo un sector de una memoria **EEPROM**, debe borrarse de a una posición por vez, hasta completar el proceso.
Algunas memorias EEPROM representativas son: 28C64, 28C128, 24C02, 93C86 y 25C256.





> **Debido al bajo costo de las memorias Flash en relación a sus pares EEPROM, su uso se ha masificado y hoy se utilizan en cientos de dispositivos, entre ellos, en los pendrives.**

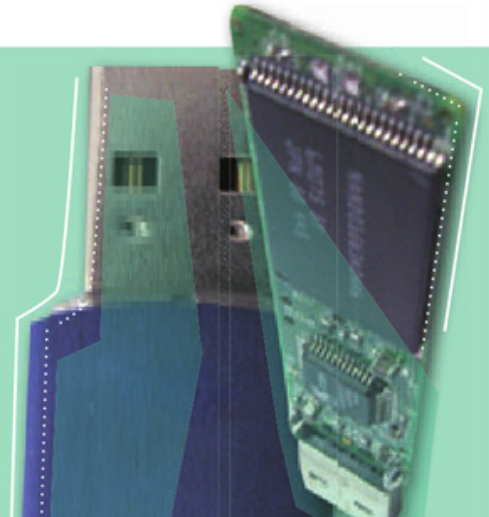
Memorias Flash

Son una evolución de las **EEPROM** y mantienen las mismas características de estas.
Su principal diferencia es que pueden borrarse por completo en una sola operación, lo cual las hace mucho más veloces que aquellas, aunque a veces esto mismo es un inconveniente. Desde hace algunos años, existen memorias Flash que permiten borrar por sectores llamados bloques, en vez de borrar toda la matriz por completo.
Los bloques son grandes sectores en los cuales está dividida toda la matriz de la memoria.

Debido a que la grabación requiere de un voltaje superior al de alimentación, en la actualidad ese voltaje es generado internamente, con lo cual se libera al desarrollador de tener que incorporar una tensión extra al circuito.
Estas memorias soportan entre 100.000 y 1.000.000 de ciclos de borrado/reprogramación y tienen 100 años de retención de los datos.
Las memorias Flash representativas son: 49F010, 29C512 y 25FS010.

Lectura y escritura

Los formatos de grabación varían entre los distintos fabricantes de memorias. En las paralelas, es común la siguiente secuencia: cargamos la dirección; configuramos el pin R/W (lectura-escritura); en caso de grabar, cargamos los datos en el bus de datos; finalmente, activamos y desactivamos el pin CS (Chip-Select).
En el caso de lectura, obtendremos los datos en el bus de datos.

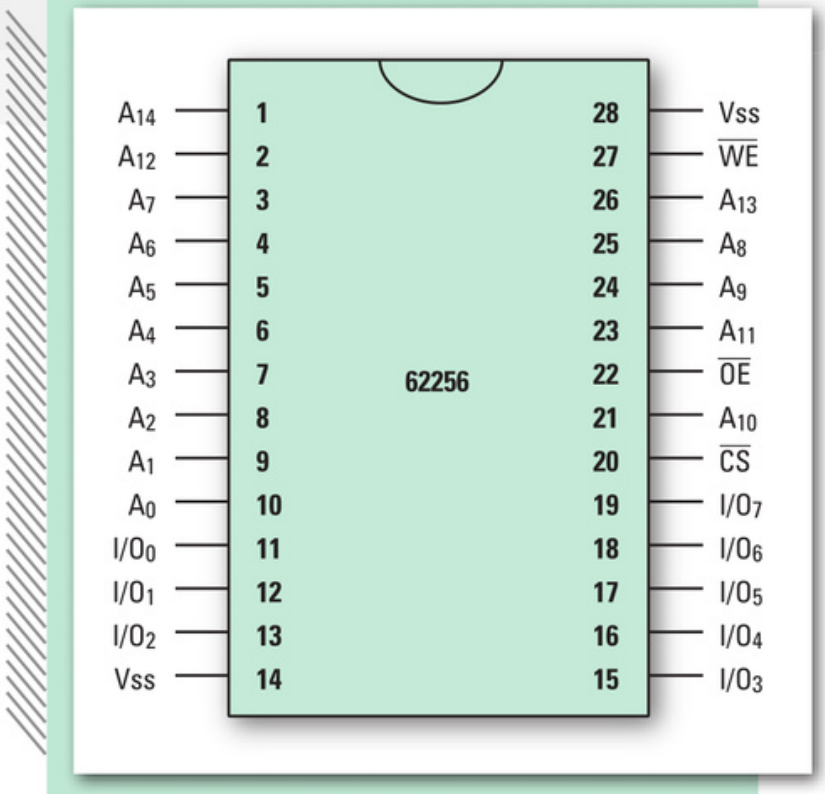


> MEMORIA FLASH

Cuando hablamos de memorias Flash, no solo estamos haciendo referencia a los conocidos Pendrive, sino a una gran variedad de memorias con diferentes factores de forma. Entre las numerosas variantes encontramos las SD, mini SD y micro SD, entre otras. Todas ellas se diferencian en su capacidad de almacenamiento y factor de forma (formato físico).

Memorias RAM

RAM significa memoria de acceso aleatorio (*Random Access Memory*), es decir que podemos acceder a cualquier posición de ella para leerla o escribirla (grabarla). Ahora bien, una memoria semiconductora **ROM** y sus derivadas también permiten tener un acceso aleatorio. Entonces, ¿cuál es la diferencia, si en ambos tipos podemos acceder a cualquier posición y obtener los datos? En los primeros tiempos de la computación, una de las memorias de almacenamiento de programas era la cinta magnética, y como es lógico suponer, el acceso a ella era secuencial; es decir, para obtener un dato grabado al final de la cinta, debíamos pasar por todas las posiciones anteriores antes de dar con él. En cambio, en la memoria **RAM** (si bien no se la usa para almacenar programas) es posible acceder directamente a cualquier posición, independientemente de cuál fue la última leída. Esta es la historia de su denominación.
Todas las memorias semiconductoras (sean **ROM** o **RAM**) son de acceso aleatorio. Por lo tanto, para nosotros, la denominación de RAM significará otra cosa, que es su característica fundamental: ser **volátil** (no siempre es así, como veremos en las memorias **FRAM**). Las memorias **RAM** son volátiles, es decir, no retienen los datos cuando el suministro de alimentación es interrumpido.
Su uso está reservado exclusivamente a almacenar datos temporales con los que efectuaremos cálculos o procesos, pero que no nos interesa guardar de manera indefinida. Debido a su tecnología de almacenamiento, el tiempo de grabación y lectura es muy inferior al de sus contrapartes de solo lectura.



> **Esta es la distribución de pines de una memoria RAM 62256 capaz de almacenar 32.768 palabras de 8 bits.**



> **El gran encapsulado requerido para el acceso paralelo a los datos.**

Memorias RAM estáticas

La forma más antigua y simple de memoria **RAM** es la denominada estática o **SRAM**. Su principio de funcionamiento radica en el hecho de que está realizada a partir de una celda básica de memoria: el **flip-flop**. Es extremadamente veloz, pero dado que un flip-flop suele construirse con **3** a **6** transistores, el espacio que ocupa en el chip es muy grande. Por lo tanto, la capacidad de almacenamiento total está limitada a la superficie. No requiere de circuitos electrónicos externos para el mantenimiento y la gestión, como ocurre con las **DRAM**, que veremos a continuación.

En la actualidad, disponemos de memorias **RAM** con interfaz serie, además de las clásicas en paralelo. El tamaño o capacidad de almacenamiento está dado por la cantidad de líneas o pines (bits en el caso de memorias serie) de dirección disponibles por la cantidad de líneas (o bits en seriales) de datos: **2m palabras**, o **2m x n bits**; donde **m** es la cantidad de líneas de dirección, y **n**, la cantidad de líneas de datos. Por ejemplo, en una memoria con **4 líneas** de dirección y **8 líneas** de datos: **2^4 palabras de 8 bits (1 byte) = 16 bytes, o 2^4 x 8 = 128 bits**.

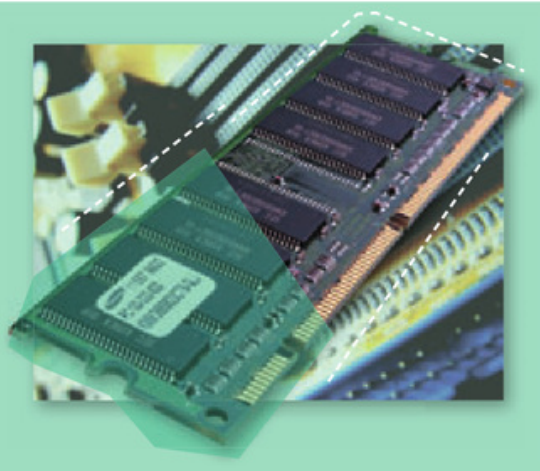


> **La desventaja de las DRAM es que necesitan un circuito exterior para realizar el refresco.**

En algunos circuitos se utilizan memorias RAM energizadas con baterías para preservar los datos.

Memorias RAM dinámicas

Las memorias **RAM** dinámicas o **DRAM** utilizan una tecnología de almacenamiento diferente de las **RAM** estáticas, que se basa en celdas capacitivas. Desde este punto de vista, la tecnología es similar a la empleada en las **EEPROM**. Sin embargo, dado que la velocidad de escritura de una memoria **RAM** debe ser muy rápida, estos capacitores tienen que ser lo suficientemente pequeños como para cargarse al instante. A diferencia de la **EEPROM**, el valor de capacidad es tan reducido que llega a mantener la carga solo durante algunos milisegundos. Por este motivo, las memorias **DRAM** necesitan de un refresco permanente para mantener la carga en el capacitor. ■



> VENTAJAS DE LAS DRAM

Esta tecnología permite, en un mismo espacio físico, lograr matrices de memoria con una capacidad total muy elevada por sobre una RAM convencional o estática. Además, el consumo de potencia es muy inferior, ya que la corriente para mantener la carga de un capacitor no es la misma que la necesaria para mantener energizados seis transistores en el caso del flip-flop de las SRAM.

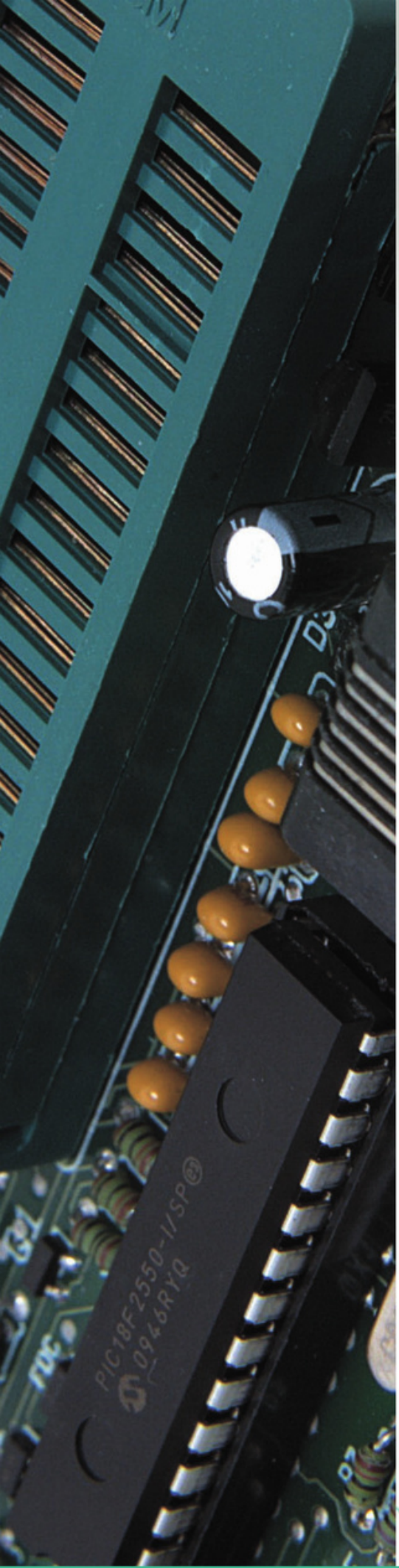
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué es el sistema decimal, binario y hexadecimal?
2. ¿Cuál es la función que desarrollan las compuertas lógicas?
3. Defina flip-flop T.
4. Mencione los alcances de un circuito secuencial.
5. ¿Qué son los contadores?
6. ¿Para qué sirven los multiplexores?
7. ¿Cómo funcionan los demultiplexores?
8. Explique para qué sirven los registros.
9. ¿Qué es una celda de memoria?
10. Caracterice a una memoria Flash.

EJERCICIOS PRÁCTICOS

1. Convierta un número de decimal a binario y a hexadecimal.
2. Realice algunos cálculos aritméticos utilizando el sistema binario.
3. Ejemplifique una compuerta OR mediante un esquema sencillo.
4. Utilice la calculadora de Windows para convertir un número de un sistema de numeración a otro.



▶ CAPÍTULO 2

MICRO-CONTROLADORES

¿Cómo funcionan?
Unidades de entrada y salida
Historias de microprocesadores
Microcomputadoras: ¿de qué se tratan?
Las interrupciones
Periféricos en microcontroladores
Conversores analógico-digitales
Actividades

02

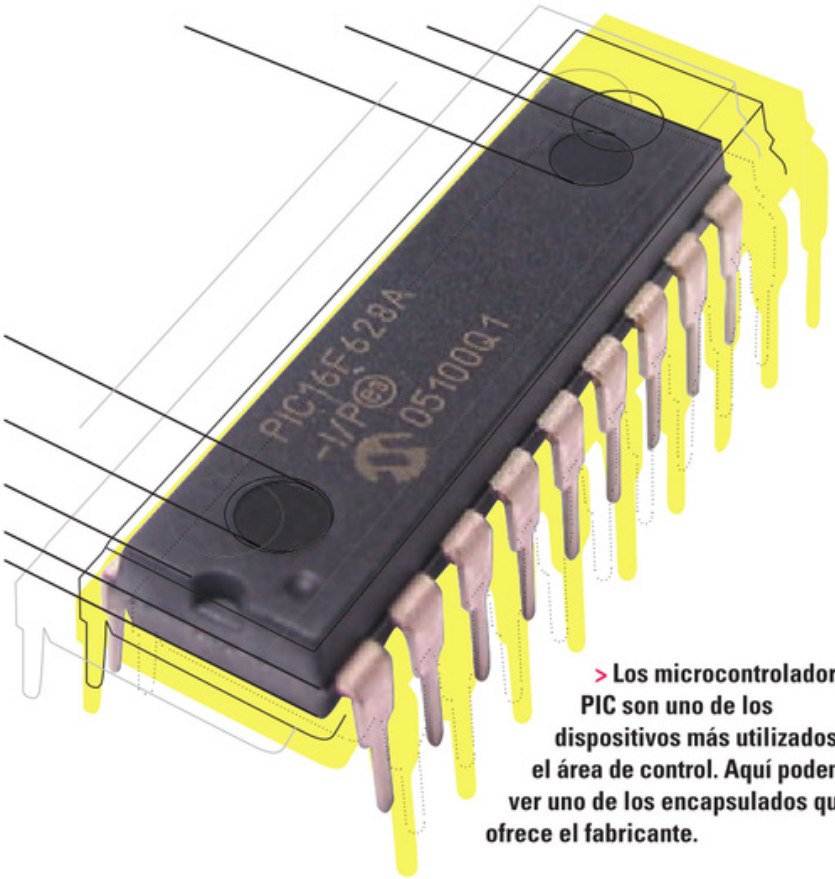
¿CÓMO FUNCIONAN?

Comenzaremos a incursionar en los elementos que más auge han tenido en estos últimos tiempos, dentro de la electrónica digital: los microprocesadores y microcontroladores.

Los microprocesadores y los microcontroladores han cambiado la forma de pensar y diseñar los circuitos electrónicos. Desde que Intel lanzó en 1971 el **8080**, el primer microprocesador exitoso, estos dispositivos no han dejado de evolucionar, y hoy en día es imposible imaginarse la vida sin ellos. Pero a esta altura del tema nos surgen las primeras preguntas: ¿qué son los microprocesadores? ¿Y los microcontroladores? ¿Cómo podemos diferenciar unos de los otros?

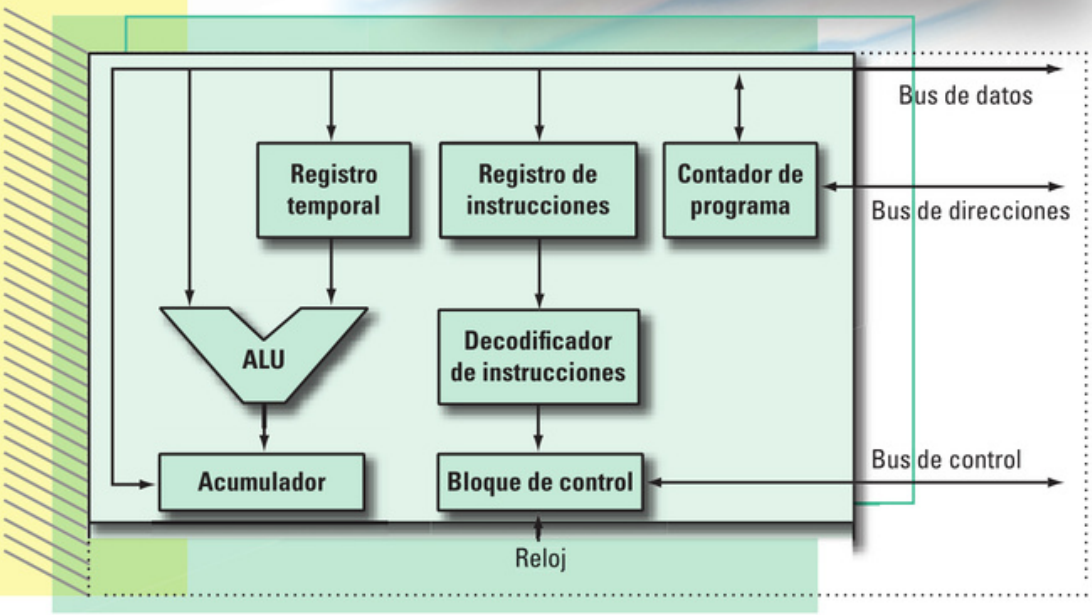
¿Qué es un microprocesador?

Los microprocesadores son circuitos integrados que contienen millones de transistores en su interior, los cuales crean circuitos complejos encargados de realizar diferentes tareas. También se los denomina **unidad de procesamiento central o CPU**, ya que muchos de ellos pueden actuar como el "cerebro" de un sistema computacional, administrando todas las tareas que este realice y llevando a cabo las operaciones con los datos.
Los microprocesadores están diseñados para interpretar y ejecutar las instrucciones que nosotros les indiquemos y que suelen ser operaciones simples, como sumar, restar, multiplicar y dividir. Pero también existen instrucciones lógicas, como **AND**, **OR**, **NOT**, etc. El listado de instrucciones recibe el nombre de **programa**, que las ejecuta una por una por medio del microprocesador.
La potencia, el tamaño y la complejidad de los microprocesadores fueron incrementándose con el correr del tiempo. Tanto es así, que hoy en día podemos observar procesadores que integran millones de transistores en su interior y con varios núcleos para aumentar su capacidad de procesamiento.



> **Los microcontroladores PIC son uno de los dispositivos más utilizados en el área de control. Aquí podemos ver uno de los encapsulados que ofrece el fabricante.**

Los microprocesadores son circuitos integrados complejos encargados de realizar diferentes tareas.



> Figura 1. Diagrama básico de los componentes que integran un microprocesador.

Diagrama básico de un microprocesador

En la **Figura 1** podemos observar el diagrama básico de un microprocesador. El elemento principal es la **ALU** (unidad aritmético-lógica), que se encarga de llevar a cabo todas las operaciones lógicas y aritméticas que requieran los procesos que se ejecuten. También podemos ver los **registros para almacenamiento temporal** de los datos; el **contador de programa**, que contiene la dirección de memoria de la siguiente instrucción por ejecutar; un **registro de instrucciones** que almacena el código de la instrucción en ejecución y el **bloque de control**.

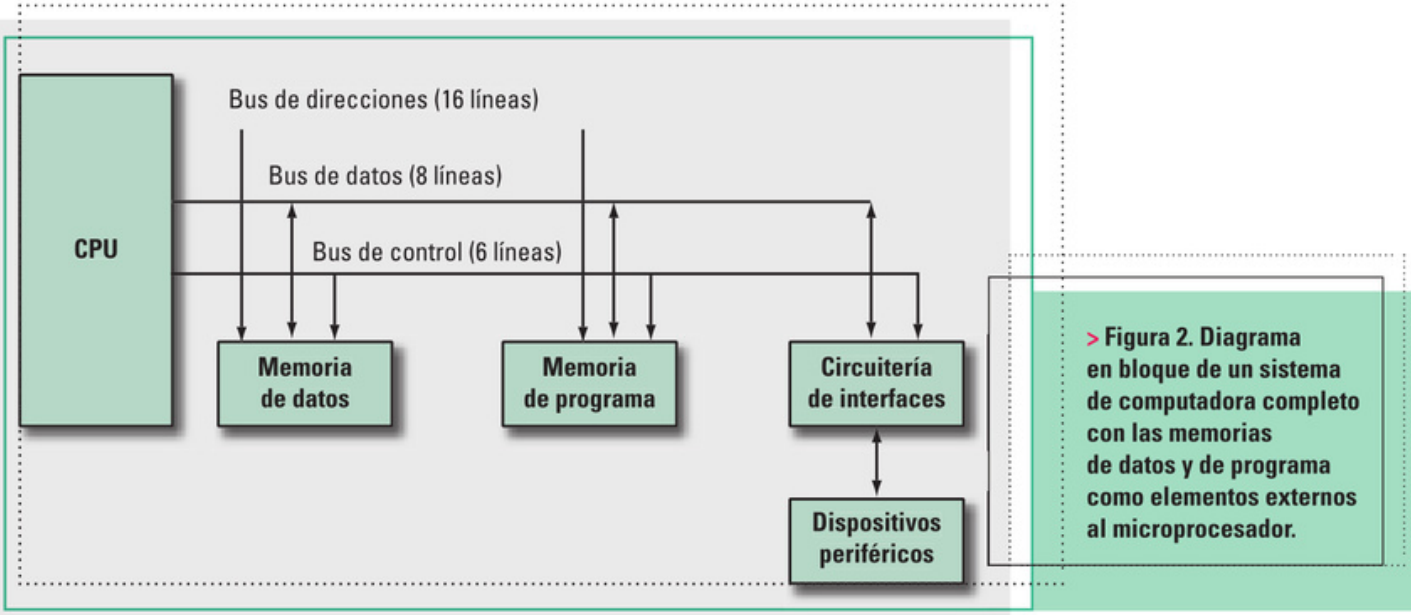
Todo este bloque de circuitos lógicos realiza dos operaciones de manera continua: la búsqueda de una instrucción (**fetch**) y su ejecución.

La búsqueda de una instrucción en la memoria de programa es la operación fundamental del procesador y se efectúa de la siguiente manera:

- El dato que contiene el contador de programa nos indica cuál es la dirección de la próxima instrucción que se ejecutará, y es colocado en el **bus de direcciones**.
- La unidad de control envía una señal de lectura hacia la memoria de programa a través del **bus de control**.
- Los datos contenidos en la dirección de memoria de programa son cargados en el **bus de datos**.
- Estos datos son procesados para que el código de operación se almacene en el registro de instrucciones, y los datos, en los **registros de almacenamiento temporal**.
- Finalmente, el contador de programa se incrementa para volver a buscar la siguiente instrucción.

En la operación de ejecución, se decodifica el código de operación, y la unidad de control genera las diferentes señales que permiten la entrada y la salida de los registros internos hacia la unidad aritmético-lógica.

La **ALU** ejecutará la operación que le indique la unidad de control, y puede devolver el resultado en un **registro interno**, normalmente llamado **acumulador**, o en una posición de la memoria de datos.



> Figura 2. Diagrama en bloque de un sistema de computadora completo con las memorias de datos y de programa como elementos externos al microprocesador.

Unidad aritmético-lógica

La unidad aritmético-lógica se ocupa de realizar las operaciones necesarias con los datos. Ellas son la **suma** y la **sustracción** de números enteros, las operaciones lógicas –como **AND**, **OR** y **NOT**– y las operaciones de **desplazamiento** de bits.

La salida o el resultado de la operación se almacena, como ya dijimos, en un registro interno del microprocesador llamado acumulador.

Este registro también puede utilizarse como un operando, con la ventaja de que ganamos en velocidad y eficiencia en las operaciones complejas.

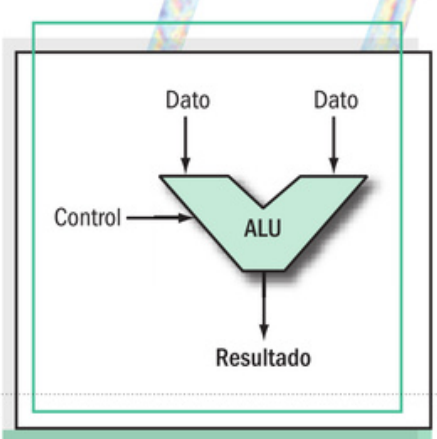
La unidad posee, además, una **entrada de control**, desde donde la unidad de control le indica cuál es la operación por realizar.

Cabe aclarar que en procesadores más complejos podemos encontrarnos con unidades aritmético-lógicas que pueden calcular la multiplicación y hasta la división de los operandos.

Contador de programa

El **contador de programa** (PC) es, básicamente, un **registro contador** que incrementa su cuenta con cada ejecución de una instrucción.

El contenido de este registro contador apunta a la dirección donde reside la instrucción que se desea ejecutar en la memoria de programa.



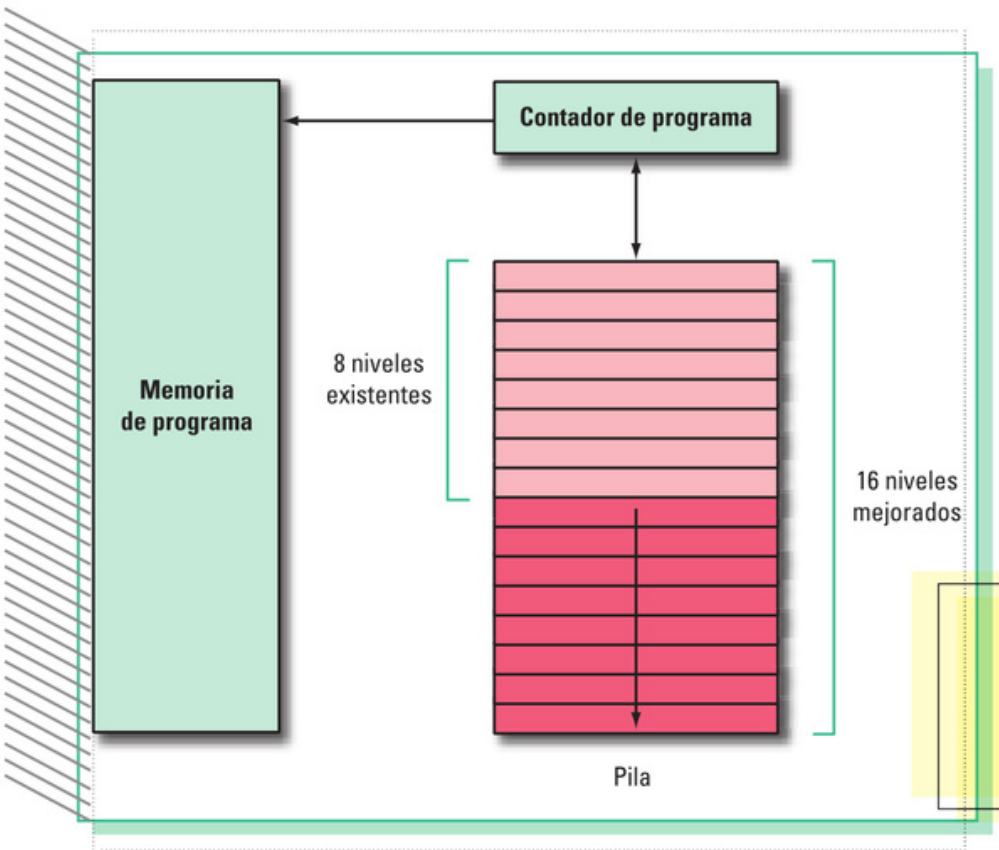
> **Figura 3. Esquema clásico de una unidad aritmético-lógica con las entradas de datos y de control, y una única salida con el resultado de la operación.**

Existen ALUs que calculan la multiplicación y la división de los operandos.

El valor inicial del contador siempre es la dirección donde está la **primera** instrucción del programa. Además, es posible modificar su contenido para crear saltos hacia rutinas que estén en lugares específicos de la memoria de programa.

La cantidad de **bits** que posee el contador sirve para calcular cuál es la cantidad máxima de instrucciones que puede direccionar el procesador. Con esto tenemos una idea del tamaño de memoria de programa que podemos utilizar.

Si tenemos un contador de programa de **16 bits**, podemos direccionar, como máximo, hasta una memoria de **64 KB (2^16)**.



> **Figura 4. El contador de programa siempre contiene la dirección de la próxima instrucción que se ejecutará. Después de un reset del sistema, se inicializa con el valor cero.**



> **Una antigua memoria EPROM cuyo contenido podía borrarse por medio de luz ultravioleta. Era muy utilizada para almacenar memoria de programa.**

Memoria de datos y de programas

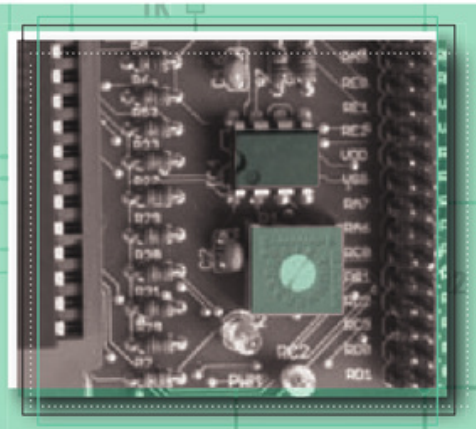
Los programas y datos que controlan la ejecución de las instrucciones en un microprocesador necesitan ser almacenados en memorias de datos y de programa.

Memoria de programa

Para realizar una tarea específica, un microprocesador necesita de un programa que le indique, instrucción por instrucción, cuáles son los pasos que debe cumplir. Este programa reside en una memoria externa al procesador llamada **memoria de programa**. Su característica principal es que no debe perder su contenido cuando el sistema no está energizado. Normalmente, se utilizan memorias **ROM** de solo lectura para grabar un programa, porque poseen esta característica tan preciada, aunque tienen la desventaja de que solo pueden ser escritas una sola vez.
Gracias al avance de la tecnología, hoy podemos utilizar memorias **EEPROM** y **Flash** para almacenar código, con la ventaja adicional de que es posible borrar su contenido eléctricamente.

Memoria de datos

La memoria de datos es también una memoria externa al microprocesador, pero que se encarga de almacenar la información que precisa el procesador para ejecutar las operaciones que le indiquemos.
El tipo de memoria que se emplea para los datos es la **RAM**, porque puede almacenar datos temporales que pueden ser escritos y leídos una infinidad de veces. Además, son de acceso rápido, por lo que el sistema gana en velocidad y eficiencia.
También es posible querer almacenar datos que no se pierdan luego de que el sistema se queda sin alimentación. En este caso, se pueden utilizar memorias no volátiles, como las **EEPROM** de baja capacidad, ideales para guardar contraseñas o nombres de usuarios. ■

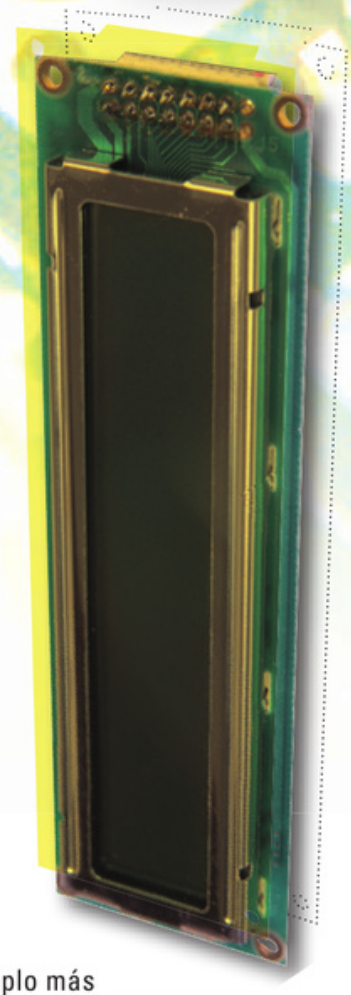


> MAPA DE MEMORIA

El mapa de memoria indica las áreas de las que dispone el procesador, su extensión y direccionamiento. Según la arquitectura, puede haber solo un área general (Von Neumann), un área de datos y otra de programa (Harvard), o, incluso, áreas específicas de entrada/salida (Z80, 80 × 86, AVR). El procesador puede tener registros fuera de este mapa de memoria.

UNIDADES DE ENTRADA Y SALIDA

Para que un procesador pueda comunicarse con el mundo externo, necesita unidades de entrada y de salida que codifiquen los mensajes para interpretarlos.

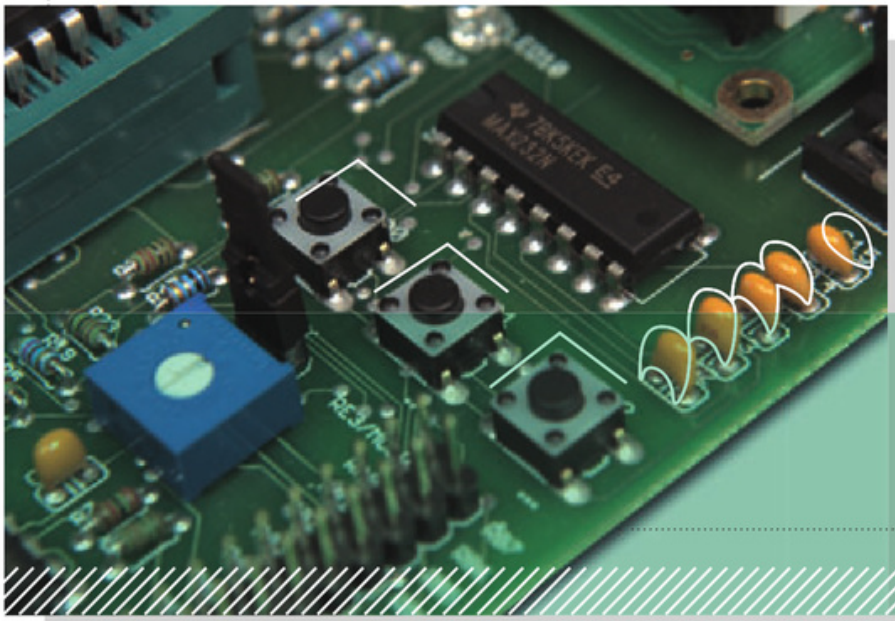


A las unidades que funcionan como interfaz entre el mundo externo y el procesador se las llama **periféricos**. Los periféricos se comunican con el procesador mediante los **buses** de **dirección**, de **datos** y las señales de **control**.
Existen dos formas de transmitir información entre un periférico externo y el procesador: en **paralelo** y en **serie**. La transmisión en **paralelo** utiliza todas las líneas de comunicación del bus de datos, y no requiere realizar ningún sincronismo entre el periférico y el procesador. Por su parte, la que es en **serie** hace la transformación de paralelo a serie y transmite el byte, bit por bit. Este tipo de trasmisión necesita de un sincronismo entre el **procesador** y el **periférico**.
Los periféricos pueden clasificarse de forma general en: periféricos de **entrada** y de **salida**.

Periféricos de entrada

Se ocupan de codificar los mensajes o señales del exterior para que el procesador pueda interpretarlos.

El ejemplo más sencillo de un periférico de entrada es el **teclado**, con el cual un usuario puede introducir un programa o datos. Pero esto no solo se limita al ingreso de datos por parte de un usuario, sino que también es posible recibir datos de una aplicación de control. Los dispositivos de monitoreo, como los **sensores**, son periféricos de entrada, ya que pueden convertir distintas magnitudes, como el calor o la presión, en señales que una computadora sea capaz de leer.



> El pulsador es el ejemplo más simple de un periférico de entrada. Si armamos una matriz de pulsadores, podemos crear un sencillo teclado para ingresar datos.

Periféricos de salida

Permiten observar los resultados arrojados por el procesador de una manera más cómoda que si se presentaran como unos y ceros. La pantalla y la impresora son los periféricos de salida más conocidos, pero también tenemos dispositivos actuadores que afectan de manera mecánica todo lo que los rodea, como los motores y los relés.

Programación de microprocesadores

La programación de microprocesadores puede realizarse entre tres tipos de niveles básicos de lenguajes: en **código máquina**, **ensamblador** y de **alto nivel**.

Lenguaje en código máquina

Es el lenguaje elemental del microprocesador, pero el más complicado de utilizar. Cada instrucción posee **códigos hexadecimales** que son específicos de ese procesador. Esto hace que la programación de las distintas familias de microprocesadores sea incompatible. Solo se trabaja en código máquina con algunos periféricos que disponen de un repertorio determinado de comandos.
Todos los lenguajes superiores al final serán transformados a lenguaje máquina para ser introducidos en la memoria, ya que este es el único lenguaje que entienden los microprocesadores. Pero esta conversión no la realiza el programador, sino que existe un software específico para este fin.
Si se desea programar en **código máquina**, hay que entender previamente a fondo el microprocesador que se va a utilizar, ya que cada bit de cada instrucción tiene un significado concreto y es muy fácil equivocarse.

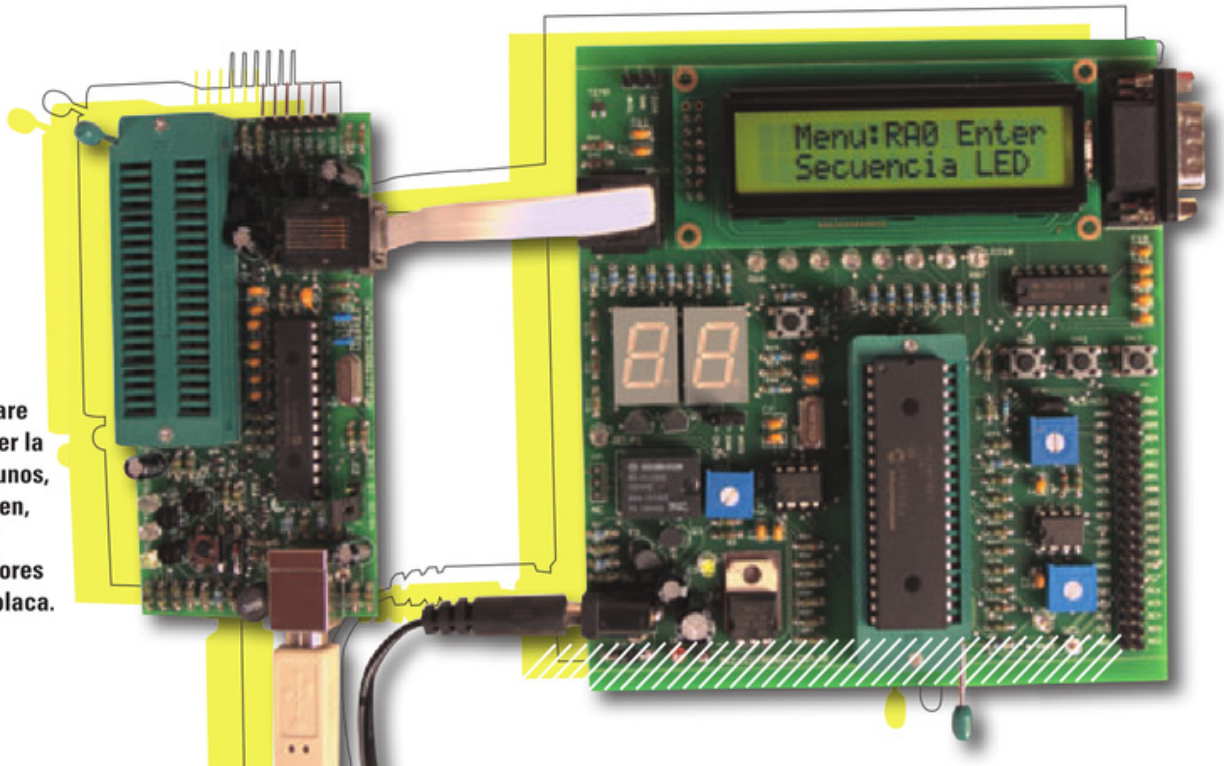
> PARA TENER EN CUENTA

Debido a que cada microprocesador tiene su **assembler** específico, los distintos lenguajes ensambladores de las diferentes familias de procesadores que existen comercialmente no son compatibles entre sí, pese a que emplean instrucciones con idéntico cometido. En este punto de incompatibilidad, se asemejan el lenguaje ensamblador y el lenguaje máquina.

El lenguaje ensamblador utiliza nemotécnicos para las mismas instrucciones en binario.

El lenguaje ensamblador

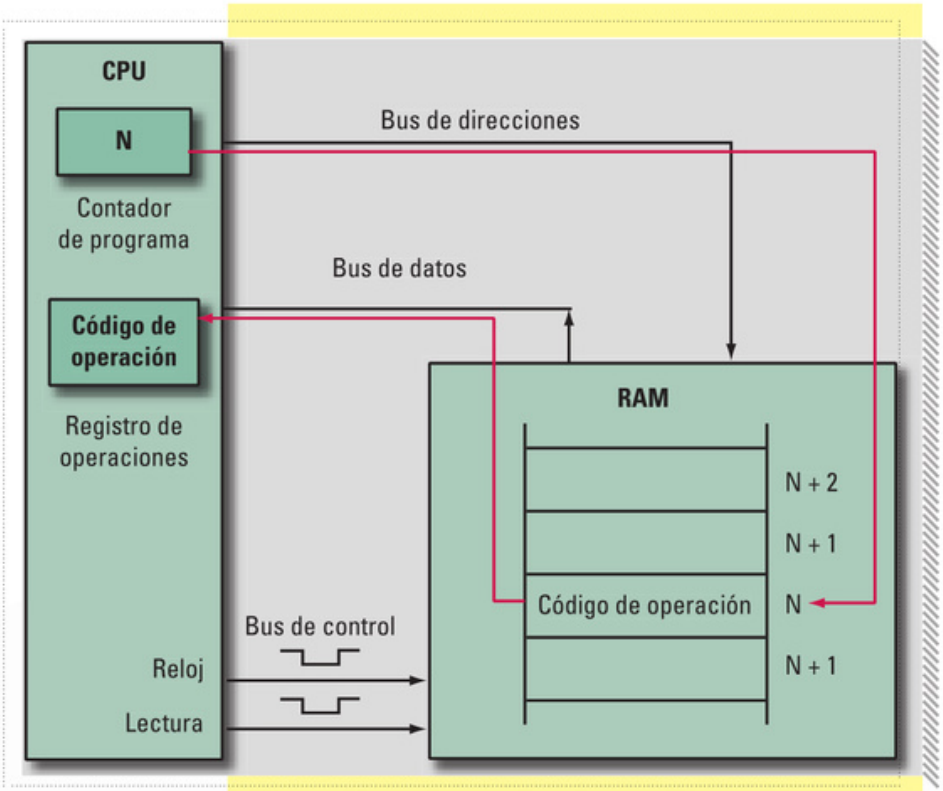
También llamado **ASSEMBLER**, es un tipo de lenguaje intermedio entre los de **alto nivel** y el lenguaje **máquina**. Cada microprocesador tiene su propio lenguaje assembler, que está en relación directa con su estructura. Este lenguaje usa las mismas instrucciones que posee el microprocesador, solo que el programador no emplea su correspondencia en hexadecimal, como en el lenguaje máquina, sino que utiliza los **nemotécnicos** de dichas instrucciones.



> En muchos procesadores se necesita hardware adicional para hacer la programación. Algunos, como el de la imagen, pueden programar los microcontroladores sin sacarlos de la placa.

Secuencia de ejecución de un programa

Para el manejo del programa, el microprocesador dispone de dos registros de importancia: el **contador de programa** (**PC**) y el **registro de instrucciones** (**IR**). Para entender el proceso de interpretación del programa, veamos los pasos que se realizan para decodificar una instrucción.
Al comienzo de cada instrucción, lo primero que debe hacerse es la lectura del **primer byte** de la instrucción; para esto, el **PC** tiene que direccionar a la posición de memoria en la que se encuentra. Mediante señales de transferencia en el **bus de control**, el byte entra en el microprocesador por el **bus de datos** al registro de instrucciones. Allí es interpretado por el decodificador de instrucciones, mientras el **PC** es incrementado en una cuenta. Si la instrucción es de más bytes, una vez interpretada por el decodificador, se abrirá de nuevo el bus de direcciones y de datos para tomar el segundo byte en la posición de memoria indicada por el **PC** ya incrementado.
El proceso se repetirá otra vez si la instrucción es de tres bytes.
Así, según la instrucción sea de uno, dos o tres bytes, la velocidad o tiempo de ejecución será menor o mayor, respectivamente.
Una vez que los bytes de la instrucción han entrado en el microprocesador, este procede a ejecutarla; tras esta operación, vuelve a tomar el byte cuya dirección esté apuntada por el **PC**. Este proceso se repite hasta llegar a la última instrucción.



> Secuencia que lleva a cabo el procesador para buscar una instrucción en la memoria. El PC apunta a la dirección donde se encuentra la instrucción, y la memoria coloca el código de operación en el bus de datos.

Los microprocesadores realizan cuatro operaciones básicas para ejecutar una instrucción.



> EL CICLO DE MÁQUINA

Los microprocesadores realizan una serie de operaciones básicas: búsqueda de la instrucción, decodificación, ejecución y almacenamiento de los resultados. Estas cuatro operaciones conforman el ciclo de máquina. Todas se encuentran sincronizadas con un reloj general (clock del sistema). El tiempo que dura un ciclo de máquina nos da idea de cuántas instrucciones puede ejecutar un microprocesador en un segundo (MIPS).

Lenguajes de alto nivel

Se llaman de alto nivel porque su sistema de programación está a la altura misma del lenguaje conceptual, matemático y de organización del propio hombre. El desarrollo de los lenguajes de alto nivel fue necesario como consecuencia de la adaptación de la máquina al hombre. Esto trajo muchas ventajas que hicieron que este tipo de lenguaje de programación se impusiera rápidamente. Por un lado, al ser un lenguaje próximo al del hombre –que, en definitiva, es quien tiene que programarlos–, permite la reducción de los costos de software, así como también el tiempo de desarrollo. Otras ventajas son su facilidad de aprendizaje, la posibilidad de realizar programación estructurada y el hecho de que para usarlo no es imprescindible tener conocimiento del hardware.
Entre los lenguajes de alto nivel, los más conocidos son:

- **MATLAB**, *MATrix LABoratory* (laboratorio de matrices): es un lenguaje diseñado para hacer cálculos matemáticos, y empleado en el mundo científico y técnico.
- **COBOL**, *Commom Business Oriented Language* (lenguaje orientado hacia aplicaciones comerciales y de gestión): es un lenguaje para uso específico en gestión por tener poca capacidad de cálculo, pero con potencia en el manejo de datos.
- **BASIC**, *Beginners All Purpose Symbolic Instruction Code* (código de instrucción simbólica universal para principiantes): fue desarrollado por la Universidad de Dartmonth (EE.UU.) para los estudiantes que se inician en el mundo de la programación.
- **C/C++**: es un lenguaje extremadamente poderoso y eficiente, que nos da la libertad de realizar casi cualquier tarea con una computadora. Es muy popular entre los desarrolladores de software profesional. ■

> PARA TENER EN CUENTA

Los lenguajes **de alto nivel** también fueron pensados para eliminar la incompatibilidad entre los de **bajo nivel** y los distintos sistemas de procesadores. Sin embargo, esto no es del todo cierto, ya que existen algunas diferencias dentro de un mismo lenguaje de alto nivel con los distintos sistemas que no proporcionan total compatibilidad. Lo cierto es que un programa en lenguaje de alto nivel debe ser traducido a **código máquina**, para lo cual se utilizan programas intérpretes o compiladores.



> En procesadores con alto poder de cálculo, como los DSP (micros con procesamiento digital de señales), resulta muy difícil y costoso programar en assembler.

HISTORIAS DE MICROPROCESADORES

Mucho tiempo ha pasado desde el desarrollo del primer microprocesador, y aún varios de los conceptos originales siguen vivos en los microcontroladores más modernos.

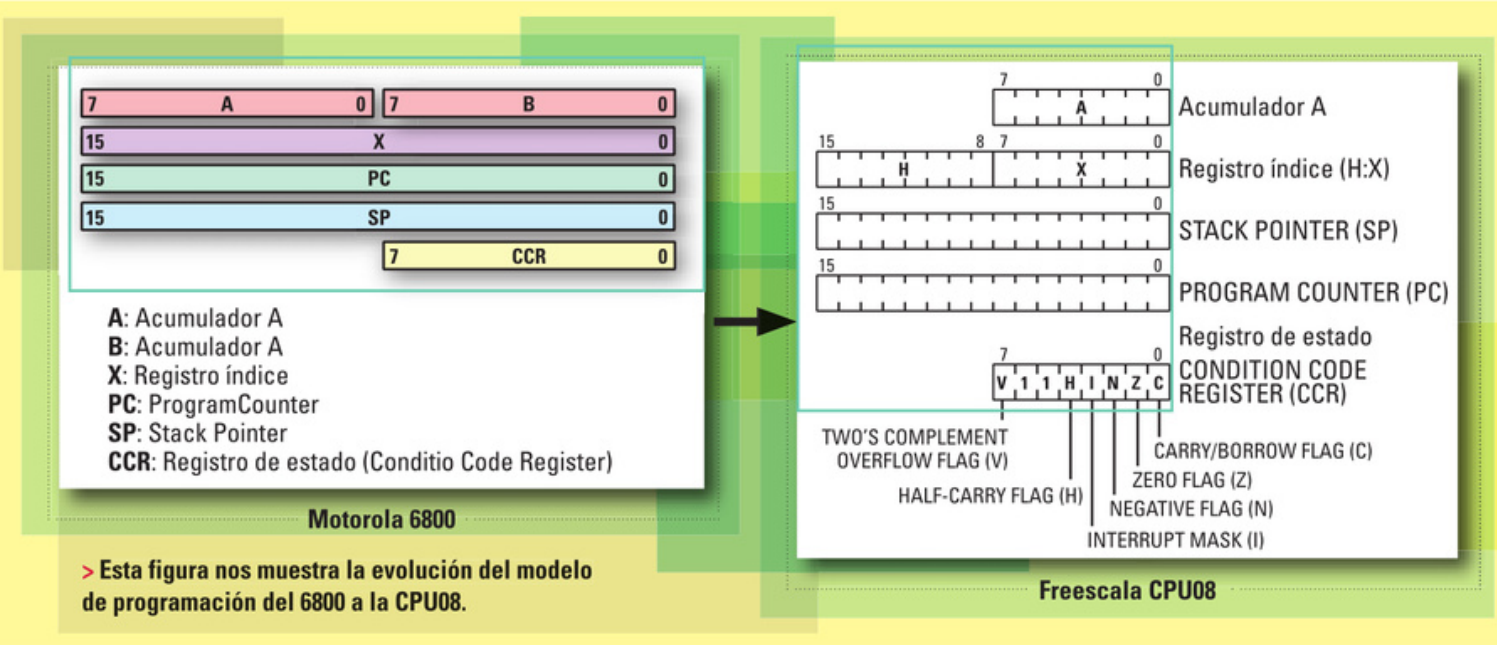
Hasta el momento hemos conocido la estructura interna y algunos conceptos generales que comparten los microprocesadores y microcontroladores, al menos los de 8 bits. Estos conocimientos son la piedra fundamental sobre la que se edifica toda la estructura que contiene a los microcontroladores más modernos. Aun con sus mayores diferencias, todos conservan una historia y ciertos puntos en común. Comprendiéndola, podremos abordar indistintamente cualquiera de ellos para nuestro trabajo.

De 8080 a Rabbit 5000

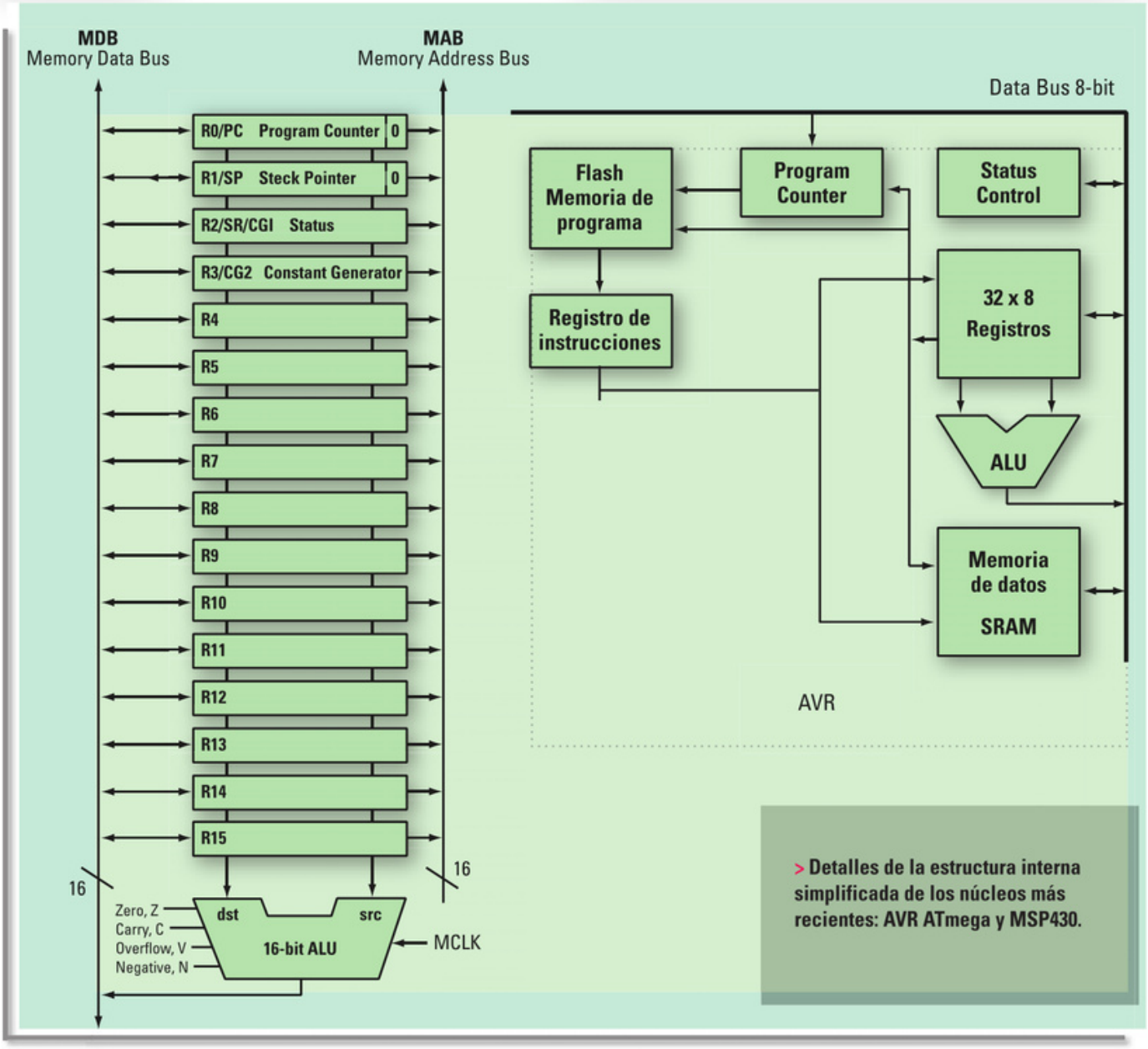
Como sabemos, el primer microprocesador exitoso fue el **8080** de Intel, sobre el que se basó el **Z80** de la empresa Zilog. Con ellos, nació uno de los primeros sistemas operativos de amplia distribución: el **CP/M**.
La característica distintiva de estos micros fue su gran cantidad de registros y un variado set de instrucciones orientado al procesamiento de datos. El **Z80**, en particular, introdujo la innovación del **set alternativo de registros**, para acelerar el cambio de contexto en interrupciones, tema que desarrollaremos más adelante.
Otra novedad fue un área especial para el acceso a puertos de entrada/salida (I/O) fuera del mapa de memoria de datos. Sobre esta plataforma se desarrolló **Rabbit**, que estudiaremos en la clase 17.

Del 6800 al HCS08

El primer microprocesador exitoso de **Motorola**, hoy Freescale, fue el **6800**, contemporáneo y competidor del **8080** e, incluso, del **Z80**. Las líneas de 8 bits de mayor éxito de **Freescale** en la actualidad son las **HC08** y **HCS08** (**S08**),



> **Esta figura nos muestra la evolución del modelo de programación del 6800 a la CPU08.**



> Detalles de la estructura interna simplificada de los núcleos más recientes: AVR ATmega y MSP430.

basadas en la **CPU08**, que, a su vez, es descendiente de la original **6800**. La característica distintiva de estos micros es un modelo de programación simple y compacto, con gran cantidad de instrucciones cortas y eficientes, orientadas a la resolución de tareas simples.

El 8051

En la figura de la página 10 se muestra en detalle la arquitectura **MCS51**, perteneciente al **8051**, un microcontrolador diseñado por Intel que ha pasado a ser el estándar de facto en el mundo de los microcontroladores, y que es modificado y producido por decenas de fabricantes. Su particularidad distintiva es que puede **procesar bits individualmente**, dentro de un área de **128 bits** (32 bytes).

MSP430 y AVR

Dos cores de relativamente reciente aparición son, por un lado, el **MSP430**, de 8/16 bits, fabricado por Texas Instruments; y, por el otro, el **AVR**, de Atmel. Ambos se caracterizan por tener una gran cantidad de registros del procesador, fuera del mapa de memoria de datos, además de un poderoso set de instrucciones. El **MSP430** permite operar en **8** o **16** bits sobre los registros o la memoria. AVR incluye, además, un área de **I/O**. Ambos núcleos permiten que **cualquier registro o posición de memoria sea fuente o destino de casi cualquier operación**.

Microchip PIC

Otro core de gran repercusión es el PIC, de Microchip, elegido por la mayoría de los aficionados en el mundo, cuya historia conoceremos más adelante. Él será el protagonista de la clase. Se distingue por la simpleza y eficiencia del core, y por tener un muy compacto set de instrucciones de rápida ejecución, con acceso a toda la memoria de datos como uno de los operandos y posible destino de muchas de las operaciones.
Éstos son solo algunos de los núcleos de 8 bits de algunos de los fabricantes de mayor repercusión. Por supuesto que existen muchas más alternativas. ■

MICROCOMPUTADORAS: ¿DE QUÉ SE TRATAN?

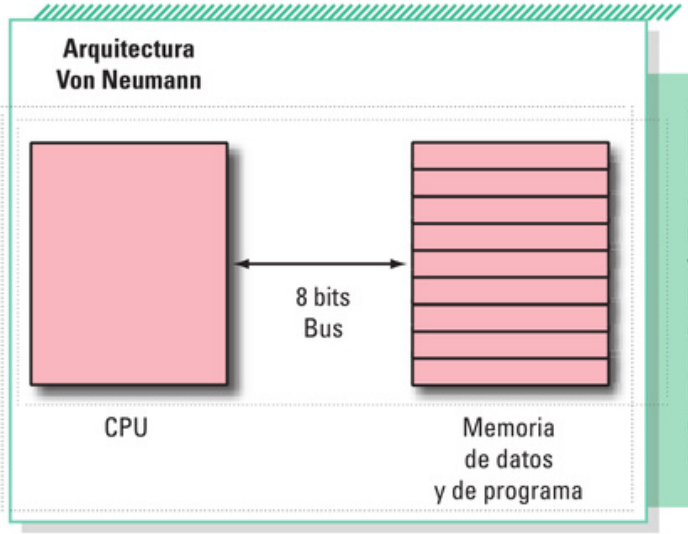
Se encuentran en todos nuestros electrodomésticos, alarmas y automotores, entre otros equipos, donde aplican inteligencia a todos los aparatos que nos rodean.

El avance de las escalas de integración permitió integrar cada vez más transistores por unidad de superficie. El desarrollo de la tecnología **VLSI (muy alta escala de integración)**, que produjo los microprocesadores, pronto permitió introducir en un solo chip todo un sistema mínimo. Esto dio nacimiento a la microcomputadora, actualmente conocida como **microcontrolador**. Dentro de ellas encontramos una **CPU**, una **memoria de programa**, una **memoria de datos**, el **circuito de reset** y el **circuito oscilador**, además de los **puertos de entrada/salida**, también conocidos como **PORTS I/O**. Las microcomputadoras o microcontroladores nacieron a mediados de los 80 y rápidamente ganaron mercado, al desplazar a los sistemas mínimos desarrollados con microprocesadores en el campo del control industrial. Los **microcontrolad ores**, debido a su muy bajo costo, alta inmunidad al ruido eléctrico y pequeño tamaño, produjeron la revolución microcontrolada, que desplazó a toda la **lógica cableada** (utilizada en la electrónica industrial) y a la **lógica programada**. Es en este campo donde se los bautizó con el nombre de microcontroladores y se desechó el de microcomputadoras. A partir de los **90,** los microcontroladores invadieron la electrónica de consumo, brindando a los electrodomésticos y a todo tipo de sistema electrónico de consumo la capacidad de inteligencia y conectividad. El mundo actual está rodeado de microcontroladores; desde nuestros celulares, sistemas de alarmas y lavarropas, hasta las computadoras de a bordo de los automóviles. Sin ellos, nuestro mundo actual no existiría.

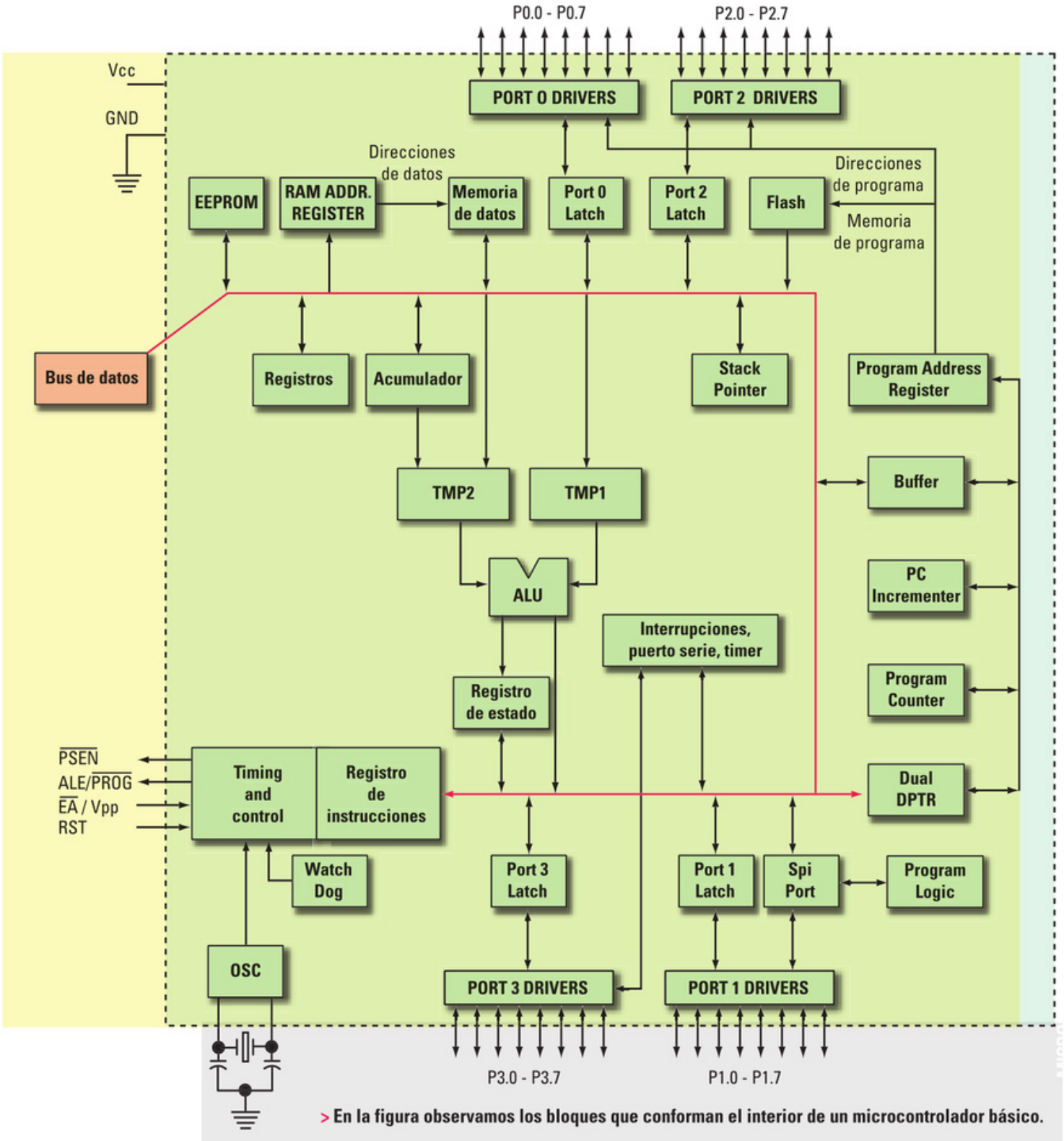
Arquitectura interna de los microcontroladores

Los microcontroladores están formados por varios bloques. Dentro de ellos existe, como elemento principal, la **CPU**, que se interconecta con sus periféricos para formar lo que se conoce como arquitectura interna. Esta puede ser de dos tipos: **Von Neumann** o **Harvard**.

La arquitectura Von Neumann fue suplantada por los microcontroladores con arquitectura Harvard.



> En el diagrama podemos ver la conexión entre la CPU, la memoria de datos y la de programa, utilizando el bus de datos para las instrucciones y los datos. Esto genera un cuello de botella.



> En la figura observamos los bloques que conforman el interior de un microcontrolador básico.

Arquitectura Von Neumann

Fue desarrollada en **1949** por el profesor **John Von Neumann** e implementada para la construcción de las computadoras como la **ENIAC** (nombre de la primera computadora electrónica). Según esta arquitectura, existe un bus de datos que liga la **CPU** con la **memoria de datos** y de **programa,** por el cual viajan datos e instrucciones. Este concepto fue muy útil en las primeras décadas de las computadoras, pero, al incrementarse la cantidad de datos por procesar, la velocidad de procesamiento se redujo. Rápidamente, la arquitectura se saturó, ya que el **bus** de datos debía compartirse con los datos y las instrucciones, lo que generaba un cuello de botella. Entonces, se presentaron varios problemas en la arquitectura, como mencionamos a continuación.

Por un lado, el ancho del bus de datos era de **8 bits** y, como por él viajan

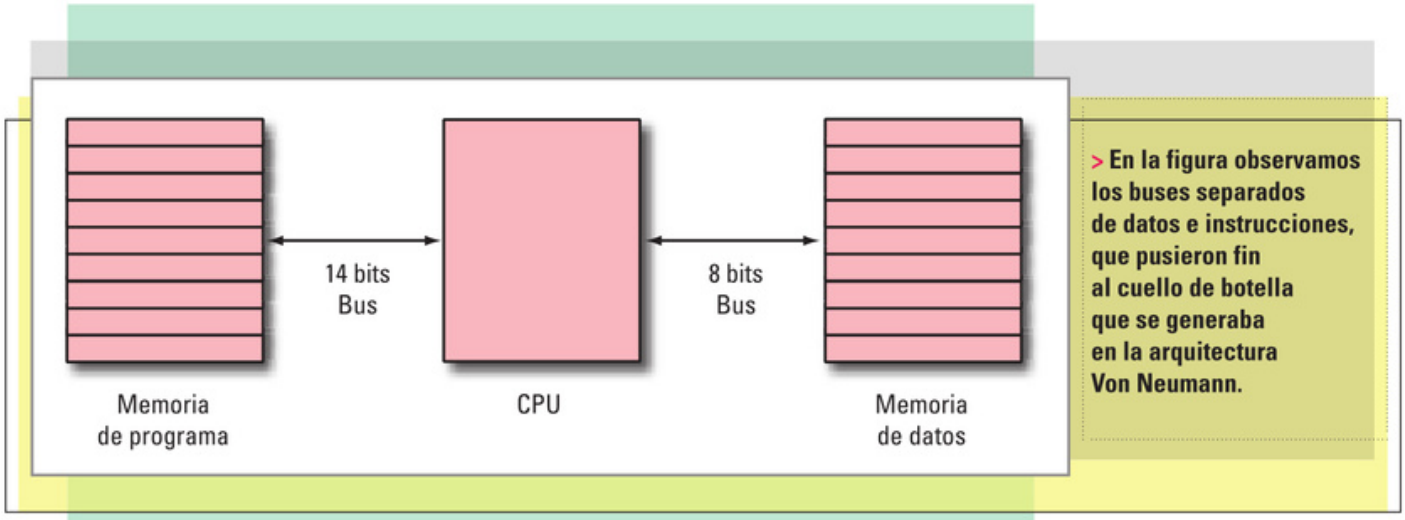
los datos y las instrucciones, el ancho de los datos limitaba el ancho de las instrucciones. Como consecuencia, las instrucciones con más de **8 bits** debían ser enviadas en varias partes, lo cual hacía que el sistema resultara lento.
Otro conflicto es que nunca se sabía cuánta memoria de programa se usaba, pues esto depende del ancho en **bytes** que tengan las instrucciones, lo cual es muy variable.
Esta arquitectura caducó a finales de los **80** y fue suplantada por los microcontroladores **Harvard**.

Arquitectura Harvard

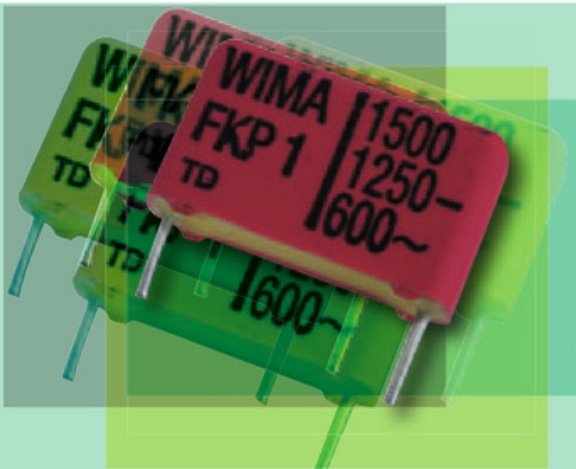
Fue desarrollada en 1970 para solucionar los problemas de velocidad de procesamiento que presentaba la arquitectura **Von Neumann**.

Los microcontroladores son una pequeña computadora inyectada en un chip.

Esta arquitectura conectó la **CPU** hacia su memoria mediante dos buses distintos: uno de datos y otro de instrucciones. De este modo, el ancho del bus de instrucciones no está limitado por el de datos, y el procesador puede recibir instrucciones por caminos diferentes, aprovechando el tiempo del ciclo de máquina.
El concepto y nombre de la arquitectura deviene de la computadora **MARK1**, construida en la Universidad de Harvard en 1944.
En 1975, una compañía americana denominada **General Instruments** formó una división especial dedicada al desarrollo de memorias y microprocesadores: **GI Microelectronics**. Esta tomó el concepto de la arquitectura Harvard y lo materializó en su primer microcontrolador denominado **PIC1650**, e introdujo mejoras en el concepto de la arquitectura. Colocó dentro del chip una pila de instrucciones de **dos niveles**. Esto dio la posibilidad de buscar y ejecutar una instrucción en el mismo ciclo de máquina. Se la denominó **arquitectura Harvard modificada**.



> En la figura observamos los buses separados de datos e instrucciones, que pusieron fin al cuello de botella que se generaba en la arquitectura Von Neumann.



> ARQUITECTURA HARVARD

Emplea dos buses de acceso, uno para datos y otro para programa. Esto optimiza el funcionamiento a costa de duplicar la cantidad de memorias. No es posible guardar constantes con el programa. La versión modificada introduce acceso a la memoria de programa como datos, resolviendo este inconveniente. Los compiladores para lenguajes de alto nivel, como C, deben introducir directivas para elegir área.

En 1985 **GI** vendió Microelectronics a un grupo inversor denominado Ventura, que reflotó el proyecto del **PIC1650** y rebautizó la compañía con el nombre de **Microchip**.
Esta desarrolló rápidamente una serie de microcontroladores basados en la arquitectura del PIC. Con el tiempo, otras firmas adoptaron el modelo de Microchip para sus núcleos.

Concepto de una computadora

La computadora es un circuito digital capaz de procesar información binaria. En este circuito encontramos un microprocesador, una memoria del tipo **no volátil** (ROM, EPROM, EEPROM, Flash), una memoria **RAM**, un circuito **oscilador**, un **circuito de reset** y los **puertos de entrada/salida**.
La memoria no volátil es conocida como memoria de programa, ya que en ella se almacena el que hace funcionar a la computadora.
La **RAM** es usada por la computadora para guardar los resultados de procesamientos de datos internos o los que provienen del exterior.
Por su parte, los **puertos I/O** (entrada/salida) se usan para intercambiar información y controlar todo el entorno externo a la máquina.
Todos estos elementos se encuentran en forma discreta (es decir, encapsulados en chips individuales) y montados sobre un **PCB**. A este conjunto se lo denomina **sistema mínimo.**

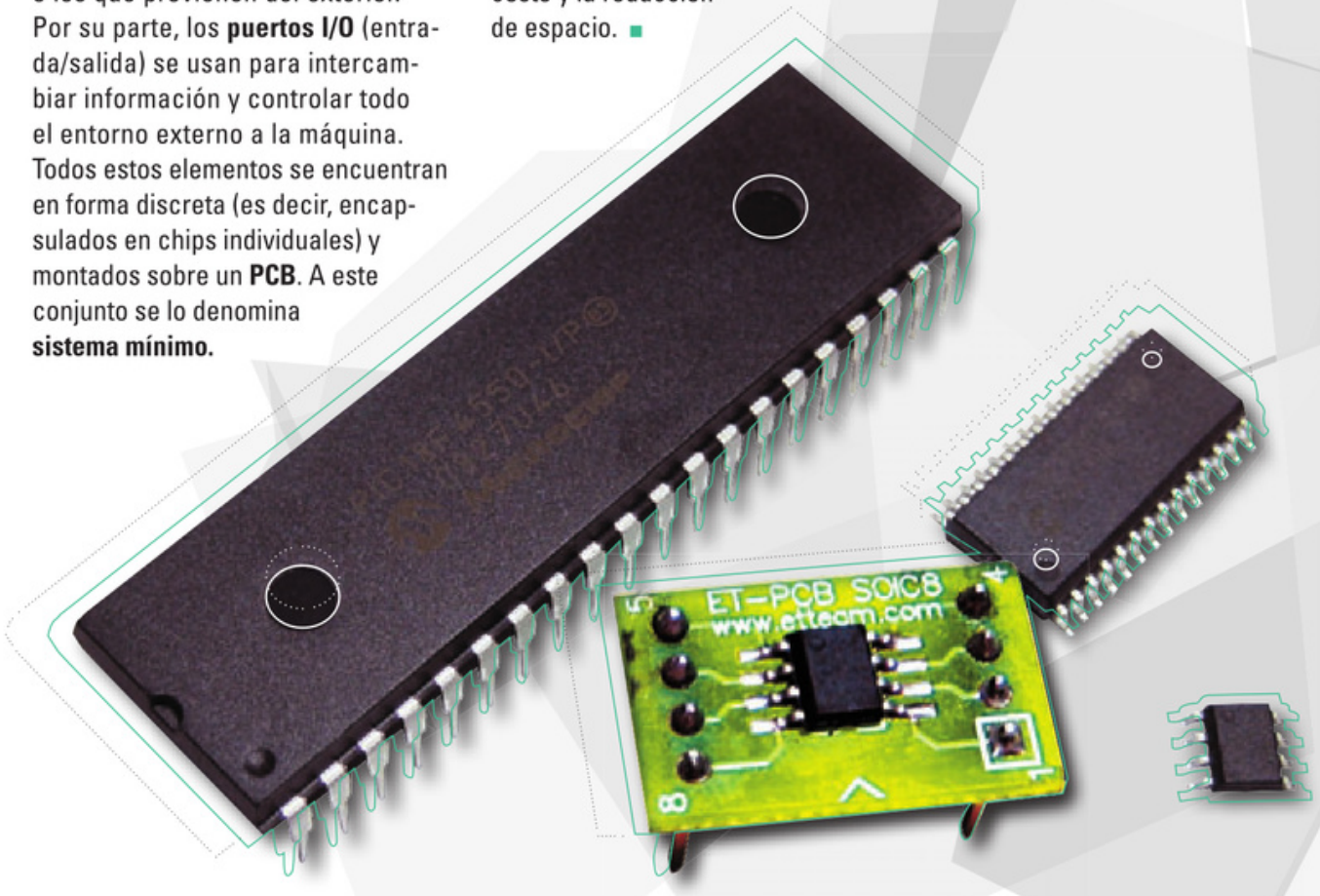
Las computadoras miden su potencia de procesamiento en función de la cantidad de instrucciones que pueden ejecutar por segundo, medida en una unidad denominada **MIPS**.

Microcontroladores y microprocesadores

No debemos confundir los microprocesadores con los microcontroladores. Los primeros, simplemente, son la **unidad central de procesamiento**. No incorporan puertos para control de periféricos, ni memoria de programa ni tampoco memoria de datos. Están especialmente diseñados para procesar grandes cantidades de datos y son muy susceptibles al ruido eléctrico.
En cambio, los **microcontroladores** son una pequeña computadora inyectada en un chip. Están diseñados, principalmente, para el control industrial y no para el procesamiento de grandes cantidades de datos. Su principal ventaja radica en la alta inmunidad al ruido, el bajo costo y la reducción de espacio. ■

> EL MICRO 6502

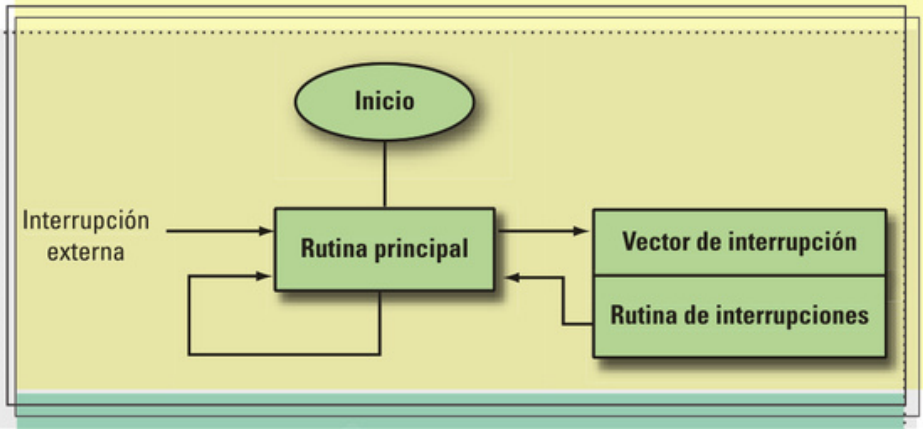
El microprocesador 6502 fue desarrollado por la compañía Rockwell. Este micro era una variante del 6800, fabricado por la firma Motorola, con la diferencia de que el 6502 poseía solo un registro de trabajo. A partir de este se fabricaron la TK2000 de Microdigital y el Apple 2e. Por su parte, Commodore desarrolló su Commodore 64 a partir de una variante mejorada del 6502, denominada 6510 (el cual incluía un PORT I/O auxiliar).



LAS INTERRUPCIONES

Con las interrupciones podemos hacer que determinados eventos que ocurren en el hardware cambien la rutina del software. Veamos de qué se trata.

Para entender qué es una interrupción, recurriremos a un ejemplo muy sencillo y que se presenta bastante a menudo en nuestros hogares. Cuando enviamos un archivo a la impresora, la rutina principal del programa se encarga de mandar los caracteres al buffer de este equipo. Si la impresora se queda sin papel, el proceso se detiene, y se despliega un mensaje en la pantalla de la PC para advertirnos al respecto. Como podemos observar, el programa principal se encarga de enviar los caracteres del archivo a la impresora. Sin embargo, si el sensor de falta de papel se activa, se interrumpe la impresión. Este evento hace que el procesador pase a procesar un programa que atienda la interrupción, el cual despliega el mensaje en pantalla. Una vez que recargamos el papel, el sensor se desactiva, la señal de interrupción desaparece, y el procesador vuelve a ejecutar la rutina principal. La interrupción es un sistema que provoca un salto a una subrutina,

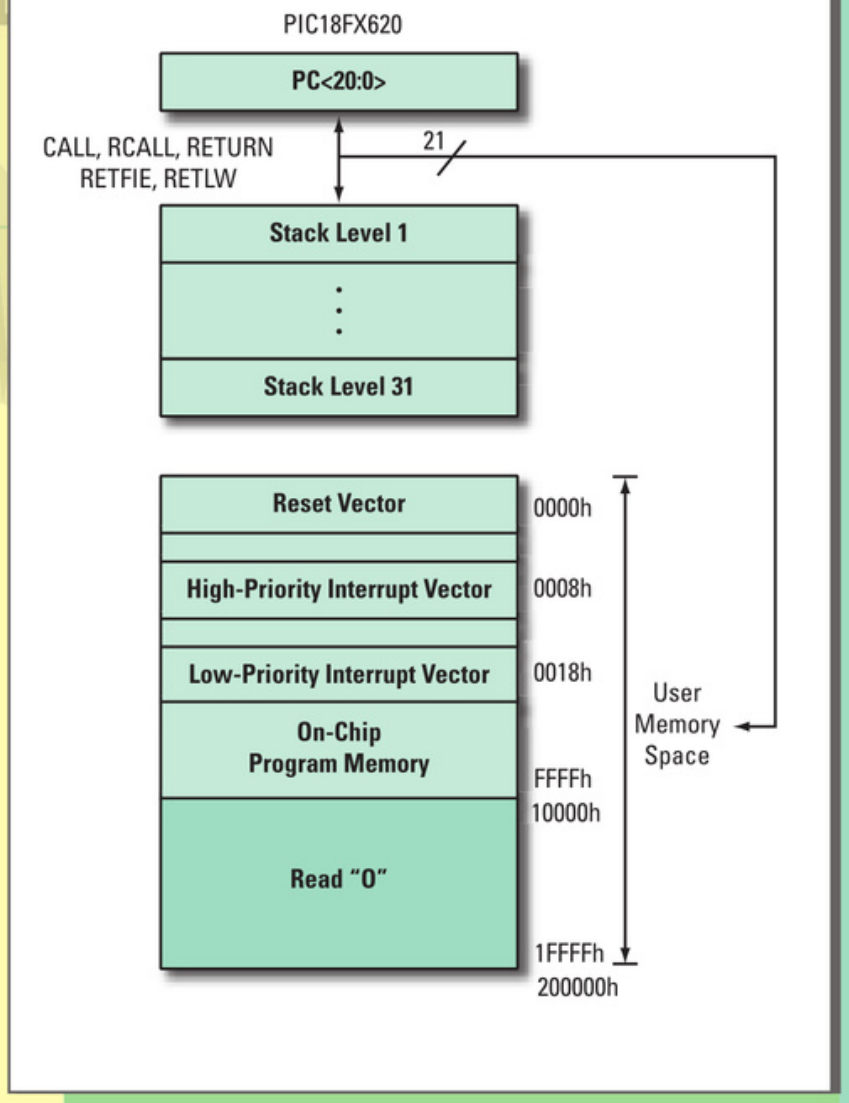


> En la figura, podemos observar el proceso de una interrupción de forma gráfica.

La interrupción provoca un salto a una subrutina, pero disparada por un evento del hardware.

> INTERRUPCIONES

El concepto nace de la necesidad de procesar eventos que, al ocurrir, requieren una atención inmediata ya que no es conveniente esperarlos. Según el procesador, se transfiere la ejecución a una posición fija o indicada por el contenido de una posición o tabla. Aquí se salva el contexto (si el procesador no lo hace), se opera y se devuelve el control.



> Aquí observamos la posición de memoria de programa de los vectores de interrupción en un microcontrolador PIC18F.

pero disparada por un evento del hardware. Se diferencia de los saltos a subrutina generados por el software, como el producido por la instrucción CALL, en lo siguiente:

- La interrupción es atemporal; el microcontrolador nunca sabe cuándo va a ocurrir.
- Cuando la interrupción ocurre, el procesador abandona el programa que está ejecutando y pasa a procesar una rutina que se encuentra a partir de una posición de memoria fija, denominada vector de interrupción.
- En la interrupción se salva el contenido interno de los registros más importantes del procesador, de modo que luego, al volver al programa que se estaba ejecutando, se prosiga sin alteraciones.

Tipos de interrupciones

Las interrupciones pueden dividirse en internas o externas, y en enmascarables o no enmascarables.
Las internas son disparadas por el hardware interno del microcontrolador, por ejemplo, el conversor analógico/digital, los temporizadores, etc. Las externas son disparadas externamente mediante la aplicación de un pulso o un estado sobre un pin del microcontrolador, denominado INT (interrupción). Pero más allá de si la interrupción es interna o externa, esta puede ser del tipo enmascarable o no enmascarable. Las primeras necesitan tener activado un bit de habilitación para generarse, aunque hayan sido solicitadas; en tanto que las segundas suceden, sí o sí, no bien ocurre el evento de la interrupción.

El vector de interrupciones

Como hemos visto, cuando la interrupción se genera, el procesador pasa a procesar la rutina que se encuentra a partir de una posición de memoria fija, conocida como vector de interrupciones. En los microcontroladores puede existir más de un vector de este tipo, uno para la interrupción enmascarable y otro para la no enmascarable.
En otros microcontroladores, puede ocurrir que cada dispositivo del hardware que interrumpe tenga su propio vector, o que haya vectores distintos según la prioridad que hayamos elegido para la interrupción. Todo depende del microcontrolador que manejemos. ■

PERIFÉRICOS EN MICROCONTROLADORES

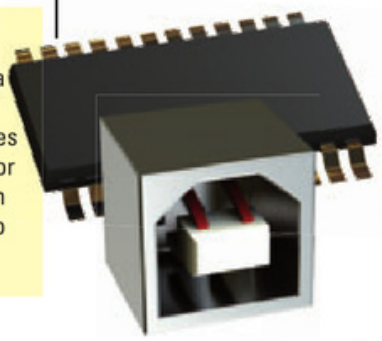
En esta infografía les mostramos los módulos de hardware internos de un PIC16F876A y una idea de conexión de algunos de los periféricos -vzmás utilizados en aplicaciones con microcontroladores.

MAX 232 D

TIP

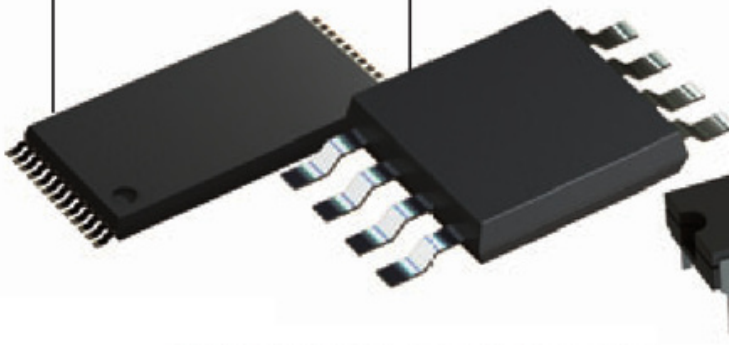
El chip FT2232D de FTDI para conectividad USB 2.0 Full-Speed, permite interfaces con nuestro microcontrolador del tipo SPI, I2C ó mismo con un módulo asincrónico como RS232

■ FT2232D



■ AT45DB041B

■ FM24CL64





Sensor de temperatura

Universal Synchronous Asynchronous Receiver/Transmitter (USART/SCI en Microchip).

Para implementación de comunicaciones asincrónicas seriales como RS232. Generador de Baud-rate configurable: Ej.: 9600 ó 19200 baudios etc. Manejan 8 ó 9 bits de datos, configuración de bits de parada, flags de errores, etc.

Módulo Serial Sincrónico Multipropósito (MSSP en Microchip): SPI / I2C

Módulo HW para interfaces sincrónicas seriales, encontrada en diversas arquitecturas. Permite configurar interfaces SPI e I2C en modos Master y Slave. Ejemplos: AT45DB041B Flash Eeprom SPI de Atmel y FRAM FM24CL64 I2C de Ramtrom.

Conversor A/D de 10 bits multicanal

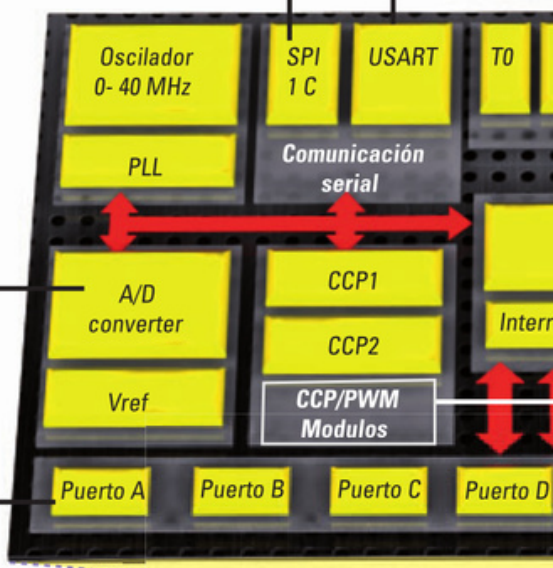
Conversores analógico-digitales. Digitalización de señales de origen analógico como el sensor de temperatura



Display Alfanumérico

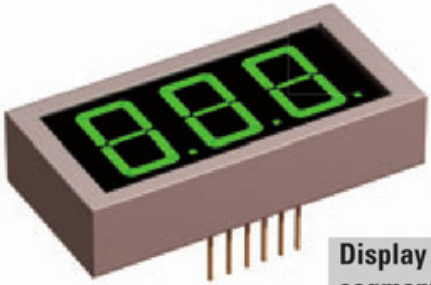
Puertos Paralelos esclavo de 8 bits.

Puertos paralelos de propósito general. Pueden conectarse típicamente a los mismos: leds, switches, botones, displays alfanuméricos, displays 7 segmentos y teclados a membrana. Dependiendo del microcontrolador utilizado, son capaces de manejar corrientes desde los 40uA a 25mA por línea.





Teclado a membrana



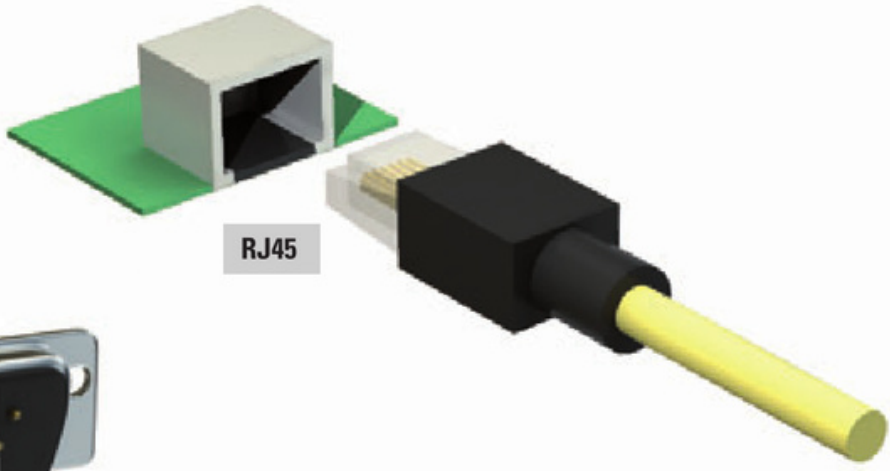
Display 7 segmentos

TIP

Se necesitan chips conversores de niveles como el MAX232 para adaptar las tensiones y corrientes del micro a los del protocolo propiamente dicho.

Driver ■





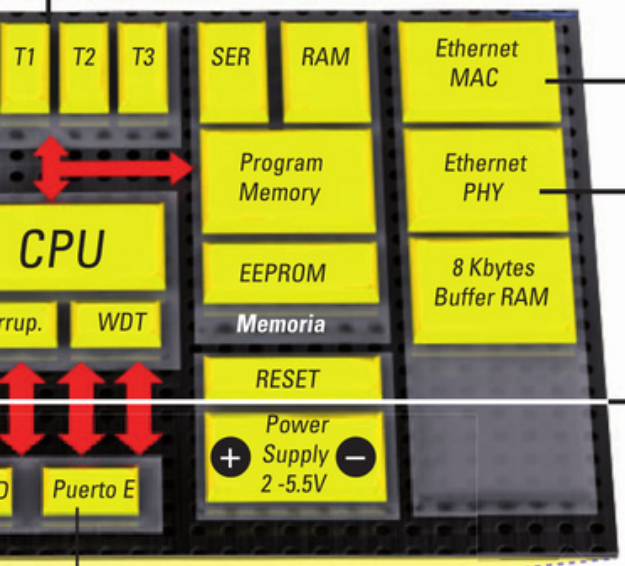
RJ45

Módulos Timer/Temporizadores

Son utilizados para diversas aplicaciones que necesiten trabajar con marcas de tiempo, implementar retrasos (delays) y analizar eventos en forma temporal generando interrupciones. Pueden alimentarse con clk´s externos o internos.
Características PIC16F876A de Microchip:

- Timer 0: Contador/Temporizador de 8 bits con preescaler de 8 bits
- Timer 1: Contador/Temporizador de 16 bits con preescaler.
  Puede ser incrementado durante "sleep", mediante un cristal o reloj externo.
- Timer 2: Contador/Temporizador de 8 bits. Posee preescaler y postcaler.

Circuito interfaz adaptador de líneas y reducción EMI



■ Módulos Ethernet (PIC18F97J60).:

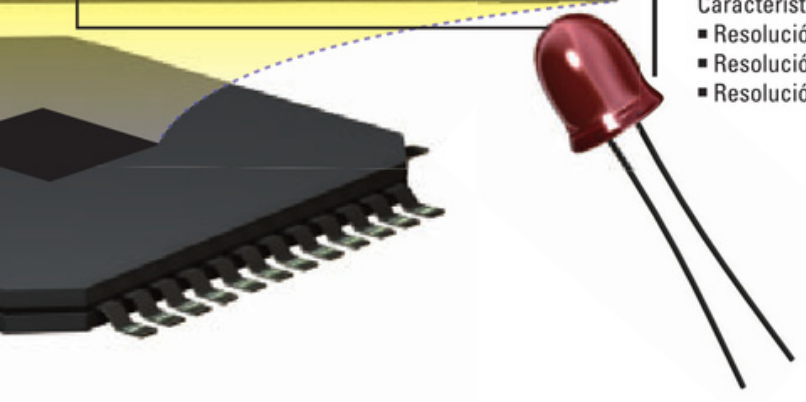
Algunos microcontroladores como el PIC18F97J60 de Microchip, ofrecen controladores Ethernet embebidos. Se encuentran completamente implementados en el HW del micro. Los módulos "Media Access Control" (MAC) y "Physical Layer Transmitter" (PHY) se encargan de ello. Para su interfaz directa a una red Ethernet, sólo se requiere la utilización de 2 transformadores de pulso y algunos componentes pasivos que ayudan a la reducción de EMI (Interferencia Electromagnética).

■ Módulos PWM.:

Utilizados para control de motores pequeños de CC ó para medir ciclos de trabajo de señales como la salida de un acelerómetro. Utilizan los recursos de los módulos "Contadores/Timers" que forman parte de cualquier arquitectura micrcontrolador.

Características PIC 16F876A de Microchip:

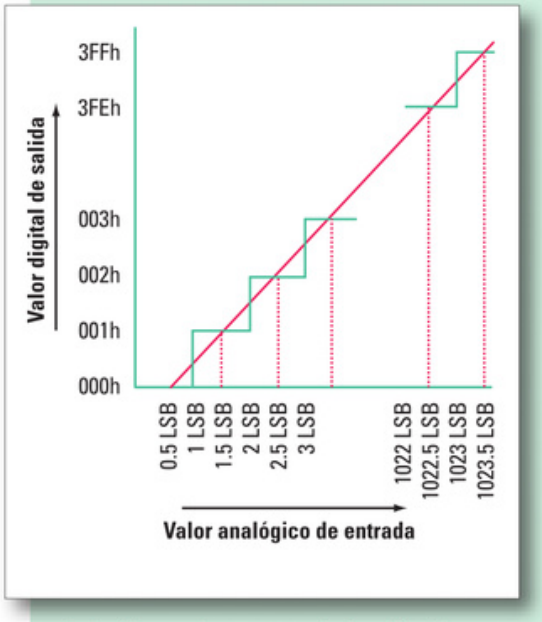
- Resolución Captura 12,5ns máx. y de 16bits
- Resolución Comparador 200ns máx y de 16bits
- Resolución del mod. Pulse Width Modulation es de 10 bits.



Led

CONVERSORES ANALÓGICO-DIGITALES

Los conversores A/D o ADCs son fundamentales para que el microcontrolador pueda leer las variaciones analógicas del mundo real y reaccionar a ellas.



>En la figura observamos la función de transferencia de un conversor de 10 bits.

Los conversores A/D son dispositivos que convierten una variación analógica en formato digital. Son indispensables en los lazos de control cerrados digitales; podríamos decir que son los "ojos" de los microprocesadores y de los microcontroladores. En todo sistema digital de control, existe siempre un **conversor A/D** que se encarga de "traducir" la señal entregada por un sensor de temperatura, presión o fuerza a un valor digital equivalente que pueda procesar el sistema de control digital, ya sea que este se encuentre implementado con micropro-cesadores o con microcontroladores.

Resolución de los conversores A/D

La resolución de un conversor A/D es un parámetro muy importante porque nos indica la precisión que tendrá el dispositivo al realizar la conversión. La señal analógica que ingresa al conversor es dividida en una serie de pequeñas fracciones. Cuanto mayor sea la resolución del conversor, más pequeñas serán las fracciones y, por lo tanto, más aproximada será la conversión al valor real.

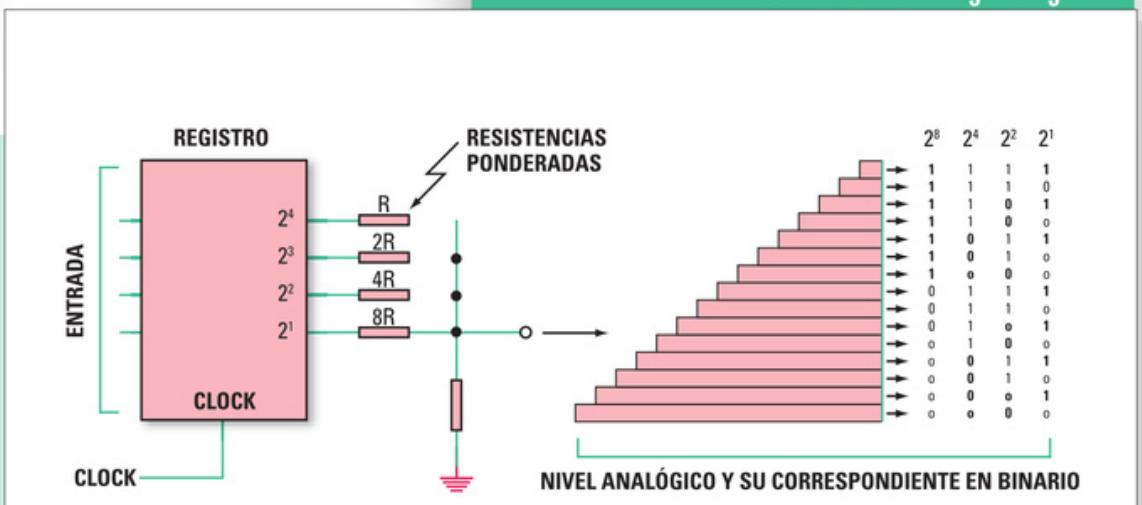
La resolución de un conversor A/D nos indica la precisión que tendrá el dispositivo al realizar el proceso.

Dicha resolución queda determinada por el número de bits que puede procesar el conversor. Por ejemplo, en un conversor de 8 bits, la señal de entrada se divide en **256 fracciones**. Sin embargo, en uno de **16 bits**, lo hace en **65536 fracciones**, y la precisión para el mismo nivel de señal resulta mayor. La fracción se denomina **escalón de conversión** o **rata de cambio**. Cuanto más pequeña es, más precisa es la conversión analógico-digital. Los conversores más comunes son de **8 bits** de resolución, pero también existen de **10 bits**, **12 bits**, **16 bits**, **20 bits** y **24 bits**. Es importante destacar que cuanta mayor precisión tiene el conversor, más lento es; *por lo* tanto, si la señal de entrada varía muy rápidamente, se producirán errores.

> LOS INICIOS DE LOS A/D

Los conversores A/D que se usaron en las primeras PCs para las placas de sonido eran de 8 bits. Su resolución limitada generaba sonidos sintéticos. Luego fueron superados por los conversores de 16 bits, que dieron vida a las tarjetas de sonido como la Sound Blaster. Sin embargo, actualmente contamos con conversores muy superiores, mediante los cuales podemos digitalizar señales con alta precisión.





> En la figura podemos ver un conversor D/A por resistores ponderadores. La dificultad que presenta este método radica en conseguir comercialmente los valores de los resistores.

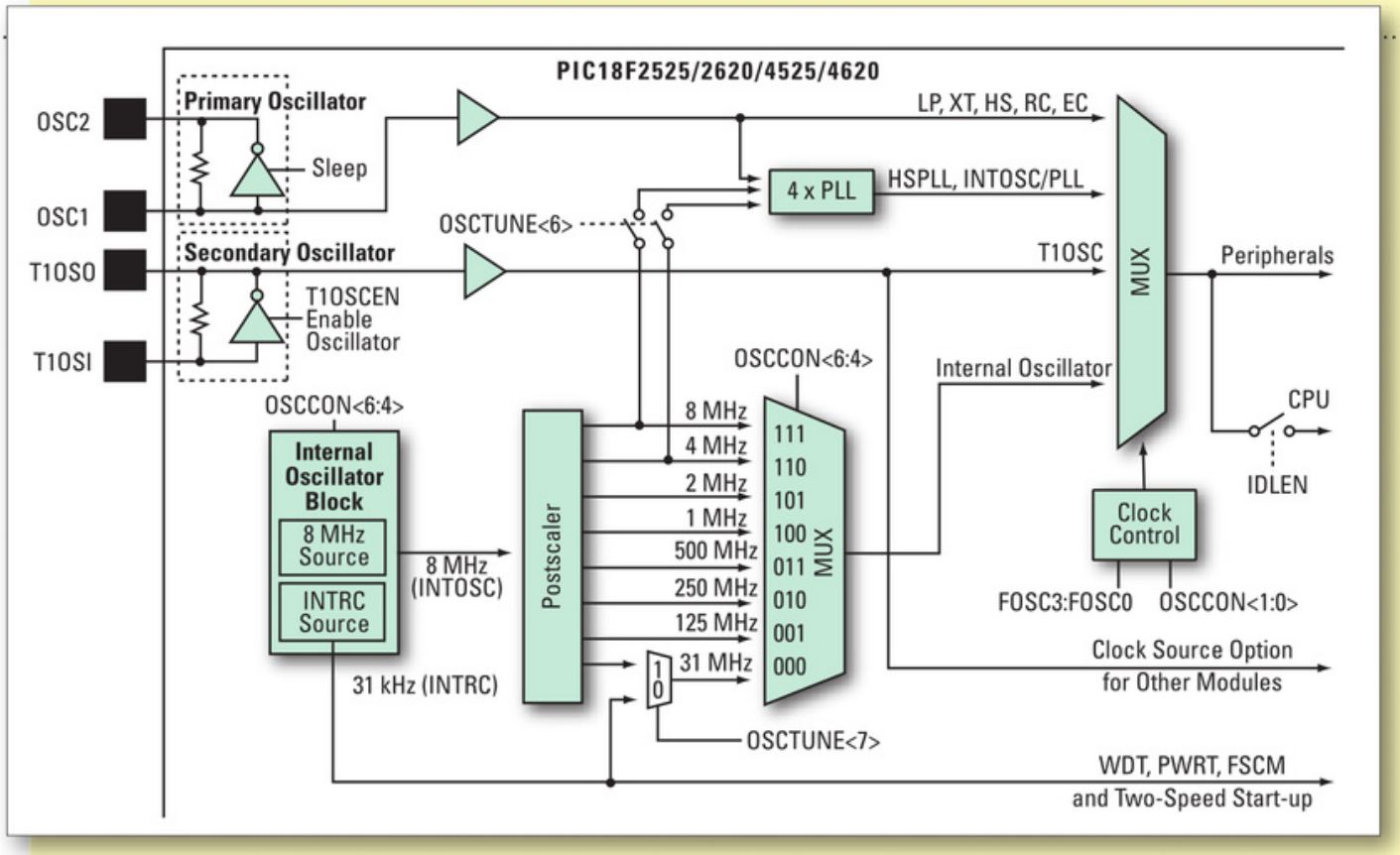
Conversores digital-analógicos

Existen varios métodos para obtener la conversión digital-analógica. Para conseguir pasar un valor **binario** a un valor de tensión **analógico** equivalente, se recurre a una red de resistencias. Esta genera una tensión en función de los niveles binarios que se encuentren en un momento dado en cada uno de los bits que intervienen en el proceso.

La mencionada **red resistiva** es, por lo tanto, la clave de la conversión. Existen dos redes perfectamente diferenciadas que cumplen este cometido: **la de resistores ponderados** y la **red R-2R**. Para explicar este concepto, supongamos que disponemos de un registro de **8 bits**, que puede ser cargado con cualquier valor desde 00 hasta FF.

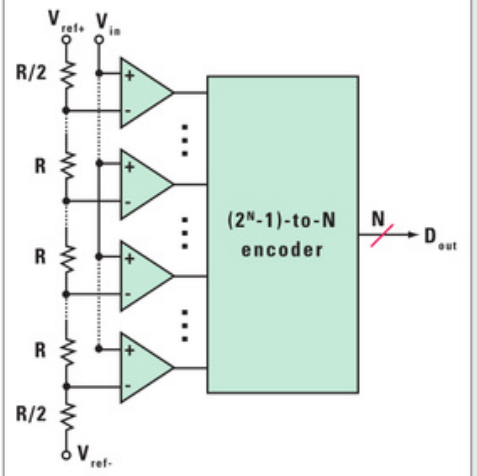
Una **red ponderada de resistores** consistirá en conectar un resistor en cada salida del registro (de allí su nombre "ponderada"), de modo

Para pasar un valor binario a uno de tensión analógico equivalente, se recurre a una red de resistores.



> Este conversor D/A del tipo R-2R utiliza solo dos clases distintas de resistores; sin embargo, su construcción necesita más resistores que el método ponderado.

que los valores de los resistores por conectar desde el bit de mayor peso hasta el de menor peso son R, 2R, 4R, 8R, 16R, 32R, 64R y 128R.
Si una vez conectados estos resistores, se aplican valores binarios de **00** a **FF** en forma consecutiva, se generará una rampa ascendente de 255 escalones, donde cada uno corresponde a un valor binario.
La conversión D/A también puede realizarse mediante otro procedimiento denominado **red de resistores R-2R**. En este sistema, a cada bit que sale del registro se le conecta un resistor de valor **2R** y, en el extremo de este, se coloca un resistor de valor R.
Por ejemplo, si **R** vale **10K**, **2R** vale **20K**. Como se puede apreciar, en esta red el valor óhmico de los resistores R determina el valor de 2R, que es el doble de R (2R = 2 x R).
Este sistema tiene la ventaja de que emplea solo dos valores de resistores distintos. La salida de señal analógica se toma después del resistor en serie del bit de mayor peso.



> **En la figura observamos un conversor estático de 4 bits, mediante el cual ilustramos su constitución interna.**



Conversores analógico-digitales (ADC)

La conversión analógico-digital consiste en transformar valores analógicos en digitales. Es el proceso inverso a la **D/A**. En este caso, a cada valor analógico aplicado en la entrada del conversor le corresponde un valor digital de salida, dentro de los que puede generar el conversor. La precisión del conversor estará dada por la cantidad de bits que formarán al resultado de la conversión. Para obtener la conversión A/Dm existen varios métodos; aquí trataremos solo los más representativos:

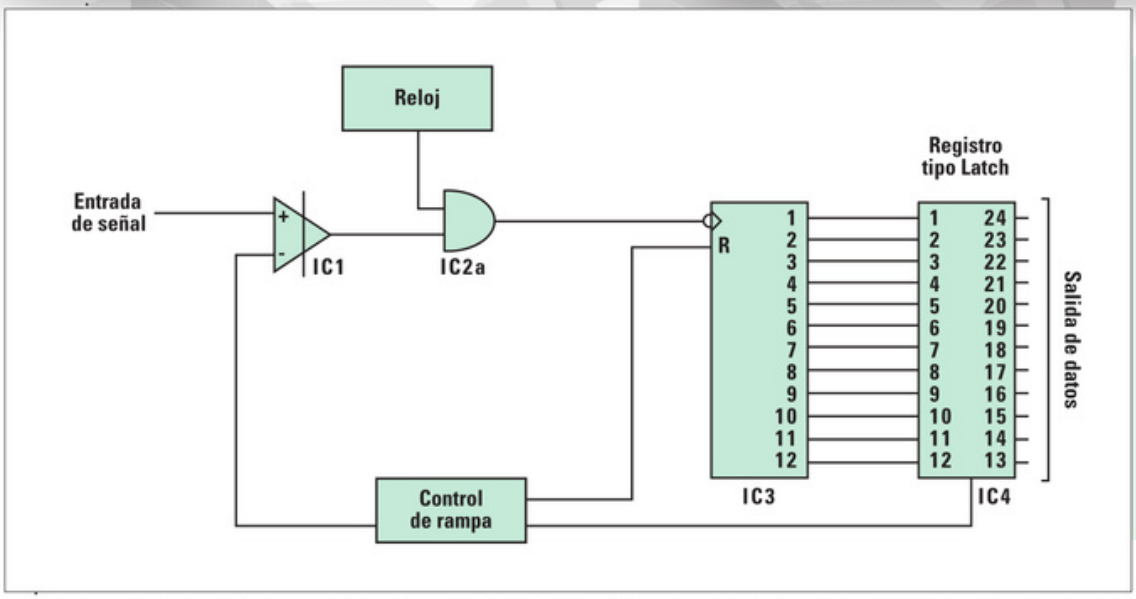
- Conversor estático o flash
- Conversor dinámico o de rampa
- Conversor de doble rampa
- Conversor SAR o por aproximación sucesiva

> VELOCIDAD DE CONVERSIÓN

La principal ventaja que tienen los conversores estáticos o de tipo flash es su velocidad de conversión. Como la señal de entrada se aplica simultáneamente a todos los conversores, el único tiempo de retardo que existe es el de propagación del comparador y el del codificador binario. Por esta causa, los conversores de este tipo son los más rápidos del mercado.

Conversor estático o flash

Este tipo de conversor es el más rápido de todos. Está formado por una cadena de comparadores que toman su voltaje de referencia desde un array de resistores. Como todos los comparadores se encuentran en paralelo, el sistema es conocido como **conversor paralelo**. La salida de los comparadores se aplica a un codificador de prioridad digital, el cual genera un número binario según la entrada que se haya activado. De este modo, cada comparador dispara una entrada, la cual, a su vez, genera un código binario en la salida del codificador.
El método de conversión es muy sencillo, ya que la tensión de entrada se aplica a la entrada de comparación de todos los comparadores. Como resultado, estos activan su salida cuando la tensión de comparación supera a la de referencia. Pero, del sistema, solo sale un código binario, pues el codificador de prioridad únicamente genera una salida equivalente a la entrada de mayor peso que esté activa, y el resto es ignorado. Si bien este método es rápido y eficiente, tiene el inconveniente del tamaño y del costo del conversor, porque se necesitan 255 comparadores para construir un conversor de 8 bits.



> El conversor dinámico o de rampa simple es más económico que el estático. La salida de la conversión se lee desde la salida del registro tipo LATCH.

Conversor de rampa simple o dinámico

Este tipo de conversor es más económico y eficiente que el anterior. Se lo suele denominar conversor A/D de rampa simple o dinámico, y se construye a partir de un contador digital y un comparador.

Cuando el conversor arranca, tanto el contador como el comparador inician en cero. Como la salida del comparador tiene un nivel lógico cero, una compuerta **AND** inhibe el paso de la señal de **clock** hacia el contador. Al aplicarse una señal en la entrada del comparador, dado que la entrada de referencia de este vale cero, la salida del comparador pasa a uno. Esto habilita la compuerta **AND**, que deja pasar los pulsos de clock al contador.

De esta forma, el comparador comienza a incrementar su estado de cuenta. La salida del contador se envía hacia un registro tipo **LATCH** y, al mismo tiempo, a un conversor D/A, mediante el cual se genera la tensión de referencia. De esta manera, la referencia comienza a incrementarse, y cuando llega al valor de la tensión de entrada, la salida del comparador pasa a cero. Esto desactiva la AND e inhibe el paso de la señal de clock. El contador se detiene y se genera la señal para que el registro tipo LATCH capture el estado de cuenta y lo presente en su salida como el valor binario de la conversión de la señal de entrada.

El tiempo de conversión depende del nivel de la señal de entrada: cuanto más grande sea, mayor será el tiempo de conversión. La principal desventaja del conversor de rampa simple es su inestabilidad en la generación de la rampa. Por otra parte, como no existe una sincronización entre la señal de clock y la generación de la rampa, cualquier corrimiento afectará el resultado de la conversión. Este problema se compensa en el sistema de conversión de doble rampa, que es más lento pero mucho más estable.

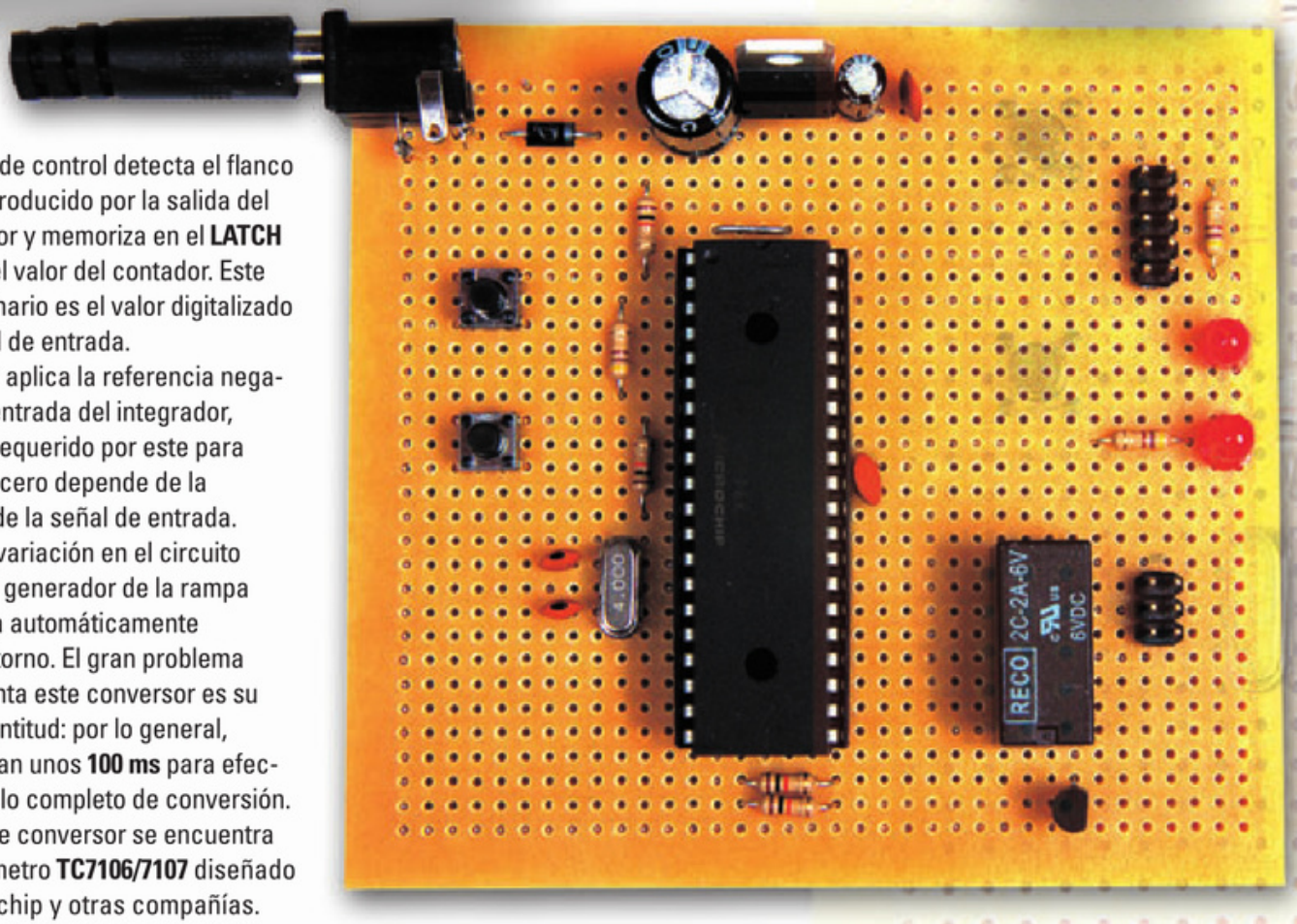
> APLICACIÓN DE LOS CONVERSORES

Este tipo de conversores, con ligeras modificaciones, ha sido adoptado por su simplicidad para la conversión de audio en formato digital, en cuyo caso se denominan conversores tipo sigma-delta. Los procesadores de efecto para guitarras eléctricas, al igual que los pedales de efecto digitales, lo emplean para generar la conversión, con la variante de que la salida del comparador se aplica directamente a la entrada de un procesador de señales digital.

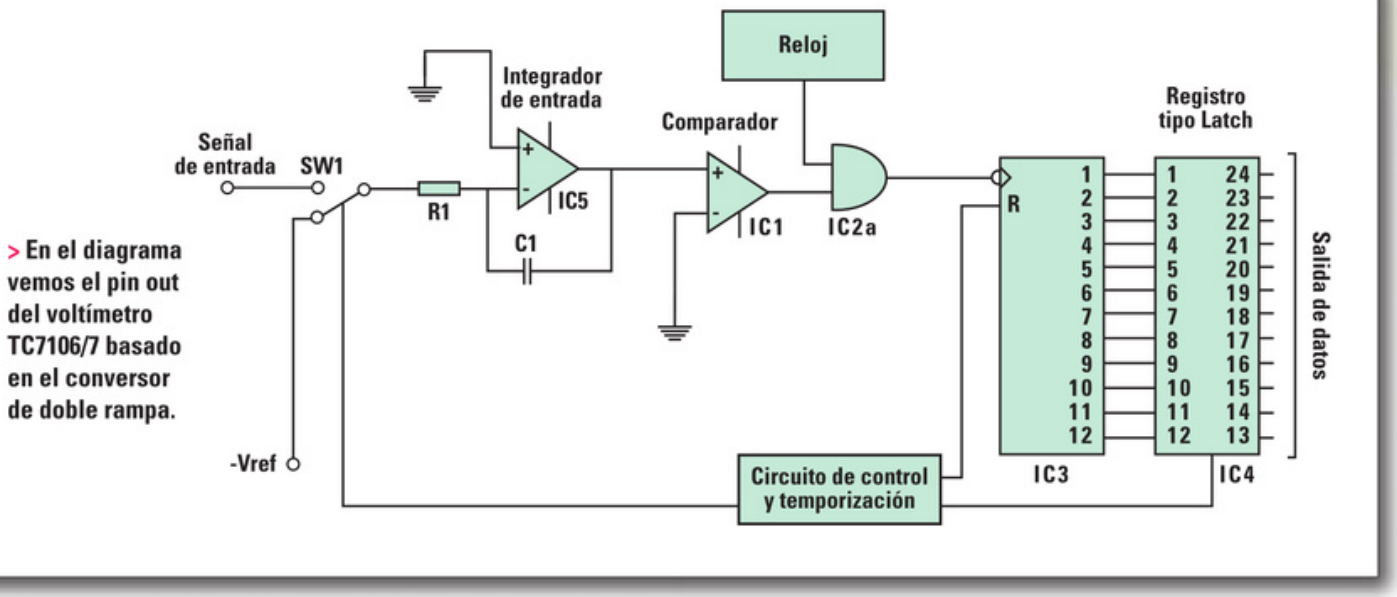
Conversor de doble rampa

Este tipo de conversor subsana las deficiencias del sistema de **rampa simple**, pero es más lento que su antecesor. Elimina el efecto del corrimiento del voltaje de rampa a lo largo del tiempo, y también utiliza un integrador de entrada.

Este circuito está formado por un amplificador operacional y un capacitor en el lazo de realimentación. Cuando aplicamos una tensión positiva a la entrada del integrador, la salida crece, pero en sentido negativo. Dicha tensión provoca que la salida del comparador pase a uno, lo que activa la **AND**, que permite el paso de los pulsos de clock que hacen avanzar al contador. La rampa negativa generada por el integrador tiene un tiempo fijo, determinado por el **RC** del integrador. Después de este tiempo, el circuito de control pone a cero el contador y, también, pone la entrada del integrador a una tensión de referencia negativa. En estas condiciones, el integrador genera una rampa positiva. El contador iniciará su cuenta hasta que la salida del integrador llegue a cero, lo que provocará que el comparador entregue cero en su salida.



El circuito de control detecta el flanco negativo producido por la salida del comparador y memoriza en el **LATCH** de salida el valor del contador. Este número binario es el valor digitalizado de la señal de entrada.
Cuando se aplica la referencia negativa en la entrada del integrador, el tiempo requerido por este para retornar a cero depende de la magnitud de la señal de entrada. Cualquier variación en el circuito integrador generador de la rampa se cancela automáticamente en este retorno. El gran problema que presenta este conversor es su extrema lentitud: por lo general, se necesitan unos **100 ms** para efectuar un ciclo completo de conversión. Este tipo de conversor se encuentra en el voltímetro **TC7106/7107** diseñado por Microchip y otras compañías.

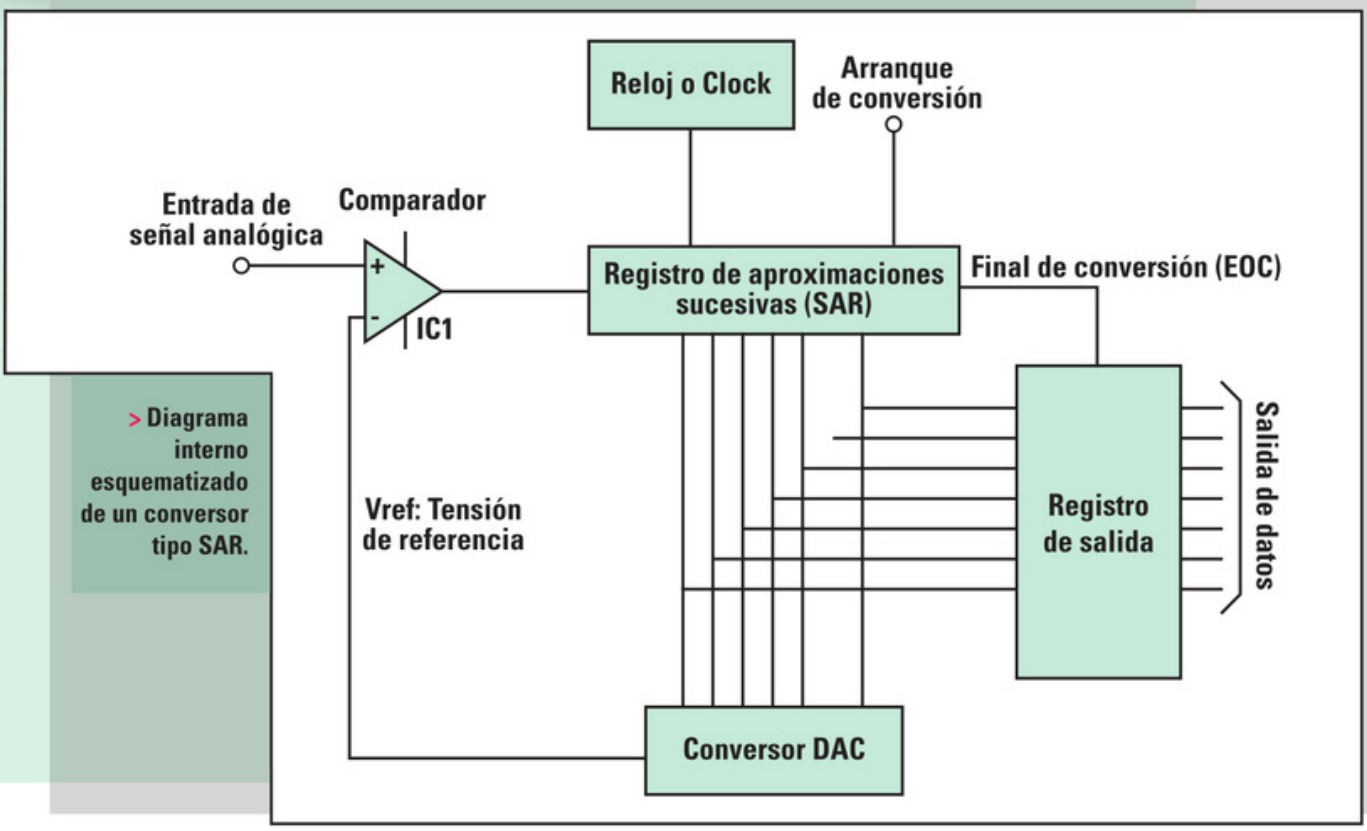


> En el diagrama vemos el pin out del voltímetro TC7106/7 basado en el conversor de doble rampa.

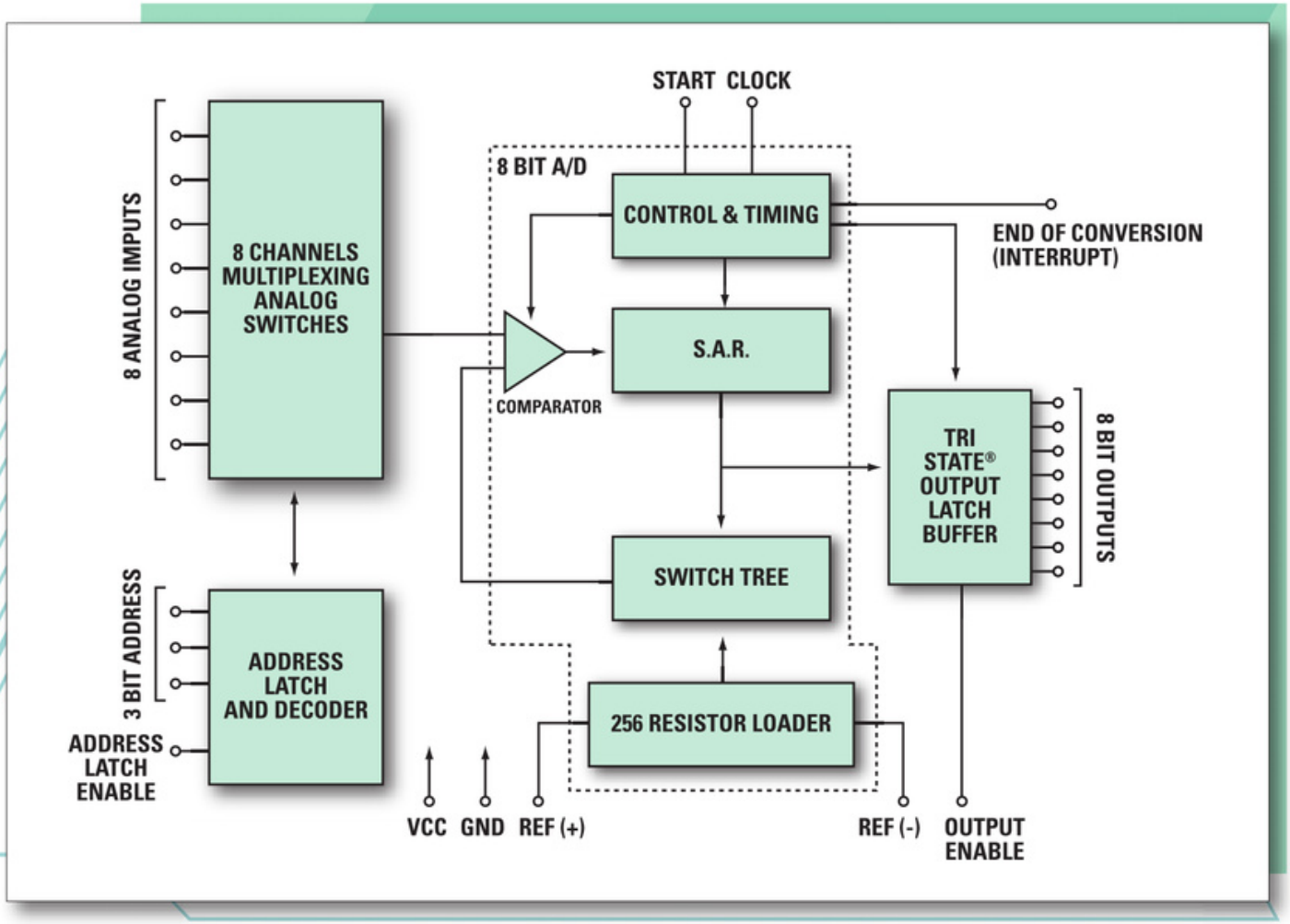


> VOLTÍMETRO DE 3 ½

A mediados de los años 80, la empresa Intersil, que desarrollaba soluciones embebidas para voltímetros, presentó su voltímetro de 3½ dígitos basado en un conversor de doble rampa. Este circuito integrado, denominado ICL7106/7, fue producido, luego, por otros fabricantes como Microchip (TC7106/7). Su uso se extendió ampliamente en la fabricación de testers digitales.



> Diagrama interno esquematizado de un conversor tipo SAR.



En la figura vemos el pin out del ADC0808, de la firma National Semiconductors. En la actualidad, otros fabricantes también desarrollan este tipo de conversor.

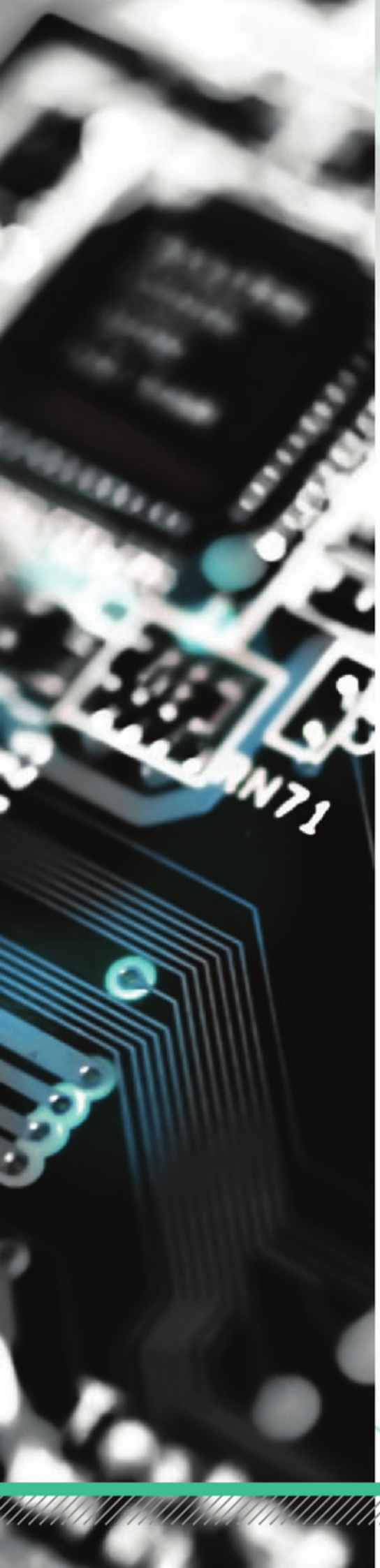
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Mencione las características de un microprocesador.
2. Enumere los componentes que integran un microprocesador.
3. ¿Qué es y qué función cumple la ALU?
4. ¿Para qué sirve el bus de direcciones?
5. Entregue algunos ejemplos de periféricos de entrada y salida.
6. ¿Qué ventajas proporciona el uso del lenguaje ensamblador?
7. ¿Qué es el ciclo de máquina?
8. Detalle las características de la arquitectura Von Neumann y Harvard.
9. ¿Cuál es la principal función de las interrupciones?
10. ¿Qué son los conversores analógico-digitales?

EJERCICIOS PRÁCTICOS

1. Realice un diagrama que muestre los componentes de un microprocesador.
2. Experimente con algún lenguaje de alto nivel como COBOL o BASIC.
3. Genere una línea de tiempo en la cual se identifique la evolución de los microprocesadores.
4. Analice el esquema de un conversor tipo SAR.



CAPÍTULO 3

MICRO-CONTROLADORES PIC16F

Características del PIC16F
Contador de un dígito con display de 7 segmentos
Simulador de hogar de leña
PIC16F por dentro
MPLAB
Prácticas con MPLABSIM
El grabador de PICS
Actividades

03

CARACTERÍSTICAS DEL PIC16F

Son fabricados por la firma Microchip y se ofrecen con una amplia gama de funcionalidades; se destacan por su sencillez y bajo consumo.

Los microcontroladores PIC16F pertenecen a la familia de rango medio de los micros de 8 bits de Microchip. Poseen un repertorio de 35 instrucciones, procesador de filosofía RISC (set de instrucciones reducidas), buses de arquitectura Harvard construidos en pastillas de tecnología CMOS, frecuencia de operación de hasta 20 MHz, hasta 8 KB de memoria Flash de programa, conversores A/D integrados y manejo de interrupciones.
Son ideales para aprender las bases de la programación de estos dispositivos.

Microcontrolador PIC16F887

Pertenece a la familia PIC16F88X, la nueva familia de microcontroladores de rango medio de Microchip. Posee un encapsulado de 40 pines del tipo DIP (*Dual In-Line Package*); podemos ver la disposición de sus pines en la **Figura 1**.

El PIC 16F887 tiene las siguientes características en términos de CPU:

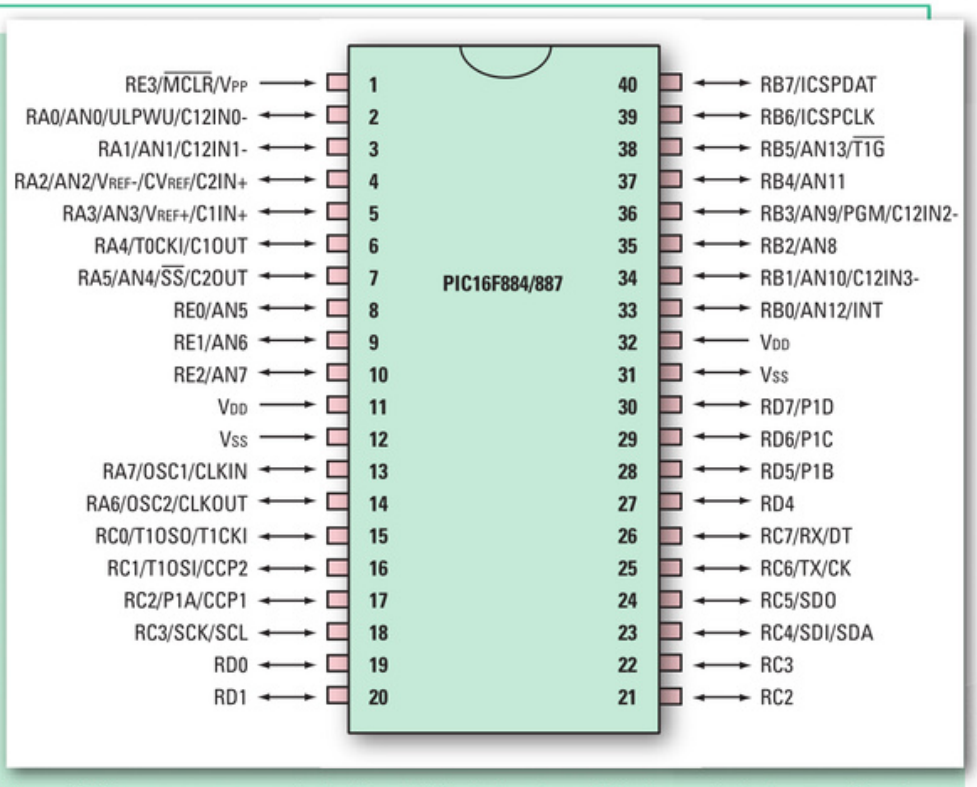
- Tecnología RISC con un repertorio de 35 instrucciones.
- Todas las instrucciones se ejecutan en un ciclo, salvo las de salto, que lo hacen en dos.
- Frecuencia de operación hasta 20 MHz.

En términos de memoria, presenta las siguientes características:

- 8 K x 14 bits de memoria Flash de programa.
- 368 bytes de memoria RAM de datos.
- 256 bytes de memoria EEPROM para datos.
- Lectura/escritura de la CPU a la memoria Flash de programa.
- Stack de hardware de 8 niveles.

Otras especificaciones relevantes del PIC 16F887 son:

- Modo SLEEP de bajo consumo de energía.
- Programación y depuración serie "In-Circuit" (ICSP).
- Rango de voltaje de operación de 2 a 5,5 V.
- Rangos de temperatura comercial, industrial y extendido.
- Conversor A/D de 10 bits de resolución.
- 2 timers de 8 bits y 1 de 16 bits.
- 35 pines de entrada/salida digital.
- Oscilador interno de gran precisión.



> Figura 1. Diagrama que muestra la disposición de los pines del microcontrolador y sus funciones.



Circuito de alimentación

El microcontrolador **PIC16F887** se alimenta con **5 V**, que deben provenir de una fuente que mantenga una alimentación estable y confiable. Los pines de alimentación para **VDD** (**potencial positivo de la fuente**) se encuentran en las posiciones **11** y **32**, y para **VSS** (masa), en **12** y **31**. En la **Figura 2** se muestra el esquema de conexión de la alimentación del PIC.
Es importante colocar un capacitor de desacople de **0,1 µF** entre los pines de **VDD** y **VSS** para evitar que los transitorios generados por un componente ingresen en la fuente y afecten a otros dispositivos.
El consumo a **5 V** operando a **4 MHz** es inferior a **1,5 mA**; en tanto que a **32 KHz**, es menor a **40 µA**.

El oscilador

El **PIC16F887** permite hasta ocho modos diferentes para el oscilador. El usuario puede seleccionar alguno de ellos modificando un registro especial de configuración en la dirección **2007H** de la memoria de programa.
Los modos de operación para el oscilador son:

- **RC**: resistencia/capacitor externos.
- **LP**: cristal para baja frecuencia.
- **XP**: cristal para frecuencias intermedias.
- **HS**: cristal para altas frecuencias.

Los modos **LP**, **XT** y **HS** usan un cristal o resonador externo; la diferencia es la ganancia de los drivers internos, que se ve reflejada en el rango de frecuencia admitido y en la potencia consumida. En la **Tabla 1** se muestran los rangos de frecuencia, así como los capacitores recomendados para un oscilador basado en cristal. Lo más común es utilizar un cristal externo de **4 MHz** con capacitores de **22 pF**.

RANGOS DE FRECUENCIAS

Modo	Frecuencia típica	Capacitores recomendados	
		C1	C2
LP	32 KHz	68 a 100 pF	68 a 100 pF
	200 KHz	15 a 30 pF	15 a 30 pF
XP	100 KHz	68 a 150 pF	68 a 150 pF
	2 MHz	15 a 30 pF	15 a 30 pF
	4 MHz	15 a 30 pF	15 a 30 pF
HS	8 MHz	15 a 30 pF	15 a 30 pF
	10 MHz	15 a 30 pF	15 a 30 pF
	20 MHz	15 a 30 pF	15 a 30 pF

> **Tabla 1. Aquí vemos los diferentes rangos de frecuencias que soporta el PIC para un cristal externo. Es importante respetar el rango de valores que recomienda el fabricante para los capacitores que acompañan al cristal.**



> **Figura 2. Circuito de alimentación para el microcontrolador PIC16F887. Los 5 V pueden provenir de una batería o de una fuente regulada, como la que se armó en el fascículo 6.**

Oscilador interno

El PIC16F887 posee **dos osciladores internos** independientes que pueden controlarse por medio de registros internos. Uno de ellos funciona a bajas frecuencias, en tanto que el otro lo hace a altas.
Este último está calibrado de fábrica en **8 MHz**, y su frecuencia puede ser alterada por software. Podemos obtener más información sobre este modo de operación en las hojas de datos del microcontrolador.

Oscilador externo

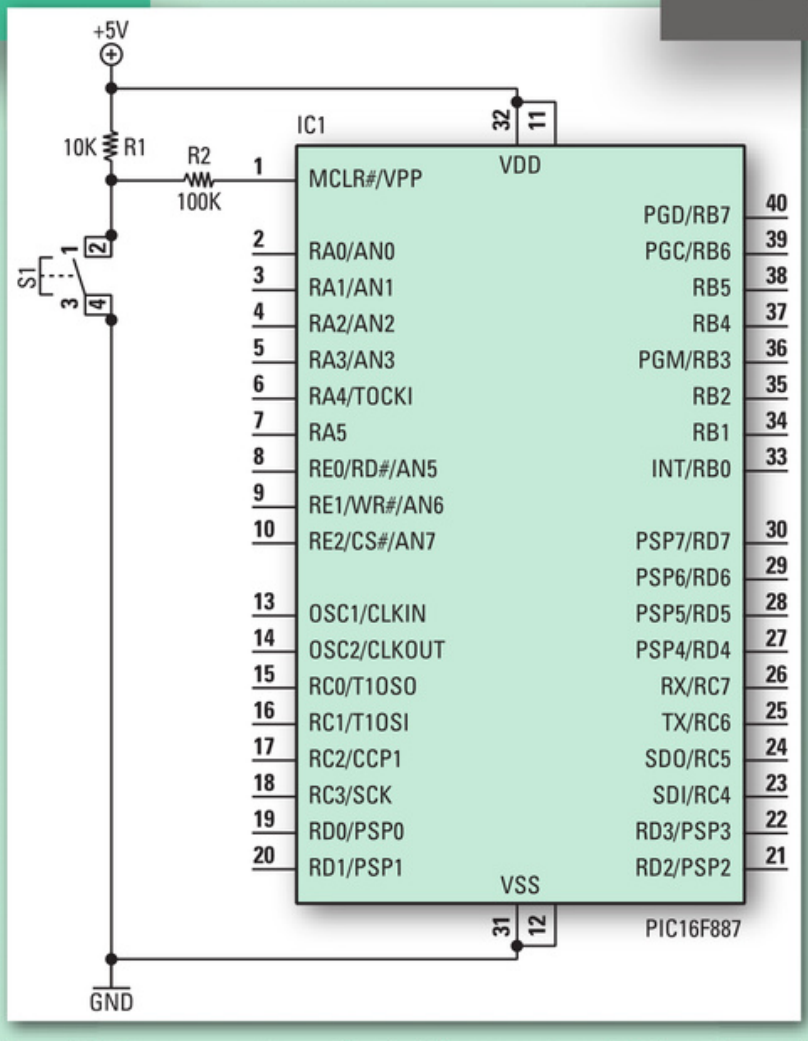
También es posible conectar una señal de reloj generada mediante un oscilador externo a la patita OSC1 del PIC. Para esto, el PIC debe estar en uno de los tres modos que admiten cristal (LP, XT o HS).

Circuito de reset

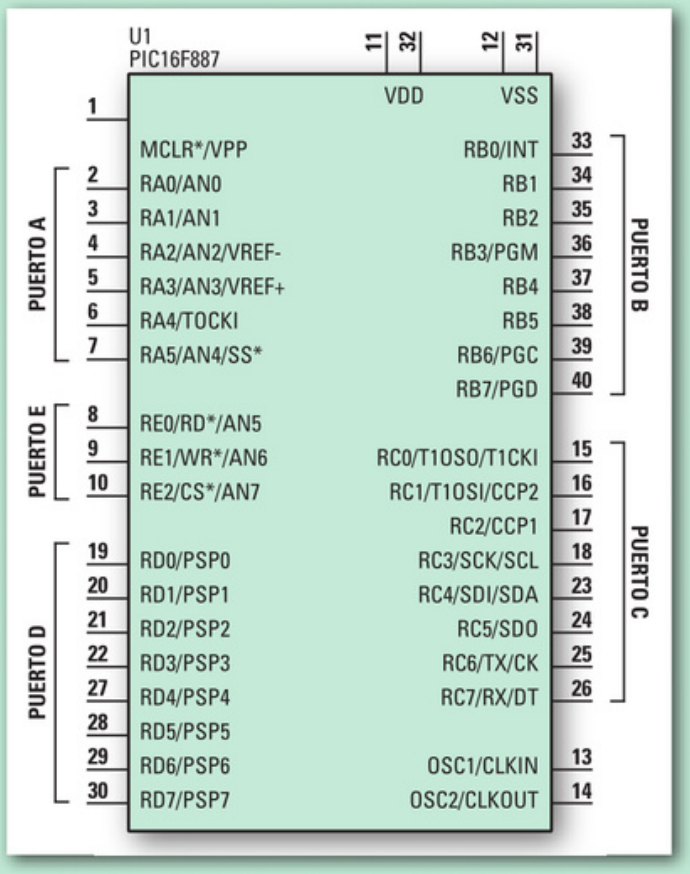
Cuando se produce un reset, el procesador se reinicializa, el programa en ejecución se abandona, y el contador de programa se carga con la dirección 0. Es allí donde comienza el programa de la aplicación. **Las dos formas más comunes para provocar un reset son:**

- Quitar y volver a establecer la alimentación del microcontrolador. Cuando se conecta la alimentación, se produce automáticamente un reset, conocido como **Power-On Reset** (POR).
- Aplicar un nivel lógico bajo en el terminal **MCLR** (*Master Clear Reset*), que permite provocar un reset externamente en el momento en que queramos, sin necesidad de desconectar el microcontrolador de su alimentación. Para introducir un nivel bajo en este pin, debemos colocar un pulsador que lo lleve a ese estado.

En la **Figura 3** observamos un esquema típico para el circuito de reset.



> Figura 3. La imagen muestra un circuito típico para el reset. Esta configuración aplica un nivel bajo en el pin MCLR, que genera un reset del sistema.



> Diagrama esquemático del PIC, donde se ve en detalle la disposición de los cinco puertos del microcontrolador.

Puertos de entrada y salida

El microcontrolador PIC16F887 posee cinco puertos de entrada y salida: **A**, **B**, **C**, **D** y **E**. Debido a que el bus de datos es de **8 bits**, es lógico que el tamaño de los puertos sea el mismo. Por eso, los puertos **B**, **C** y **D** tienen ocho líneas de E/S digitales, mientras que el **A** tiene seis, y el **E** solo tres. Cada pin de cada puerto puede configurarse de manera individual como entrada o salida digital para recibir o enviar datos, según las necesidades.

Pero no todos los puertos son solamente **E/S digitales**: para ahorrar pines en el encapsulado, estos se comparten entre los distintos periféricos. Por ejemplo, el puerto **A** puede usarse, además, como entrada analógica para el conversor A/D incluido en el integrado, mientras que el **B** tiene los pines para programación en **RB6** y **RB7**, y una entrada para interrupción externa en el pin **RB0**, por ejemplo.

Organización de la memoria

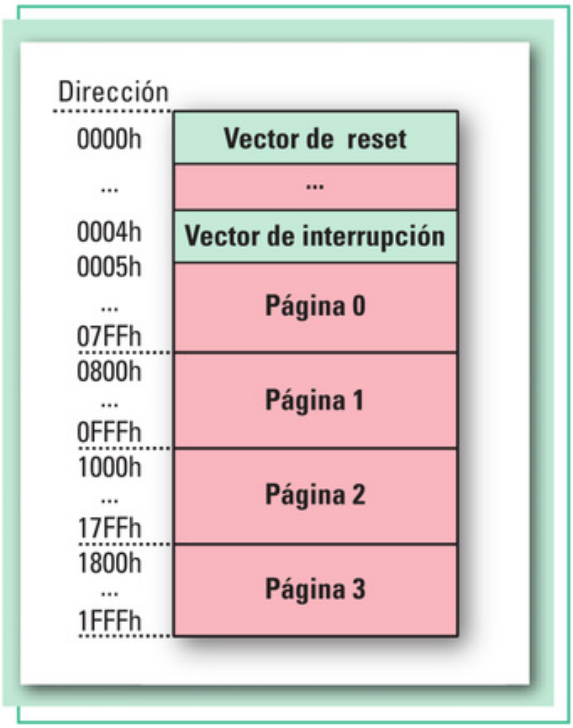
El PIC16F887 tiene dos tipos de memoria: de datos y de programa. Por su parte, cada bloque de memoria posee su propio bus: el de datos y el de programa. De este modo, se obtiene una arquitectura Harvard, que permite tener acceso a cada bloque de memoria durante el mismo ciclo de instrucción. Estas dos memorias tienen funcionalidades diferentes y por eso su tecnología de fabricación es distinta.

La memoria de programa

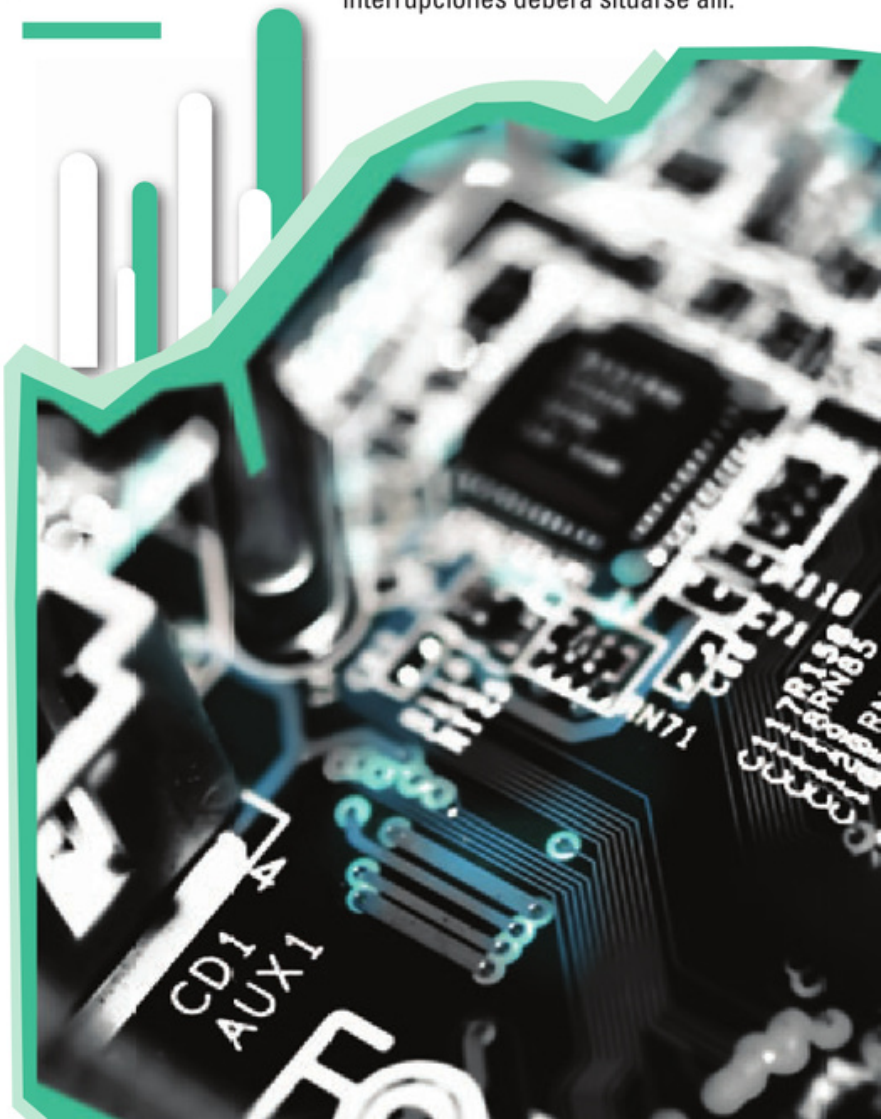
Almacena las instrucciones de la aplicación y está realizada con memoria del tipo Flash, lo que permite escribirla y borrarla las veces que sean necesarias. Las instrucciones de los **PIC16F88X** (familia a la que pertenece el PIC16F887) tienen una longitud de **14 bits**, por lo que las posiciones de memoria deben tener igual longitud. El **PIC16F887** en particular posee **8192 posiciones** de memoria para programa (**8 KB**), de manera que para acceder a una de las posiciones necesitaríamos un bus de 13 líneas (**2^{13}= 8 K**). Este es el tamaño de bus que Microchip utiliza para los PIC de rango medio. La memoria de programa está dividida en páginas de igual tamaño; en los PIC16F88X, son de 2 K. El PIC con el que trabajaremos en este caso tendrá cuatro páginas. La página 0 es común en todos los PIC de esta familia y tiene solo dos posiciones que se encuentran reservadas:

- **Vector de reset** (ubicado en la posición 0000H): cuando se conecta la alimentación o se resetea el procesador, la primera instrucción que se ejecuta es la que se encuentra en esta posición. Todas nuestras aplicaciones empezarán en el vector de reset.
- **Vector de interrupción** (ubicado en la posición 0004H): cuando se produce una interrupción, el **contador de programa** siempre apunta a esta dirección. Por lo tanto, el comienzo del programa que atiende a las interrupciones deberá situarse allí.

El PIC16F887 posee dos buses de datos independientes: uno para la memoria de programa y otro para la de datos.



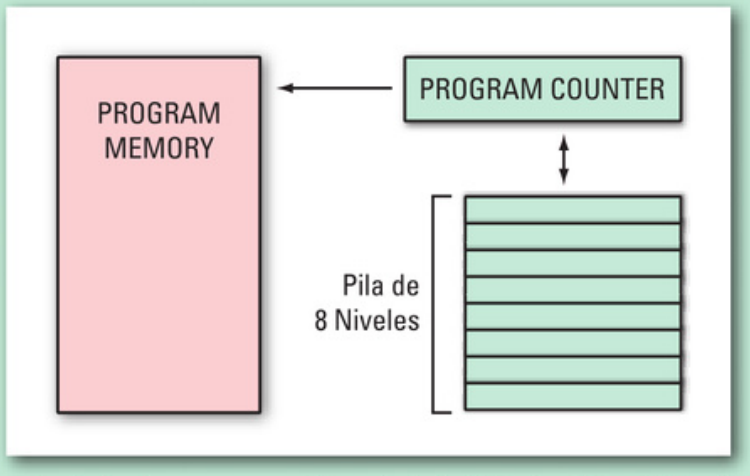
La memoria de programa posee cuatro páginas de 2 K, lo que completa un total de 8 K para posiciones de memoria. En la imagen podemos ver el vector de reset y el de interrupciones.



Contador de programa

Este registro contador incrementa su valor en una unidad en cada ciclo de instrucción, excepto cuando hay una instrucción de salto o una interrupción, en cuyo caso puede cargarse con cualquier otro valor. Ante un salto con retorno (interrupción o llamada a una subrutina, **CALL**), el valor que tiene el contador de programa debe ser almacenado para retomarlo posteriormente. Para esto se utiliza una pila **LIFO** (*Last Input First Output*, o último en entrar, primero en salir) de **8 niveles**, de forma que guarde la última dirección antes del salto en el nivel 1 de la pila, y pueda retomarla en el momento en que la necesite.

Las interrupciones y la instrucción CALL guardan el valor del contador de programa en la pila.



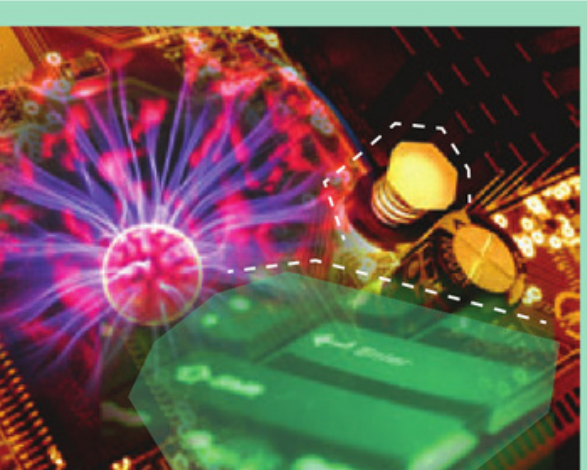
> **Contador de programa y pila o stack de 8 niveles. El contador siempre contiene la dirección de la próxima instrucción por ejecutar.**

La pila o **stack** se carga y descarga por el nivel 1, por lo que cuando se carga una dirección en este nivel, el contenido de cada nivel pasa al siguiente. Cuando se descarga el nivel 1 en el contador de programa, el contenido de cada nivel pasa al precedente. Esto permite una operación **anidada** de llamadas a subrutinas (una dentro de otra).

Memoria de datos

En el área de datos se almacenan los de propósito general y los registros específicos cuyos bits controlan el funcionamiento del microcontrolador. Como los datos que maneja esta memoria son de carácter temporal, es de **tipo RAM**. El PIC también tiene la capacidad de almacenar los datos que queremos conservar al desconectar la alimentación, los cuales son guardados en una zona especial de memoria **EEPROM**.

La región de memoria de datos se compone de cuatro bancos de 128 posiciones de datos de 8 bits cada uno. Cada banco posee **registros de propósitos generales** (GPR), que almacenan datos utilizados por el programa, y **registros de funciones especiales** (SFR), que contienen bits de control para el procesador y los periféricos.



> REGISTRO W

Es operando obligado en operaciones aritméticas y lógicas con más de un operando. Muchas de las instrucciones permiten elegir el destino del resultado. Este puede ser el registro W o la posición de memoria (GPR) sobre la que se opera. Para asignar un valor constante a una variable (GPR), primero hay que cargar el registro W y copiarlo al GPR.

Registros de propósito general

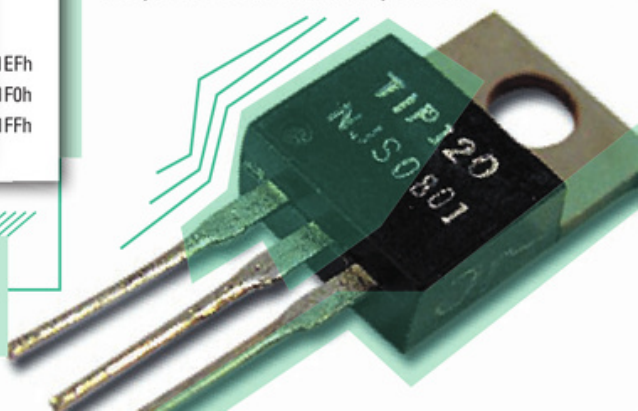
Cuando ejecutamos una instrucción, el procesador puede necesitar datos que se encuentran almacenados en la memoria correspondiente. Si estos son utilizados por el usuario, deben ser almacenados en la zona de **registros de propósitos generales** (GPR), que guarda las variables requeridas por los programas.
En la **Figura 4,** podemos visualizar los bloques para los registros GPR: en el banco **0** tenemos **96** bytes para almacenar este tipo de datos, comenzando en la posición **20h** y llegando hasta **7Fh**; en el banco **1** tenemos **80** bytes, desde **A0h** hasta **EFh**; y en los bancos **2** y **3** volvemos a tener 96 bytes para datos.
En las últimas posiciones de los bancos **1**, **2** y **3,** hay un bloque de **16** posiciones que replican los últimos **16** registros del banco **0**.
Si intentamos acceder a algún registro en las últimas posiciones de los bancos 1, 2 y 3, en realidad estaremos accediendo a los registros del banco 0.
Estas posiciones son ideales para los datos que se utilizan de manera frecuente, porque evitamos realizar los cambios de bancos para direccionar esos registros.

	File Address		File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PC LATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTOON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Reglaters 16 Bytes	116h	General Purpose Reglaters 16 Bytes	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
RCREG	19h	SPBRG	99h		119h		199h
TXREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
	20h		A0h		120h		1A0h
General Purpose Reglaters 96 Bytes	3Fh 40h 6Fh	General Purpose Reglaters 80 Bytes	EFh	General Purpose Reglaters 80 Bytes	16Fh	General Purpose Reglaters 80 Bytes	1EFh
	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh

La memoria de datos está dividida en cuatro bancos de 128 posiciones cada uno. Al principio de cada banco, vemos los registros de funciones especiales y, luego, los de propósitos generales.

Registros de funciones especiales

Las primeras posiciones de los bancos del área de memoria de datos son ocupadas por los **registros de funciones especiales** (SFR), que sirven a fines específicos en el funcionamiento y la configuración del microcontrolador.
Cada SFR está asociado a una función general, y dentro de cada uno, cada bit tiene una función determinada. Es muy importante aprender la función y el manejo de cada registro para hacer un uso correcto del microcontrolador.
Estos registros están dedicados a manejar los periféricos o recursos que posee el PIC, almacenar el estado del procesador, realizar el direccionamiento de la memoria, controlar las interrupciones, y manejar los puertos de entrada y salida.



STATUS

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
Bit 7	6	5	4	3	2	1	Bit 0

> **Registro de estado, llamado STATUS, y sus bits.**

El contador de programa (PC) del PIC16F887 posee un registro mapeado en la memoria de datos.

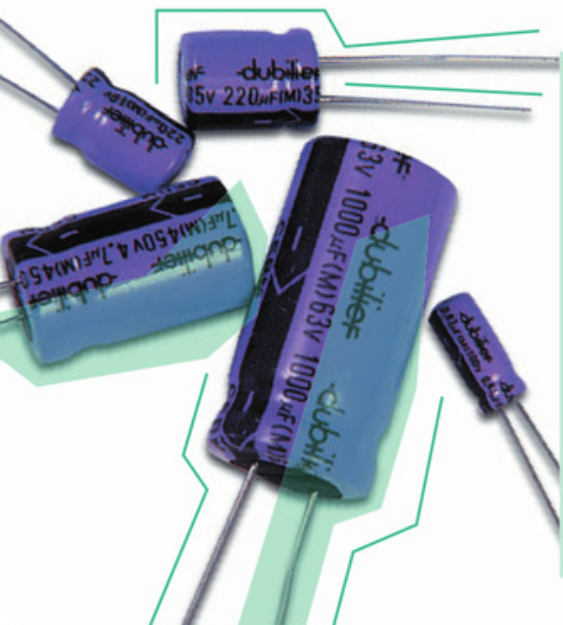
El registro de estado

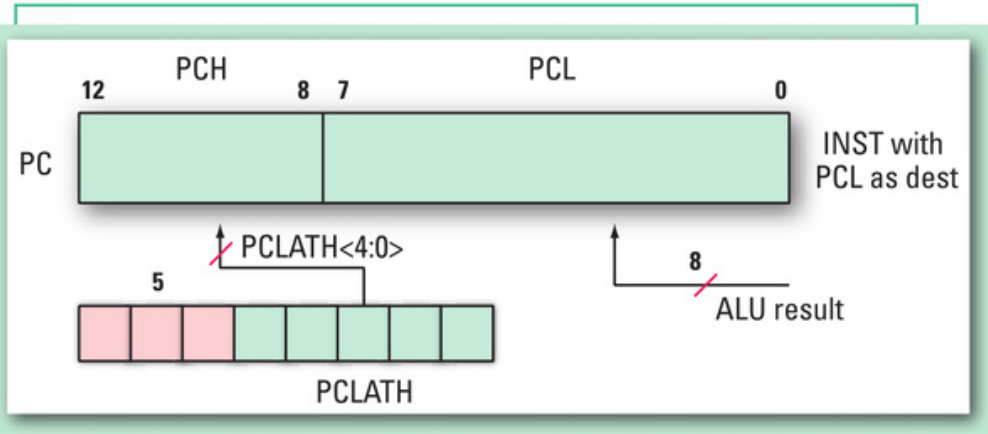
Este registro ocupa la cuarta posición en los cuatro bancos de memoria y se denomina **STATUS**. Es un registro de control en cuyo interior se almacena el estado de las operaciones aritméticas de la **ALU**, el estado del **reset** y los bits para la selección del banco de memoria de datos. Veamos en detalle las funciones de cada uno de sus bits.

- **Bit C** (CARRY): es el bit de acarreo, que sirve para indicar si se produjo un acarreo en el bit más significativo durante la última operación.
- **Bit DC** (Digit Carry): es el bit de acarreo de dígito. Es similar al bit C, salvo que indica el acarreo entre los bits **3** y **4**. Se utiliza cuando realizamos operaciones con datos en BCD.
- **Bit Z** (Zero): es el bit de 0, y se activa cuando el resultado de una operación aritmética o lógica es 0.
- **Bit /PD** (Power Down): bit de apagado. Se pone a 0 cuando se ejecuta una instrucción sleep (modo de bajo consumo).
- **Bit /TO** (Time Out): bit de fin de tiempo. Se pone a 0 cuando ocurre un desborde del watchdog, o perro guardián (temporizador que resetea el micro al terminar su cuenta).
- **Bits RP1 RP0**: son los bits de selección del banco de registros.
- **Bit IRP**: bit para la selección de bancos utilizado en el direccionamiento indirecto.

El registro contador de programa

El **contador de programa** (PC) del PIC16F887 posee un registro mapeado en la memoria de datos. Como contiene **13 bits** y las posiciones de memoria son de 8 bits, el contador está representado por dos registros llamados **PCL** y **PCLATH**. El primero tiene los 8 bits menos significativos del **PC** y se ubica en la posición **02h**, pero los 5 **bits** de la parte alta utilizan un registro que sirve para poder escribirlos; ese registro es el **PCLATH**. El **PCL** siempre es actualizado con los 8 bits menos significativos del contador de programa, mientras que el PCLATH nunca se actualiza. ■



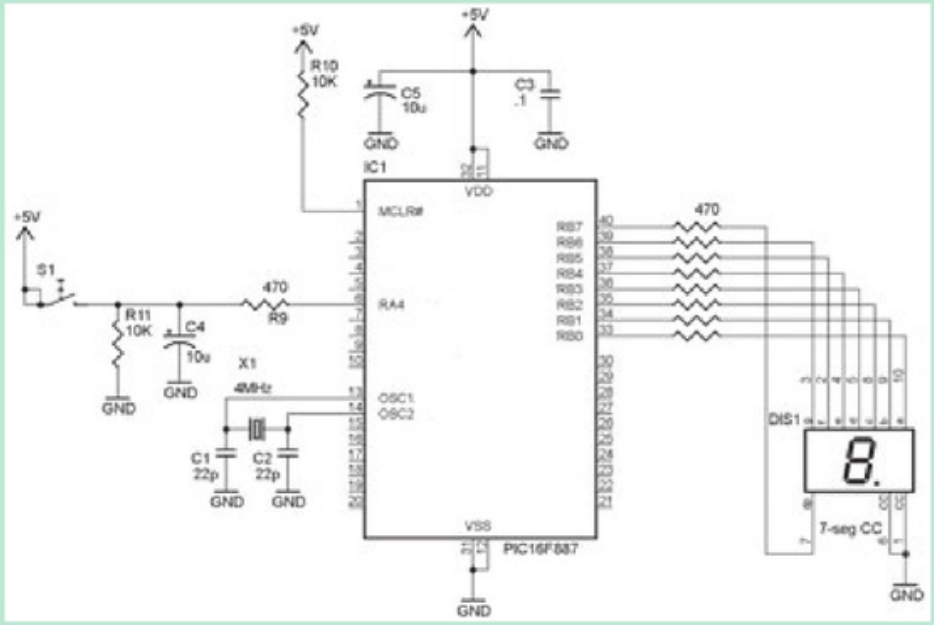


> **El registro del contador de programa ocupa 2 bytes, el PCL maneja los 8 bits menos significativos, y el PCLATH se utiliza como latch de escritura para los 5 bits más significativos del PC.**

CONTADOR DE UN DÍGITO CON DISPLAY DE 7 SEGMENTOS

En estas páginas aprenderemos a programar sobre estos microcontroladores; resolveremos paso a paso la tarea propuesta, dividiéndola en pequeños problemas simples.

> **En la figura podemos ver el esquemático de nuestro contador de 7 segmentos completo.**



Un programa es una secuencia ordenada de instrucciones que indican al micro las tareas que debe realizar. Este nunca se equivoca: si el programa no hace lo que queremos, es porque no se lo estamos indicando correctamente al micro. El desarrollo de un programa sigue una secuencia de pasos:

- **Plantear el problema** por resolver e identificar las variables involucradas.
- Realizar de forma gráfica una secuencia de pasos lógicos que nos permitan llegar a la solución (**diagrama de flujo**).
- Traducir el diagrama de flujo en un **lenguaje de programación**.

Una vez que lo tenemos escrito, es necesario ejecutar un programa que traduzca el lenguaje de programación al **lenguaje de máquina** (binario puro): **ensamblador** para Assembler o **compilador** para lenguajes de alto nivel. Finalmente, el programa se carga en lenguaje de máquina en la memoria del microcontrolador, usando un **programador**.

Hardware del proyecto

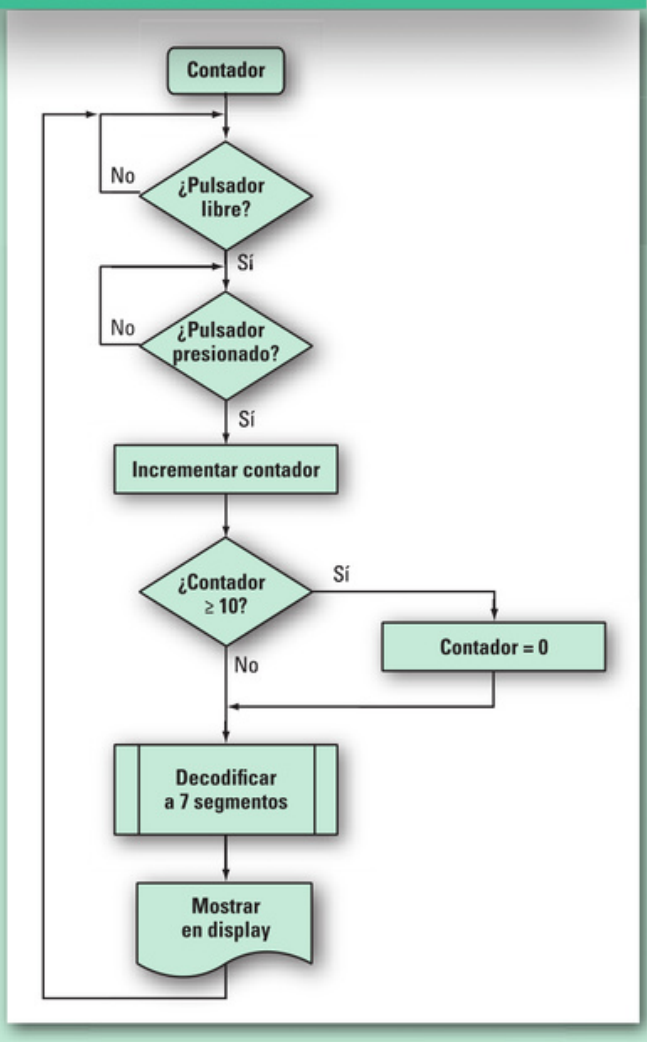
Podemos darnos cuenta de que en este esquema es el microcontrolador el que oficiará de contador y decodificador. Los pulsos ingresan por el pin **RA4** (PORTA, 4) provenientes de un pulsador.

El **PORTB** del **PIC** está conectado a un **display de 7 segmentos** de cátodo común por medio de 8 resistores de **470 Ω** (ohm), para limitar la corriente.

Planteo del problema

Para comprender mejor el planteo de este problema, es necesario observar el diagrama de flujo **Operatoria**, presente en la **página 86**.

- Debemos observar el pulsador. Primero esperaremos a que esté libre, es decir, en cero (**0**).
- Cuando detectamos que está presionado (**1**), incrementamos el valor de una variable, **CONTADOR**, que previamente pusimos en **cero**.
- Luego, verificamos que el **contador** no haya sobrepasado la cuenta de **9**; si lo hizo, lo colocamos en **0**.
- A continuación, transformamos el valor del contador en un valor binario que podamos ver en el display.
- Finalmente, repetimos el ciclo.



> Operatoria. Este diagrama de flujo simboliza la operatoria que resuelve el problema. En él pueden apreciarse los distintos símbolos aplicados a cada parte.

El programa Assembler

Su estructura comprende cuatro campos, en el siguiente orden:

- **Etiquetas**: son puntos de referencia en el programa, que siempre deben empezar con una letra y no contener espacios intermedios.
- **Instrucciones**: son los **nemónicos** del micro utilizado, que obtenemos de la descripción del set de instrucciones en su hoja de datos (en este caso, PIC16F887).
- **Operandos**: son los nombres de los registros, constantes y variables que participan en las instrucciones.
- **Comentarios**: se indican con un punto y coma (;), y permiten incluir un texto que explique lo que sucede en esa parte del programa.

Esto constituye la **documentación**.

Escribimos entonces el programa, traduciendo la operatoria del diagrama de flujo en las instrucciones (nemónicos) del micro.

```
; Inicialización
          CLRF CONTADOR              ; CONTADOR = 0

; Lectura del pulsador en un pin de I/O
; Esperamos a que esté libre
P1        BTFSC PORTA,4      ; lee el pin, no ejecuta la siguiente instrucción
          GOTO P1            ; si está en 0, si no salta a P1
; Esperamos que lo presionen
P0        BTFSS PORTA,4      ; lee el pin, no ejecuta la siguiente instrucción
          GOTO P0            ; si está en 1, si no salta a P0

; El pulsador está presionado, incrementamos el contador
          INCF CONTADOR,F    ; incrementa CONTADOR, y guarda
                                        ; el resultado allí

; Verificamos que no exceda el valor 9
          MOVLW 0A                   ; Carga el registro W con el valor 10 (0A)
          SUBWF CONTADOR,W           ; Se lo resta a CONTADOR, el resultado
                                        ; se guarda en W
          BTFSC STATUS,C        ; El flag C en 1 indica que la resta es >=0, un
          CLRF CONTADOR         ; exceso, ponemos CONTADOR en cero

; Convertimos el valor del contador a 7-segmentos
          MOVF CONTADOR,W  ; coloca el valor de CONTADOR en W
          CALL CD7SEG                ; convierte, obtiene el valor en W

; Escribimos en el display
          MOVWF PORTB                ; Coloca el valor de W en PORTB

; Repetimos el ciclo
          GOTO P1
```



> Tenemos el problema resuelto. De igual modo podemos operar con cualquier otro micro.

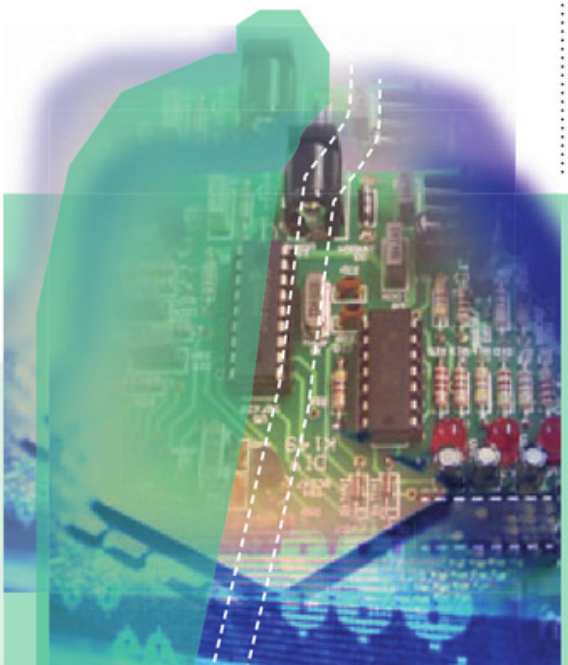
Set de instrucciones

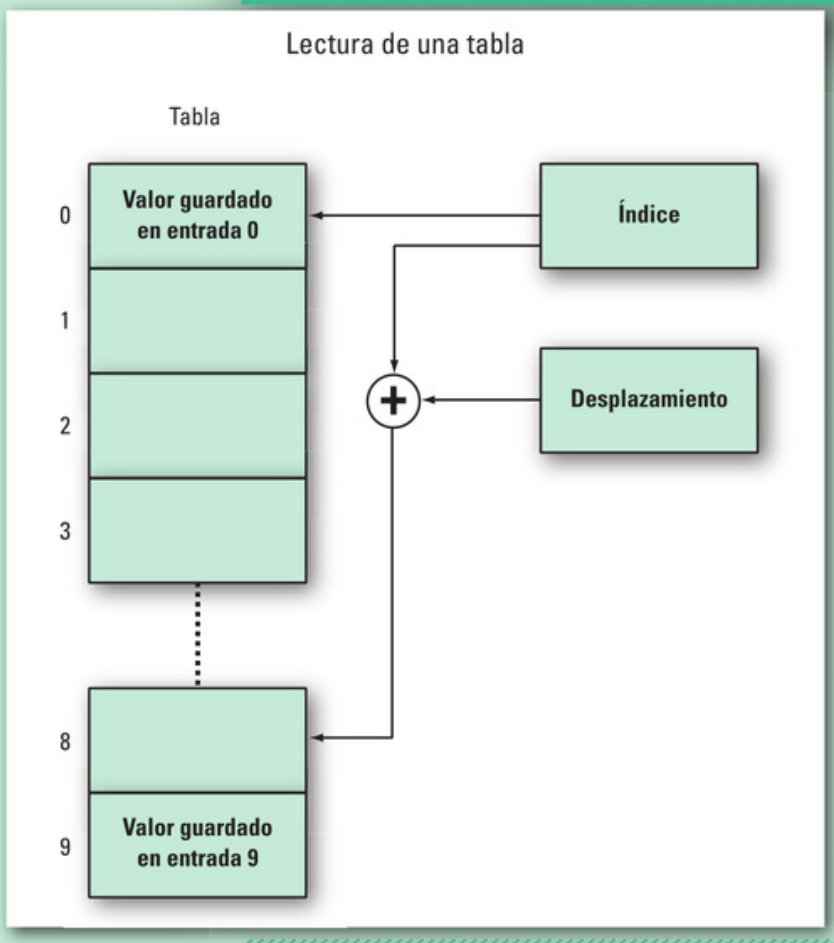
Es el conjunto de nemónicos (instrucciones) del micro, cada uno de los cuales realiza una acción específica.
Las instrucciones para operaciones lógicas y aritméticas indican estados especiales mediante flags en el registro de estado. El set de instrucciones puede consultarse en la hoja de datos o en el manual del usuario del micro.

La instrucción RETLW significa 'volver con un valor en W', y es la que nos permite leer la tabla.

Decodificación

Se realiza mediante una búsqueda en **tabla**. Esto es similar a la operación de una **PROM**. El número que vamos a convertir nos da la posición (desplazamiento, offset) dentro de la tabla donde está el valor convertido; lo sumamos a la posición de la tabla y obtenemos lo deseado.





Este diagrama simboliza la operación de búsqueda en la tabla. El registro W oficia como desplazamiento (offset) respecto del índice (PCL).

```
; Conversión a 7-segmentos
; Leemos de una tabla
CD7SEG   ADDWF PCL,F          ; Suma W al PC, salta tanto como indique W
                              ; que contiene el valor por convertir
                ; dp gfedcba
         RETLW B'00111111'    ; bits en 1 prenden segs del dígito 0
         RETLW B'00000110'    ; bits en 1 prenden segs del dígito 1
         RETLW B'01011011'    ; bits en 1 prenden segs del dígito 2
         RETLW B'01001111'    ; bits en 1 prenden segs del dígito 3
         RETLW B'01100110'    ; bits en 1 prenden segs del dígito 4
         RETLW B'01101101'    ; bits en 1 prenden segs del dígito 5
         RETLW B'01111100'    ; bits en 1 prenden segs del dígito 6
         RETLW B'00000111'    ; bits en 1 prenden segs del dígito 7
         RETLW B'01111111'    ; bits en 1 prenden segs del dígito 8
         RETLW B'01100111'    ; bits en 1 prenden segs del dígito 9
```

> INICIALIZACIÓN DEL HARDWARE

Antes de que se ejecute el programa, debemos inicializar los ports de I/O para el hardware en particular, de modo de poder accederlos, y configurar entradas y salidas. Los pines del micro suelen compartirse con otros periféricos, y es preciso reclamarlos antes de usarlos. Veremos este tema más adelante al simular este proyecto con MPLAB.

SIMULADOR DE HOGAR A LEÑA

En nuestro segundo programa aprenderemos nuevas operaciones e instrucciones. Llega el turno de ver operaciones lógicas, de bits y de desplazamiento.

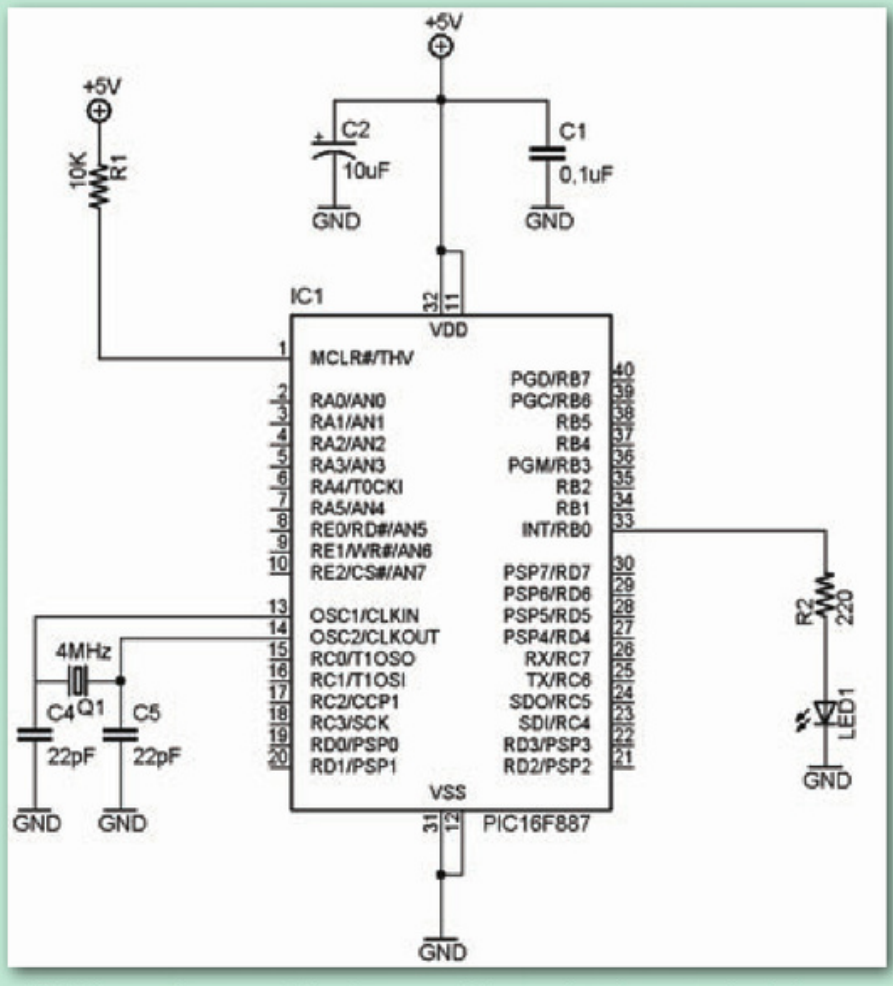
Cuando trabajamos con operaciones lógicas y de desplazamiento, recibimos la ayuda del flag de acarreo (**CARRY**). Este, si se lo aprovecha, simplifica en gran medida la operatoria y es una potencia envidiable del lenguaje Assembler. Estas operaciones son muy usadas en comunicaciones serie, en las cuales la lectura del estado de un pin se va ingresando paso a paso en un registro. De igual modo, el flag de cero (**ZERO**) nos indica si el resultado de la operación es cero.

Hardware del proyecto

En este ejemplo, el microcontrolador realizará la función de registro de desplazamiento y **XOR**.
En el **PORTB** del **PIC** conectamos el **LED** por medio de un resistor de **220 Ω**, para limitar la corriente.

Planteo del problema

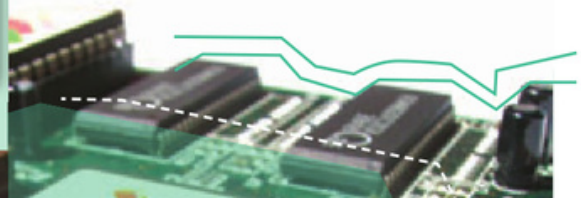
La problemática es simular la operación de un registro de desplazamiento con entrada serie y salida **paralelo**, cuya entrada es la **XOR** de dos de sus salidas como vemos en el diagrama **Planteo del problema** en la página siguiente.

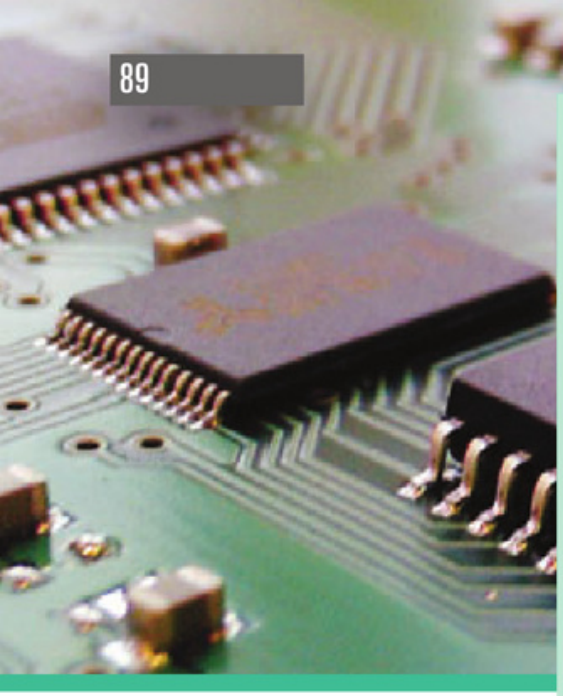


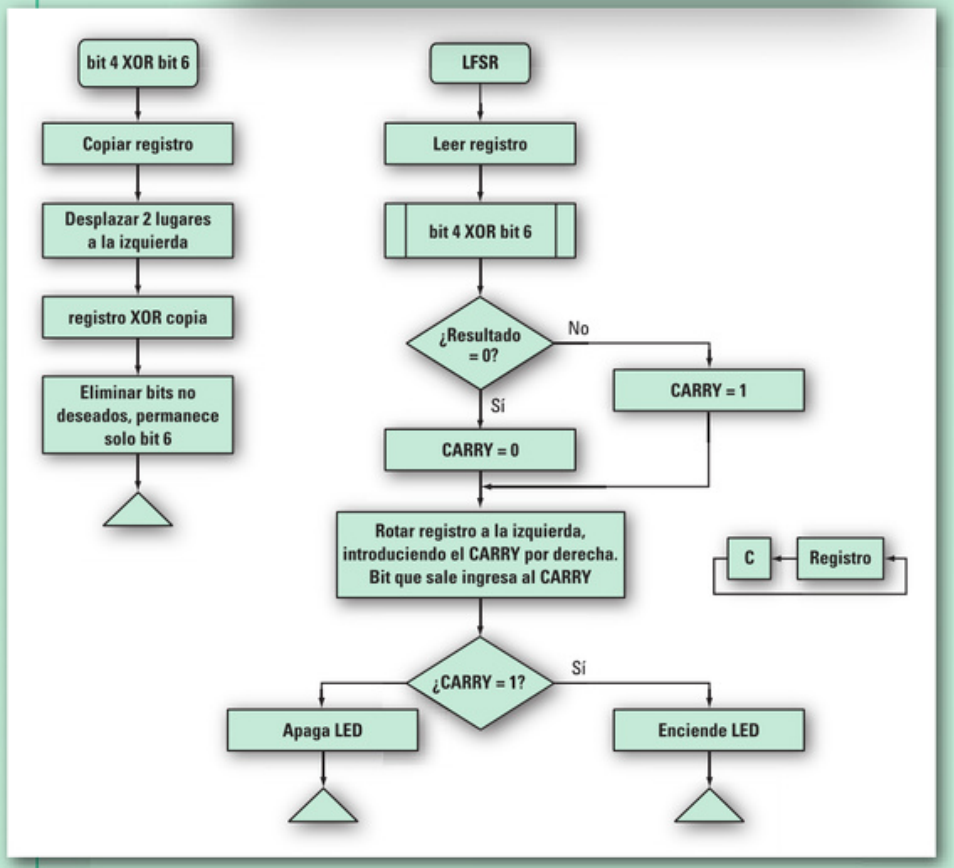
> En la figura podemos ver el circuito esquemático completo de nuestro simulador.

La instrucción RLF realiza una rotación a la izquierda del operando, pasando por el CARRY.

El registro de desplazamiento será simulado por la variable **REGSTR** en nuestro programa. Debemos hacer una **XOR** con los bits **4** y **6**, e introducir el resultado como bit **0** de dicha variable, a la vez que desplazamos todos sus bits a la izquierda. El bit que sale (**antiguo bit 7**) controlará el encendido del **LED**. La mayoría de los micros efectúan







Planteo del problema. Diagrama de flujo correspondiente al planteo del problema. El símbolo de terminación indica el fin de la operación.

operaciones lógicas solo entre bytes, de modo que para hacer una **XOR** entre bits, deberemos realizar una copia del registro y desplazarlo hasta que ambos bits coincidan, eliminando el resto mediante una **AND**.

El programa Assembler

La **XOR** entre bits se hace realizando una copia del registro:

```
; Lectura y copia del registro
MOVF REGSTR,W                   ; Copia el contenido de REGSTR a W
MOVWF AUX                       ; Copia W en la variable AUX

; Obtención de la XOR
RLF AUX,F                       ; Desplaza AUX una posición a la izquierda
RLF AUX,F                       ; Desplaza AUX otra posición a la izquierda
; el bit 4 de AUX está ahora en la misma posición que el bit 6 de REGSTR
; no nos interesan los demás bits, los limpiaremos más adelante
XORWF AUX,F                     ; W XOR AUX, como W contenía el valor de
                                ; REGSTR, esto es AUX XOR REGSTR
                                ; el resultado se guarda en AUX
MOVLW B'01000000' ; Máscara para ver solo el bit 6
ANDWF AUX,F             ; Eliminamos todos los bits excepto el bit 6
                        ; si el resultado es 0, el flag Z estará en 1
; si la XOR dio 0, Z=1, hacemos C=0
; si la XOR dio ≠ 0, Z=0, hacemos C=1
BSF STATUS,C                    ; CARRY = 1, esto no afecta al flag Z
BTFSC STATUS,Z                  ; Si Z=0, saltea la siguiente instrucción
BCF STATUS,C                    ; sino, CARRY = 0

; Desplazamiento a la izquierda y realimentación
RLF REGSTR,F                    ; Desplaza REGSTR una posición a la
                                ; izquierda; C entra por b0 y b7 pasa a C

; Salida del último bit
BTFSC STATUS,C                  ; Si C=0 no ejecuta la siguiente instrucción
BSF PORTB,0                     ; que si C=1 prende el LED
BTFSS STATUS,C                  ; Si C=1 no ejecuta la siguiente instrucción
BCF PORTB,0                     ; que si C=0 apaga el LED
```

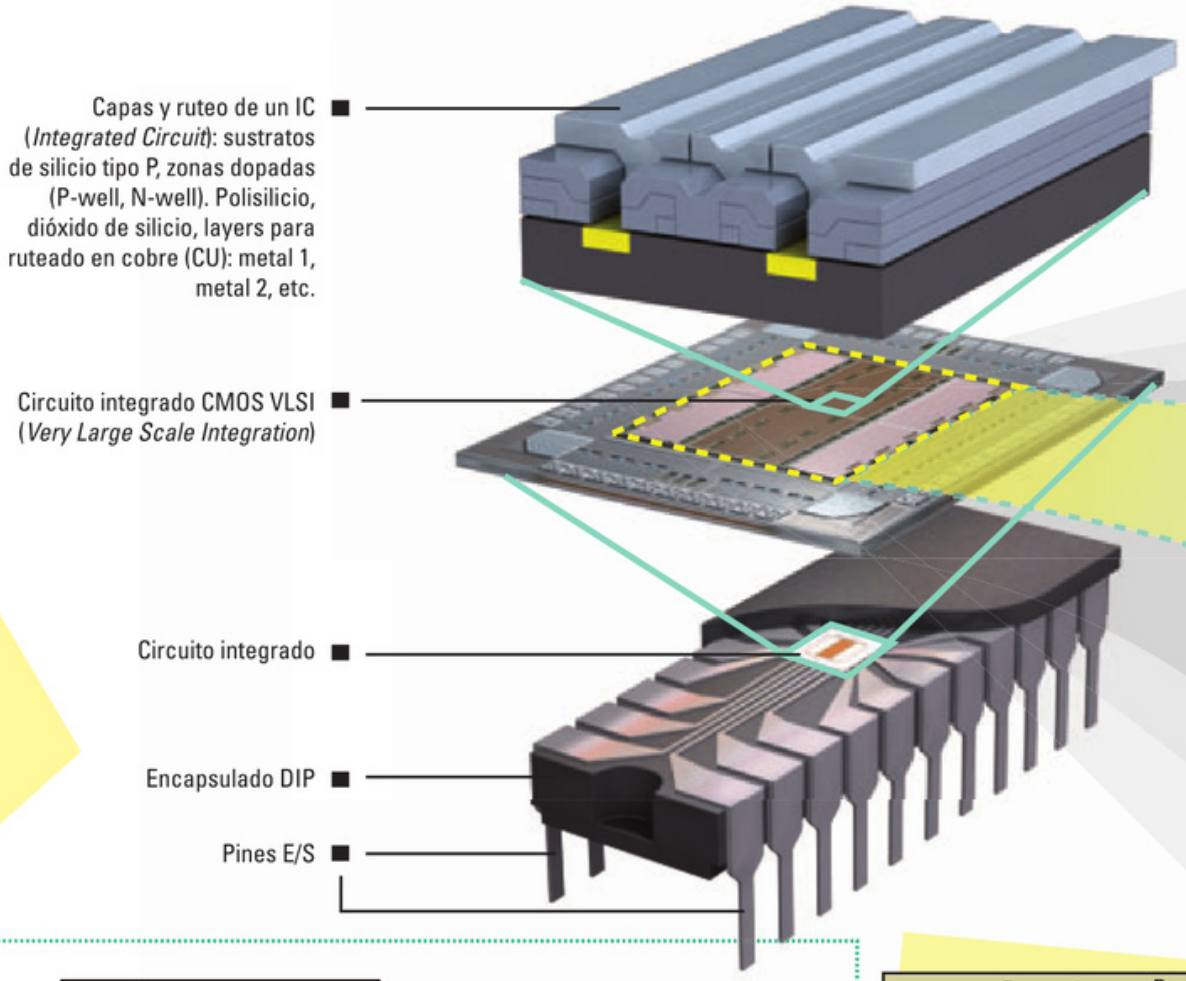
Deberíamos llamar este fragmento de código con cierta periodicidad para lograr el efecto deseado a la salida. Para este fin, podemos usar, por ejemplo, el **timer**, o hacer una demora por software. ■

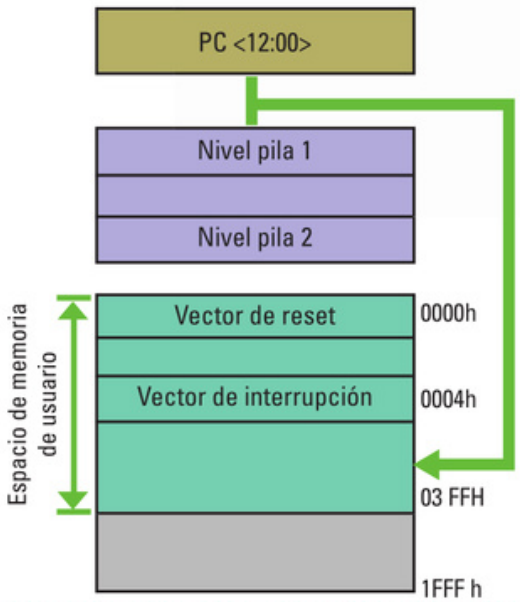
> INSTRUCCIONES

La instrucción "BTFSS STATUS, C" nos permite observar el estado del CARRY.
La instrucción "BTFSS STATUS, Z" realiza la misma operación con el flag de cero.
Ambas instrucciones saltean (no ejecutan) la siguiente instrucción si el bit es 1.
La instrucción complementaria, "BTFSC ...", efectúa el salteo si el bit es 0.

PIC16F POR DENTRO

En esta infografía podemos apreciar la arquitectura interna de un PIC16F84 de 18 pines. Es una arquitectura del tipo Harvard, con almacenamiento físico separado para instrucciones (bus de 14 bits) y datos (bus de 8 bits). Posee un conjunto de instrucciones reducido (RISC) de 35 instrucciones, memoria auxiliar EEPROM de 64 x 8 bits, 2 puertos de E/S, Watchdog y un temporizador (Timer0).



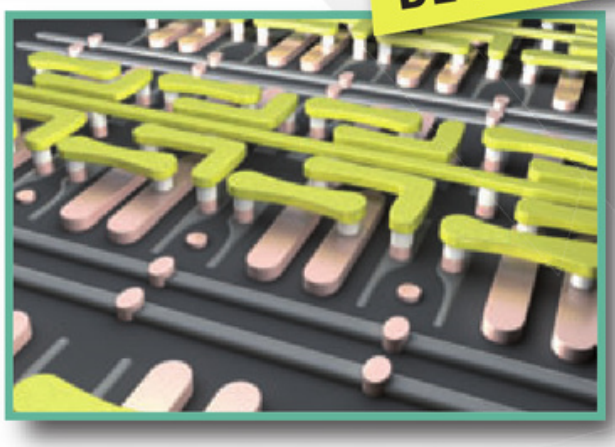


Memoria FLASH de datos

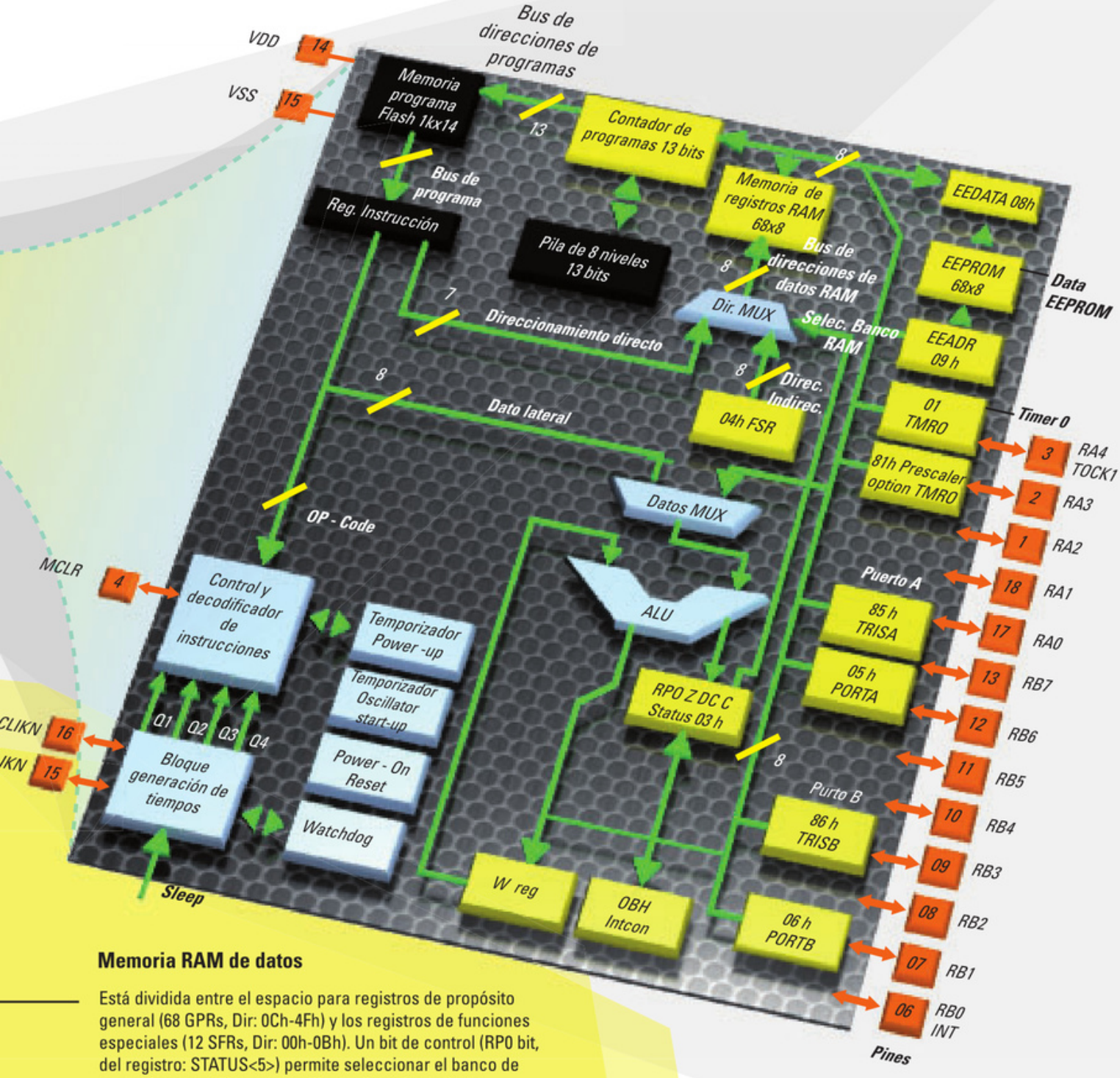
Esta arquitectura posee un PC (*Program Counter*) de 13 bits, capaz de direccionar 8 k x 14 bits de memoria de programa. Para este PIC, solo se implementa físicamente la primera porción de 1 k x 14 bits (0000h-3FFFh). Acceder por encima de esta dirección causará un acceso circular. El Vector de RESET está en 0000h, y el de interrupciones en 0004h.

	Banco 0	Banco 1	
00 h	Dir. Ind.	Dir. Ind.	80 h
01 h	TMR0	OPTION	81 h
02 h	PCL	PCL	82 h
03 h	STATUS	STATUS	83 h
04 h	FSR	FSR	84 h
05 h	PORT A	TRIS A	85 h
06 h	PORT B	TRIS B	86 h
07 h	-	-	87 h
08 h	EEDATA	EECON1	88 h
09 h	EEADR	EECON2	89 h
0A h	PCLATH	PCLATH	8A h
0B h	INTCON	INTCON	8B h
0C h	68 Registro de propósito general	Mapeados (acceso) en el banco 0	8C h
4F h			CF h
50 h			D0 h
7F h			FF h

ARQUITECTURA INTERNA DE PIC16F84

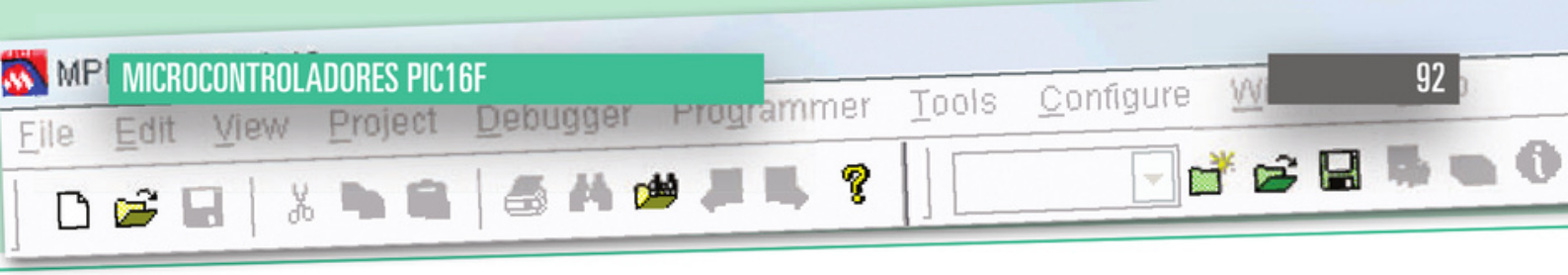


Un microcontrolador como el PIC16F84A es un circuito integrado (IC) conformado por un bloque microprocesador o CPU (Unidad Central de Procesos), memoria de programa Flash EEPROM , memoria de datos RAM y puertos de E/S y periféricos.
La CPU o núcleo microprocesador consta de una Unidad de Control (UC), una Unidad Aritmético Lógica (ALU), registros internos e interfaz a memoria. La ALU es la encargada de realizar las operaciones aritméticas básicas (resta, suma, división y multiplicación) y de operaciones lógicas (OR, NOT, AND, etc.).
La Unidad de Control es la que maneja a la ALU enviándole las órdenes en la secuencia en que deben ser ejecutadas y también se ocupa de transportar los resultados obtenidos.



Memoria RAM de datos

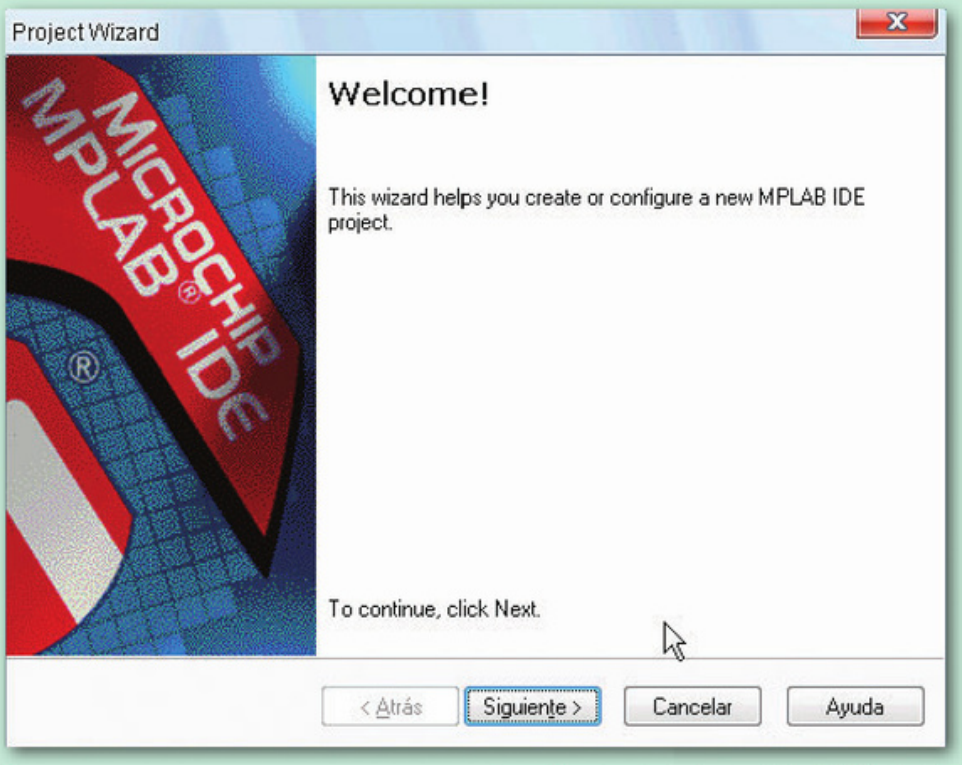
Está dividida entre el espacio para registros de propósito general (68 GPRs, Dir: 0Ch-4Fh) y los registros de funciones especiales (12 SFRs, Dir: 00h-0Bh). Un bit de control (RP0 bit, del registro: STATUS<5>) permite seleccionar el banco de memoria para utilizar por medio de direccionamiento indirecto. Cada banco posee 128 bytes.



MPLAB

MicrochiP LABoratory es el entorno de desarrollo utilizado para crear los programas que se graban en el interior de los microcontroladores PIC. Veamos de qué se trata.

Microchip creó un entorno de desarrollo muy flexible, económico y práctico que permite al usuario crear un programa, simular su funcionamiento y descargarlo dentro de un PIC a partir de una serie de programadores licenciados por la compañía. MPLAB es una herramienta muy completa, ya que integra una serie de aplicaciones de software muy útiles para el desarrollador. Permite hacer **debugging-in-circuit**, es decir, depurar un programa directamente sobre el circuito electrónico, y controlarlo desde él mismo, además de correrlo paso a paso o por segmentos de código.

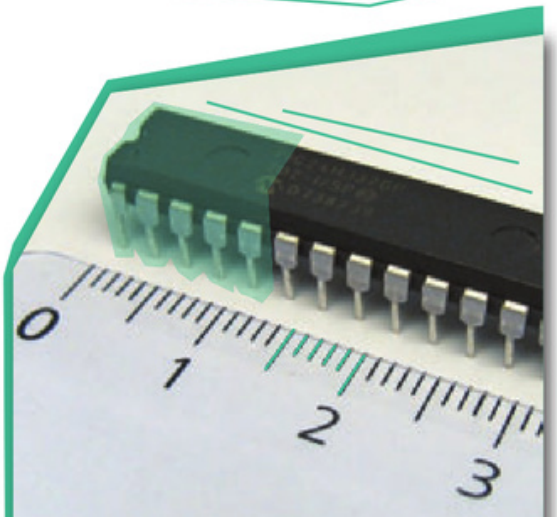


> **Una captura de pantalla del Project Wizard de MPLAB, donde observamos el entorno y el primer paso en la creación de un proyecto.**

Creación de un proyecto con MPLAB

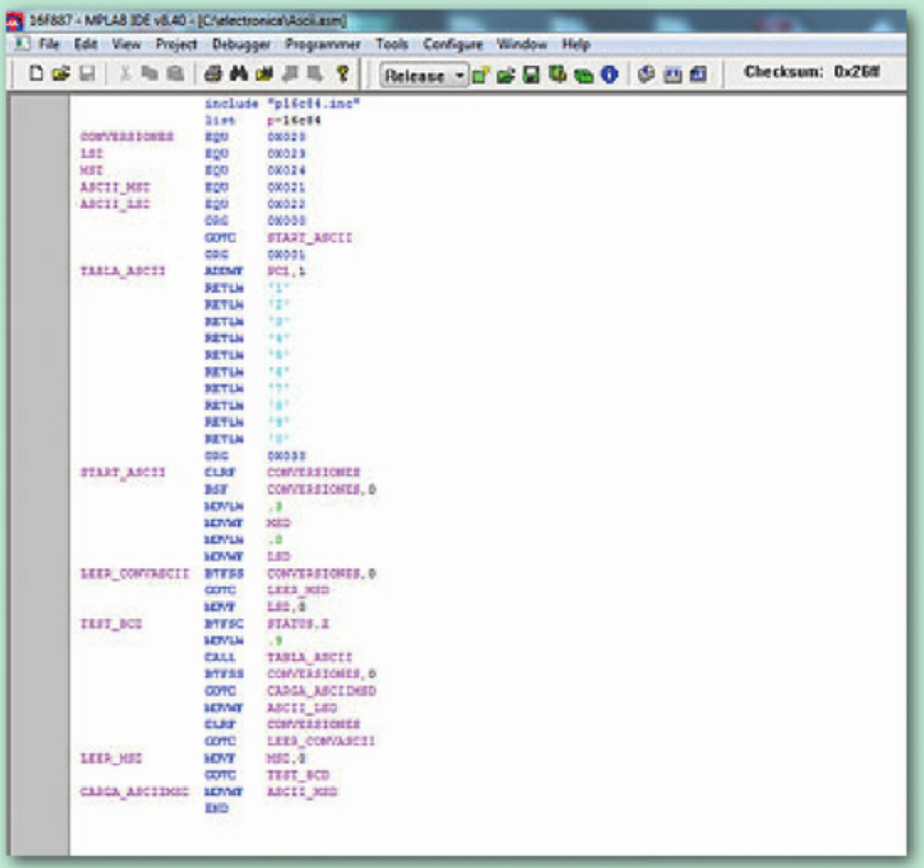
Para generar un programa en MPLAB, deberemos realizar los siguientes pasos
Nos dirigimos al menú **Project/ Project Wizard**. Una vez desplegado el mensaje de bienvenida, pulsamos el botón **Siguiente**.
En el **paso 1** se nos pide seleccionar el tipo de dispositivo que usaremos en la ventana **Device**.
A continuación, elegimos el **toolsuite**, es decir, el tipo de compilador/ensamblador, que por defecto es el **MPASM**, que es el **Assembler**. Luego, presionamos **Siguiente** para continuar.
En el **paso 3** se nos pide crear un proyecto nuevo o reconfigurar el actual. Para crear uno nuevo, pulsamos **Browse**. Se despliega una ventana donde asignamos un nombre a nuestro proyecto y lo guardamos en el directorio **Microchip**. Esta acción hace que volvamos al paso tres, donde presionamos **Siguiente**.

Acto seguido, se nos pide adjuntar un archivo que hayamos creado. Podemos tomar los archivos **template**, que son plantillas creadas por Microchip para facilitar la escritura de un programa. Existen tantos templates como modelos de microcontroladores. Para



seleccionar uno, ingresamos en la carpeta **[Microchip/MPAM Suite/Template/Code]** y elegimos el microcontrolador que sea conveniente. Luego pulsamos **[ADD]**, lo que nos permitirá agregar el archivo marcado a nuestro proyecto. En la ventana **[Summary]** pulsamos **[Finalizar]**. De este modo tendremos creado nuestro primer proyecto, que se mostrará en la ventana de trabajo.

Las directivas son órdenes al ensamblador y no forman parte del set de instrucciones del microcontrolador.



> Vemos el uso de las directivas en una porción de código. Éstas nos permiten controlar ciertas opciones del proceso de ensamblado.

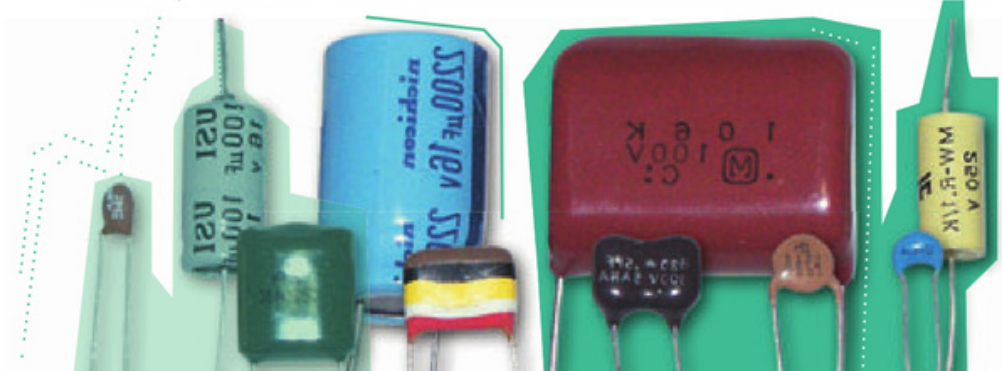
Las directivas

En los programas escritos en lenguaje Assembler, existen las llamadas directivas: se trata de órdenes que se le asignan al ensamblador y que no forman parte del set de instrucciones del micro.
Una directiva indica cómo realizar el proceso de traducción. Las más utilizadas son las siguientes:

- **ORG**: determina a partir de qué posición de memoria de programa se deberá cargar el código ubicado a continuación de ella.
- **LIST**: indica para qué modelo de microcontrolador hemos escrito el programa.
- **EQU**: permite establecer una equivalencia entre una etiqueta y un valor o posición de memoria.
- **INCLUDE**: permite incluir un archivo externo, con extensión **INC**; por ejemplo, los archivos de definiciones de los registros de funciones especiales.
- **END**: indica que se debe terminar el proceso de ensamblado en ese punto; se usa al final de todo programa en Assembler.
- **BANKSEL**: permite seleccionar un banco de registros de forma automática. Genera el código Assembler necesario para modificar los bits **RP0** y **RP1** que controlan el cambio de página.

> DIRECTIVA CBLOCK

La directiva CBLOCK nos permite definir más de una etiqueta a las que podemos establecerles un valor o una posición de memoria. Actúa igual que la directiva **EQU**, pero si definimos los nombres dentro de un bloque **CBLOCK**, los valores son asignados automáticamente por el ensamblador. En el momento de definir un bloque de variables, comenzamos con la directiva **CBLOCK** y la dirección inicial del bloque, luego se finaliza con la sentencia **ENDC**.



El MPLABSIM

Una herramienta muy útil que integra MPLAB es el MPLABSIM: un simulador del microcontrolador, que nos permite convertir nuestra PC en un PIC virtual para correr el programa sobre él, y nos da la posibilidad de observar el estado de sus registros. MPLABSIM se utiliza para realizar las siguientes tareas:

- Correr el programa paso a paso o hasta que encuentre una instrucción de alto (break point), o hacerlo en modo continuo.
- Detener el programa en cualquier momento.
- Ver el valor que adquieren los registros de la memoria RAM.
- Ver el valor que adquieren los registros de propósitos especiales.
- Ver solo el estado de algunos registros (usando la WatchWindow).
- Si el PIC tiene EEPROM, ver su contenido.
- Ver el stack de hardware.

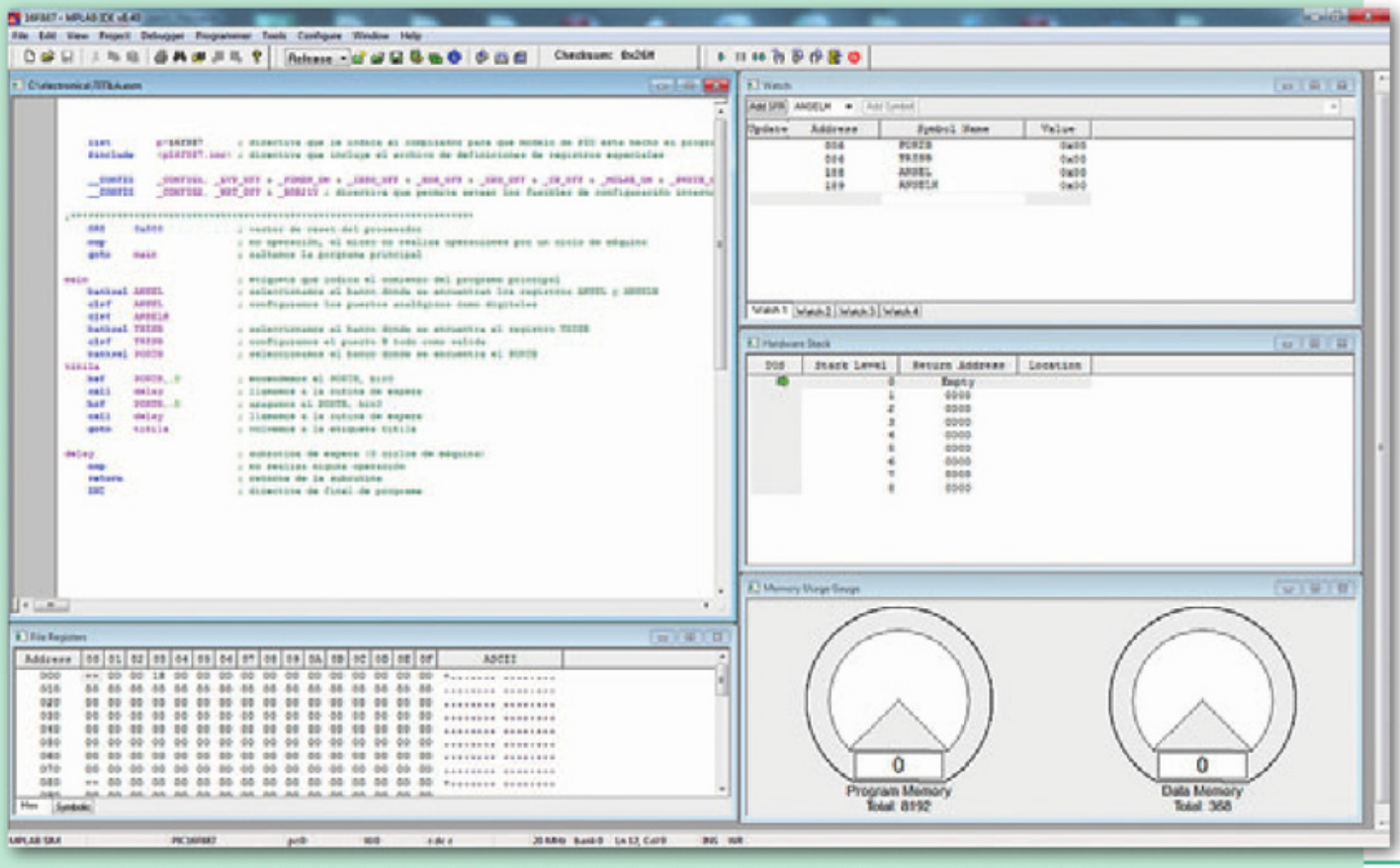
> SIMULACIÓN DE ESTÍMULOS

El MPLABSIM también nos permite simular entradas de datos en los pines del microcontrolador a través del módulo Stimulus (estímulos) del simulador.
A través de los estímulos podemos simular entradas de señales al microcontrolador o entradas de datos en los puertos que tengamos configurados como entradas.
También es muy útil cuando tengamos que simular las funciones de un pulsador conectado al PIC.

Todas estas funciones pueden visualizarse al activar cada una de las funciones que ofrece el simulador. Se presentan como ventanas que podemos acomodar en nuestro ambiente de trabajo para verlas mejor.
En todo proceso de creación de un programa, siempre es bueno hacer una simulación, para verificar que este se comporte del modo esperado en nuestro diagrama de flujo.
Si bien MPLABSIM nos permite aplicar estímulos externos para modificar el funcionamiento del programa, debemos saber que el comportamiento está simulado y que nunca reemplaza a la prueba final sobre el prototipo.
Esto es real para todos los simuladores, pues en la realidad, pueden aparecer eventos no esperados, consecuencia de variables externas, como los siguientes:

- Diseño del PCB
- Material del PCB
- Longitud de las pistas del PCB
- Efecto rebote de pulsadores
- Ruido por emisiones electromagnéticas cercanas, etc.

Sin embargo, un simulador es una herramienta excelente cuando queremos ver el funcionamiento interno del microcontrolador al correr nuestro programa. ■

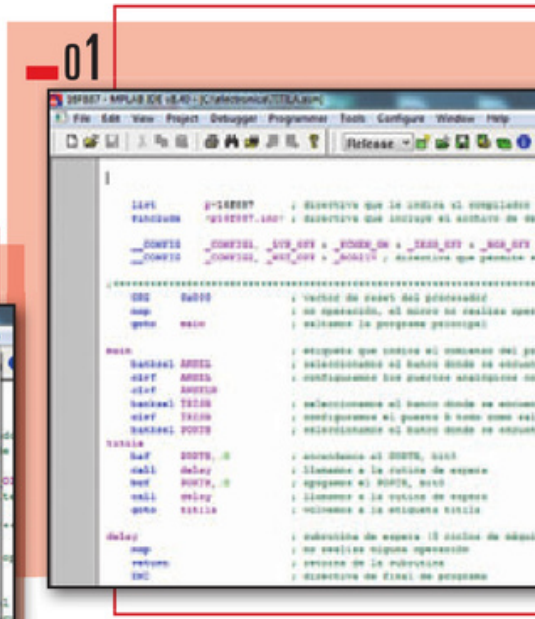


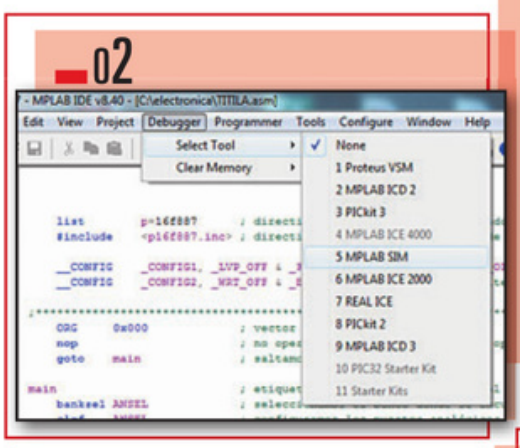
> El MPLABSIM en acción. Con este software, podemos generar un entorno de análisis para un microcontrolador.

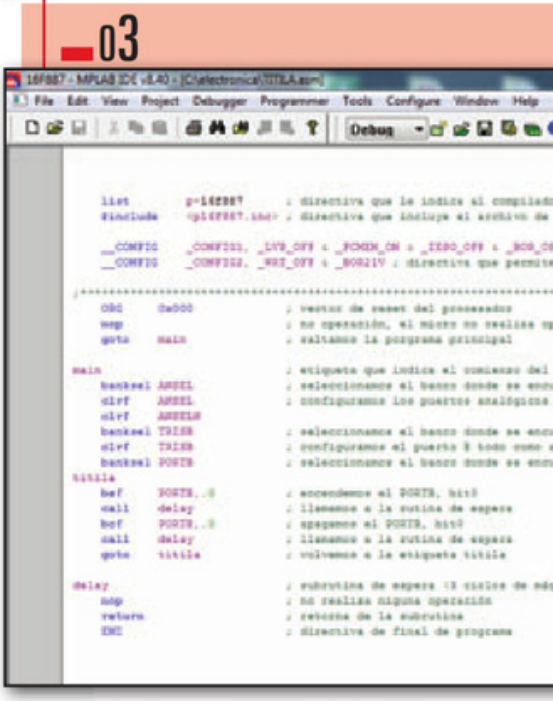
PRÁCTICAS CON MPLABSIM

En este apartado utilizaremos un ejemplo sencillo para aprender a simular código con MPLABSIM.

La idea es utilizar la simulación por software para depurar el código antes de grabarlo en el PIC. Una vez compilado, podemos ver el estado de los registros a medida que corremos el programa. Además, este software es útil para visualizar en forma gráfica la memoria que necesitamos del microcontrolador. ■





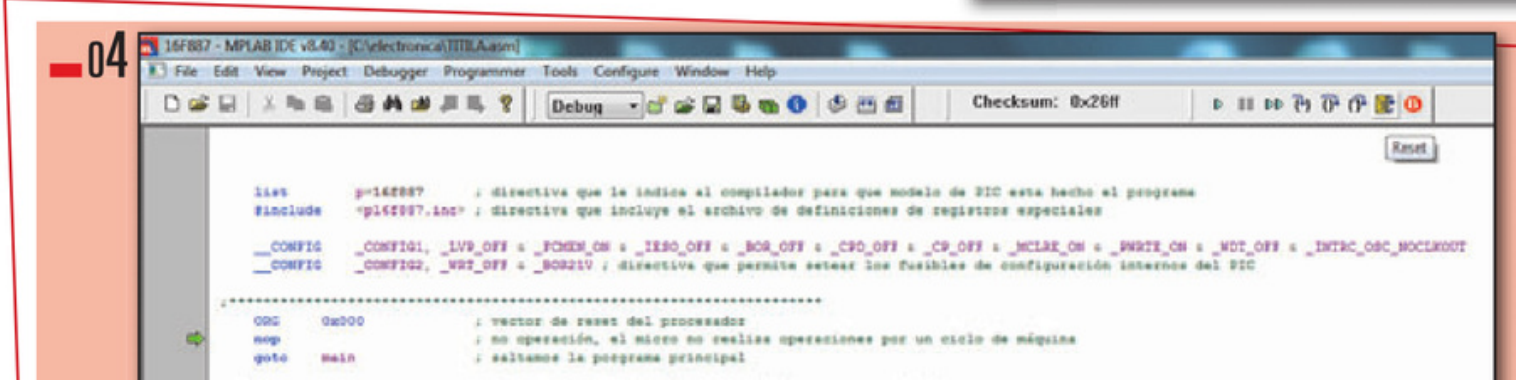


01. Vamos a crear un programa muy sencillo, que consiste en encender y apagar el estado del puerto RB0 de un PIC16F887. Tengamos en cuenta que hay que configurar tanto los puertos analógicos como los digitales mediante el registro ANSEL, como se muestra en el código en la imagen.

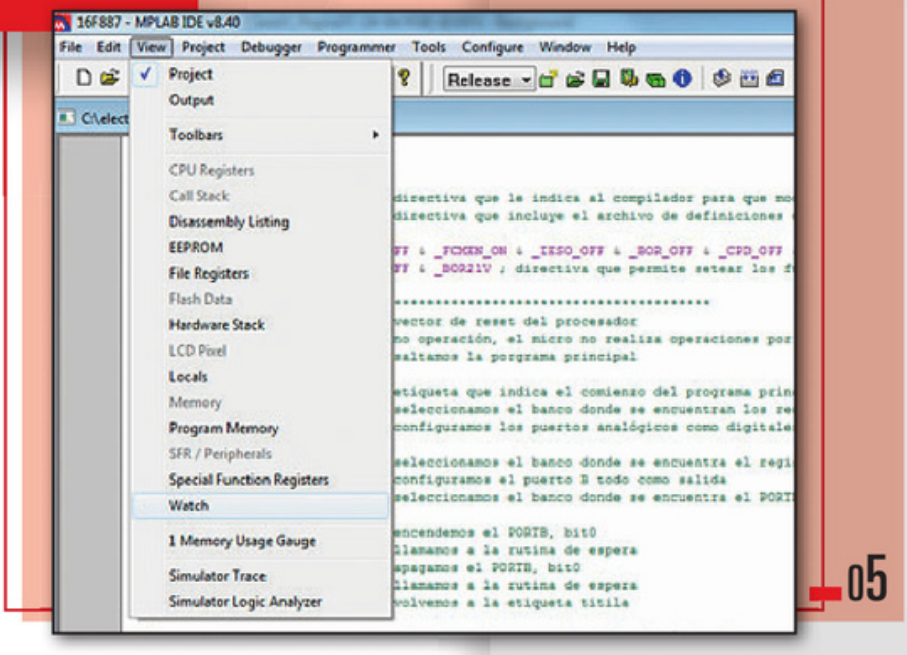
02. Para activar el MPLABSIM, nos dirigimos a la barra de herramientas y hacemos clic sobre [Debugger], para luego ir a **Select_tool** y elegir **MPLAB SIM**. Siempre es necesario compilar el programa previamente, para asegurarnos de que no tenga errores de sintaxis.

03. Para activar y detener el simulador, usaremos los controles indicados, que nos permiten correr el programa en forma continua, de manera animada, o paso a paso. Este último método es particularmente útil cuando tenemos programas complejos donde un mismo registro toma distintos valores.

04. En primer lugar debemos resetear virtualmente el PIC, pulsando el botón del mismo nombre, con lo cual el MPLABSIM pasa a ejecutar la primera instrucción. Nótese que todavía no se ejecuta la simulación, simplemente se posiciona el puntero en la primera línea de código.

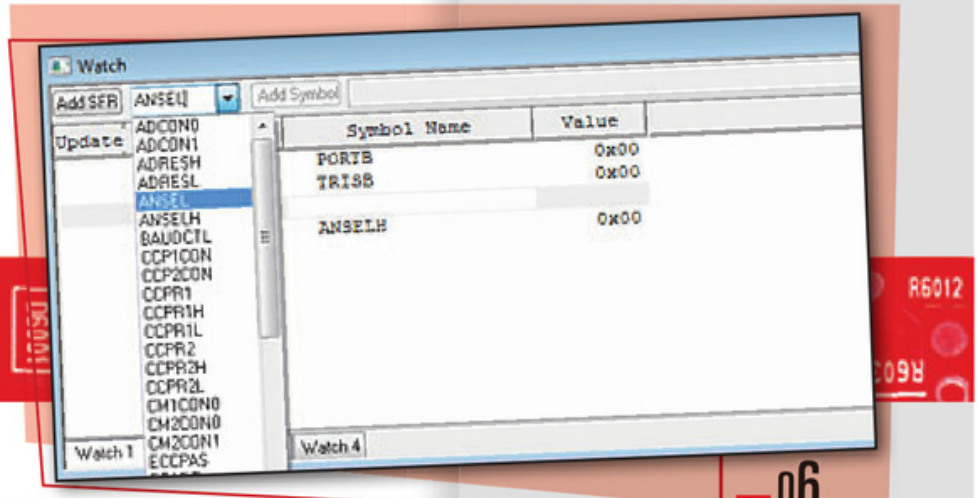


05. Antes de iniciar el funcionamiento, seleccionamos los registros más importantes afectados por el programa, para monitorear su estado usando la función **WATCH**. Para hacerlo, vamos a la barra de menús y hacemos clic sobre **VIEW**; luego elegimos la herramienta **WATCH**.



05

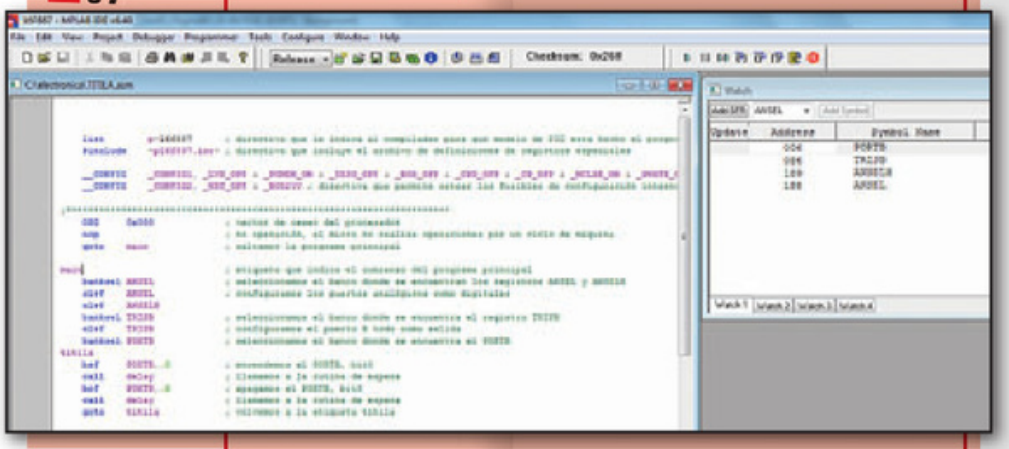
06. Vamos a seleccionar los distintos registros presionando sobre el botón de selección y agregando cada uno por medio del botón **Add SFR**. Para nuestro ejemplo, hemos elegido PORTB, TRISB, ANSEL y ANSELH por ser los más representativos en el programa que hemos creado.



06

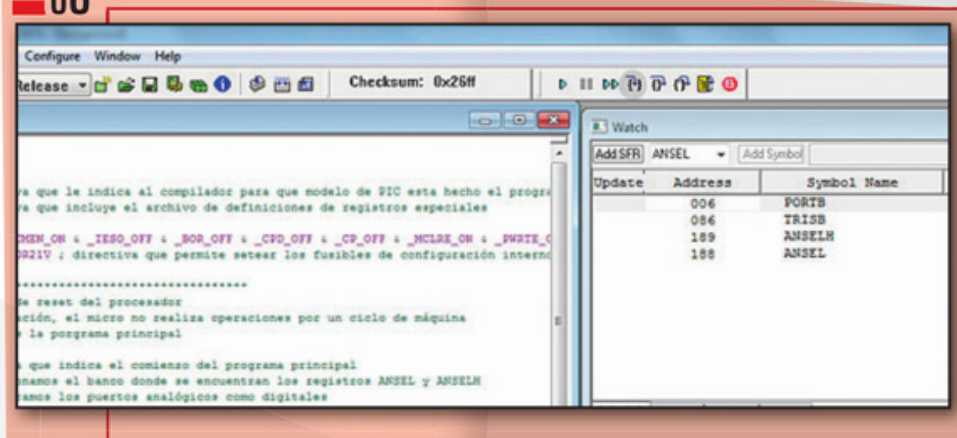
07. Minimizamos la **Watch Window** y la arrastramos hasta posicionarla a un costado de la ventana de trabajo, para así poder monitorear los cambios que se producirán a cada paso del programa. Es útil contar con una resolución de, por lo menos, 1024 x 768 px para que no se superpongan las ventanas.

07



08. Para hacer funcionar el programa e iniciar la simulación, usamos el botón **Step Into**, que nos permite ejecutar cada instrucción paso a paso.
A su vez, observamos el estado que adquiere cada registro en la **Watch Window**. Para inicializar los PORTS, configuramos el estado y lo leemos en el SFR correspondiente; por ejemplo, PORTA, PORTB. Elegimos entradas y salidas en el SFR de control asociado; por ejemplo, TRISA, TRISB. Para aquellos pines asociados al conversor A/D, reclamamos los pines mediante los SFR ANSEL y ANSELH.

08



EL GRABADOR DE PICS

Para grabar los microcontroladores PIC, utilizaremos el programador que armamos en este capítulo, con el cual podremos programar cualquier PIC16F y PIC18F que tenga encapsulados DIP18 o DIP40.

La placa programadora que ya hemos armado en un apartado anterior nos servirá para introducir nuestro programa en la memoria de programa del PIC. Esta placa suele llamarse **quemador**. Como nuestro programador casero no está licenciado por Microchip, el **MPLAB** no nos brinda soporte para él. Por eso, para controlar la placa programadora vamos a usar un software que se obtiene de manera libre desde Internet, denominado **WINPIC800**. También existen otros muy buenos, como **ICPROG** y **WINPICPGM**.

Configurar el programador

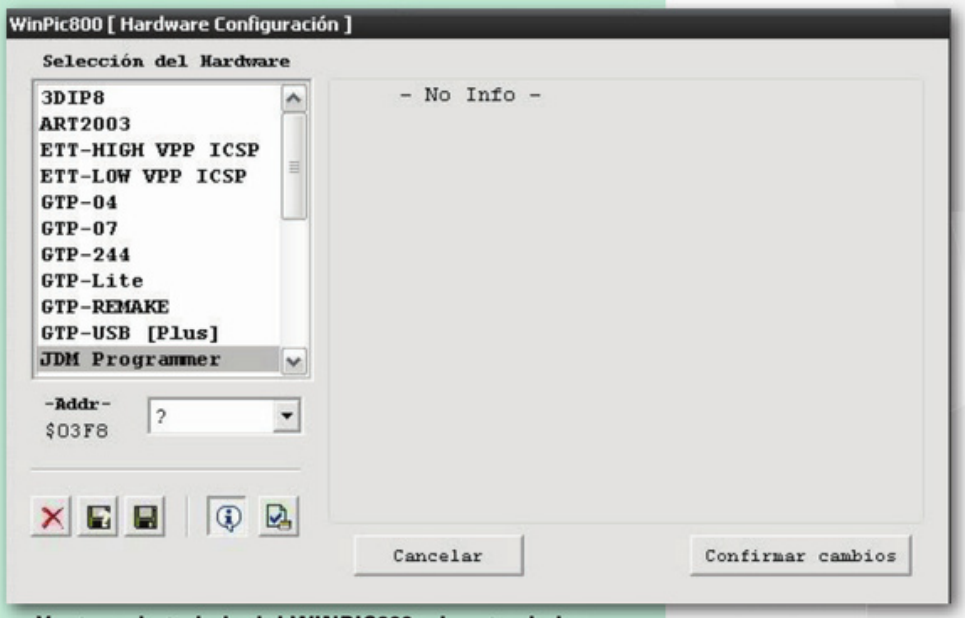
Lo primero que tenemos que hacer es configurar el hardware y el software. Como este software controla una gran cantidad de programadores, necesitamos setear el nuestro, yendo al menú **Configuración/Hardware**.
En la ventana que se despliega, elegimos el programador **JDM Programmer** y configuramos el puerto **COM** donde esté conectada nuestra placa.
Toda esta operación ya deja configurado el software del **WINPIC800** para controlar nuestro programador.

Grabación, lectura y borrado de un PIC

El proceso de grabación de un PIC es muy simple: solo tenemos que cargar el archivo HEX (resultado del ensamblado/compilación), para lo cual vamos a **Abrir** y lo seleccionamos.
Una vez cargado, podemos grabarlo desde el icono **Programar Todo**, mientras que para borrar el PIC, vamos a **Borrar Todo**, y para leer su contenido, usamos **Leer Todo**.

> VERIFICACIÓN

Luego de que terminamos de grabar el PIC, el software de programación realiza una verificación para constatar que no se produjeron errores en la grabación. Esta verificación lo único que hace es comparar el archivo HEX con los datos grabados en el micro y, si todos son iguales, será exitosa.
Esta funcionalidad del programador se puede desactivar, pero aconsejamos tenerla siempre activada para evitar complicaciones.



> Ventana de trabajo del WINPIC800, ejecutando los primeros pasos de configuración del programador.

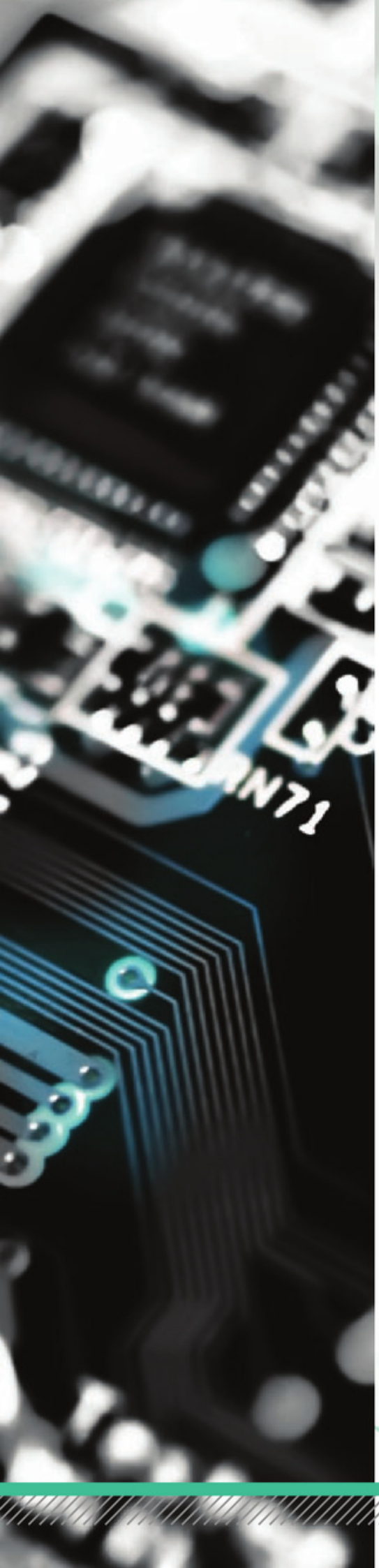
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Caracterice al microcontrolador PIC16F.
2. ¿Qué puertos de entradas posee el PIC16F?
3. Mencione los tipos de memoria correspondientes al PIC16F.
4. Enumere algunas instrucciones útiles en Assembler.
5. ¿Qué cantidad de instrucciones posee el PICF16?
6. ¿Para qué sirve MPLAB?
7. ¿De qué forma podemos crear un proyecto con MPLAB?
8. ¿Qué son las directivas?
9. ¿Para qué podemos utilizar MPLABSIM?
10. Mencione las ventajas de utilizar MPLABSIM.

EJERCICIOS PRÁCTICOS

1. Realice la programación de un contador de un dígito.
2. Ejecute la programación de un hogar a leña.
3. Cree un esquema de un PICF16 por dentro.
4. Instale y ejecute MPLAB.
5. Simule código con MPLABSIM.



CAPÍTULO 4

PIC18F

Características de la familia PIC18F
MPLAB C18
Placa experimental para PIC18LF4620
Consideraciones de armado
Periféricos externos
Contador PWM
Periféricos del PIC 18LF4620
Manejo del display LCD inteligente
Tecnologías de displays LCD
Proyecto de Alarma térmica
Proyecto Alarma térmica en MPLAB C18
Actividades

04

CARACTERÍSTICAS DE LA FAMILIA PIC18F

Son fabricados por la firma Microchip y se ofrecen con una amplia gama de funcionalidades, destacándose por su sencillez y bajo consumo.

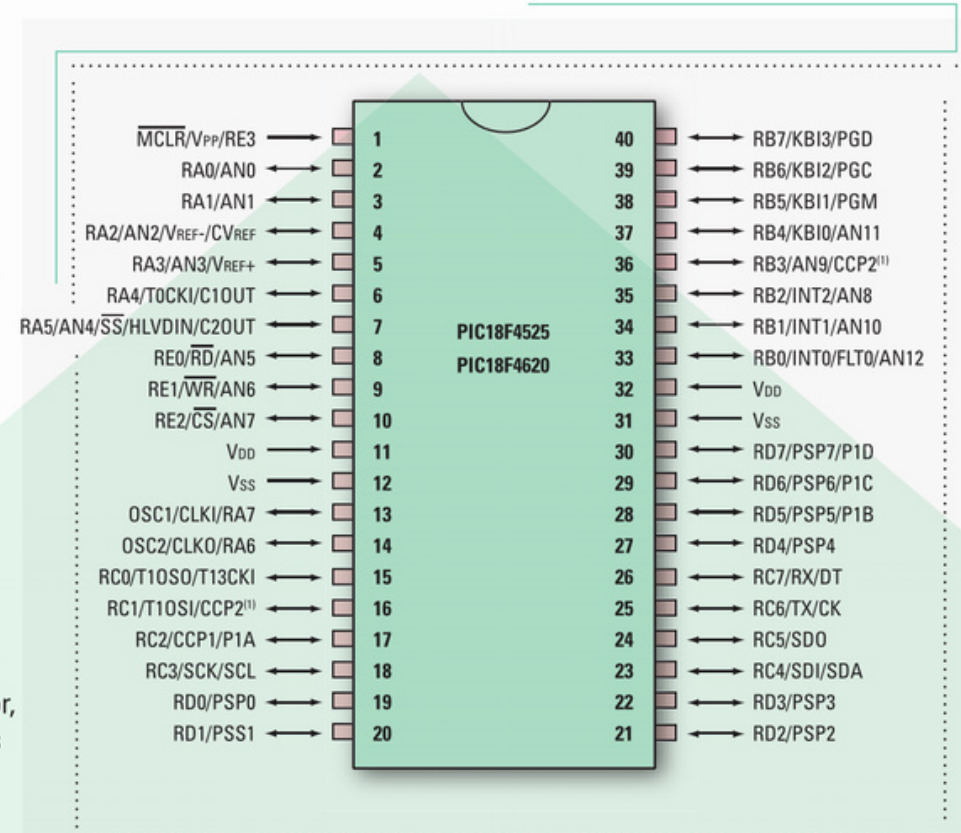
Los microcontroladores **PIC18F** pertenecen a la familia de rango medio avanzado de los micros de 8 bits de Microchip. Poseen un repertorio de 75 instrucciones en modo estándar, más 8 instrucciones especiales que solo trabajan cuando el microcontrolador opera en modo extendido (con memoria de programa interna y externa). El procesador sigue siendo de filosofía **RISC** (set de instrucciones reducido), buses de arquitectura **Harvard**, construido en pastilla de tecnología **CMOS**, frecuencia de operación de hasta **40 MHz,** hasta **32 KWord** de memoria **Flash** de programa, conversores A/D integrados y manejo de interrupciones. Son ideales para aprender las bases de la programación de estos dispositivos. Su nuevo set de instrucciones, que ampliaba la capacidad de las **35** de la familia PIC16, le permitía al programador de lenguaje Assembler una mayor comodidad al realizar el código. Pero uno de los detalles más importantes es que PIC18 se pensó para ser programado en lenguajes de alto nivel, como C, y fue por eso que Microchip desarrolló el hardware a la vez que lo hacía con el compilador, para amoldarlo a las características de este último, al cual denominó C18. Así, PIC 18 se transformó en el primer microcontrolador diseñado para ser programado en lenguaje C.

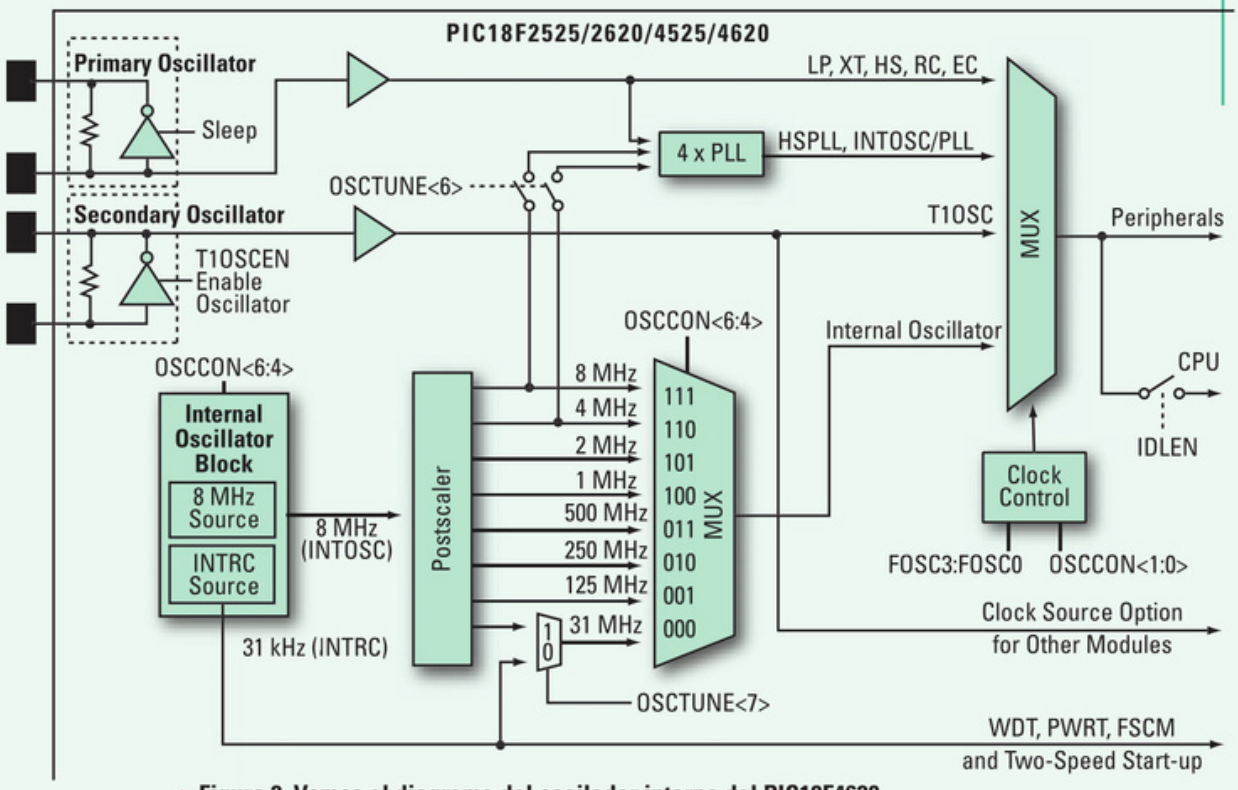
Microcontrolador PIC18F4620

Es uno de los elementos de la familia **PIC18F**. Posee un encapsulado de 40 pines del tipo **DIP** (*Dual In-Line Package*), cuya disposición podemos observar en la **Figura 1**.
El PIC 16F4620 se destaca del resto de los que integran la familia por presentar las siguientes características:

- 32 KWord de memoria de programa
- 3968 bytes de memoria RAM
- 1024 bytes de memoria de datos en EEPROM
- 13 canales A/D, el cual es de 8 bits
- 1 timer de 8 bits y 3 de 16 bits.

Figura 1. Aquí vemos la disposición de los pines del microcontrolador y sus funciones.





> **Figura 2. Vemos el diagrama del oscilador interno del PIC18F4620.**

Tipos de oscilador

El **PIC16F4620** tiene 10 modos distintos de oscilador, que se seleccionan mediante la palabra de configuración. Hay que programar los bits de configuración **FOSC3**: FOSC0 para seleccionar un modo, el cual puede ser:

- LP: Cristal de baja potencia
- XT: XTAL/ resonador
- HS: XTAL/ resonador de alta velocidad
- HSPLL: XTAL/resonador de alta velocidad con PLL activo
- RC: capacitor/resistor con salida de FOSC/4 sobre RA6
- RCIO: capacitor/resistor con Port I/O sobre RA6
- EC: reloj externo con salida FOSC/4 sobre RA6
- ECIO: reloj externo con Port I/O sobre RA6

Esta familia de dispositivos incluye un circuito PLL (*Phase Locked Loop*) o lazo enganchado de fase.

- INTIO1: oscilador interno con salida FOSC/4 sobre RA6 y Port I/O sobre RA7
- INTIO2: oscilador interno con Port I/O sobre RA6 y RA7

Recordemos que los bits de configuración nos permiten establecer las características especiales del microcontrolador: tipo de oscilador, protección contra lectura de la memoria de programa y watchdog timer, entre otras.

El oscilador interno

Los **PIC18F4620** tienen un oscilador interno que genera dos señales diferentes, cualquiera de las cuales puede utilizarse como reloj del microcontrolador. Este componente puede eliminar el oscilador externo de los pines **OSC1** o **OSC2**. La salida principal (**INTOSC**) es una fuente de reloj de **8 MHz** que puede emplearse para dirigir el reloj directamente. Esto también gobierna el **postscaler** o divisor de frecuencia programable del **INTOSC**, el cual puede proporcionar un rango de frecuencias de **31 KHz** a **4 MHz**.
La salida **INTOSC** está activa cuando se selecciona una frecuencia de reloj de **125 KHz** a **8 MHz**.

> EL PLL INTERNO

Esta familia de dispositivos incluye un circuito **PLL** (*Phase Locked Loop*) o lazo enganchado de fase. Esta opción puede usarse para que, a partir de un oscilador de baja frecuencia de clock, se obtenga una alta frecuencia de operación interna. Esto es muy útil para reducir las interferencias tipo **EMI** que se generan en el uso de cristales de alta frecuencia de operación. En el modo **HSPLL (oscilador PLL de alta velocidad)**, es posible trabajar con un cristal externo de hasta **10 MHz**.

Circuito de reset

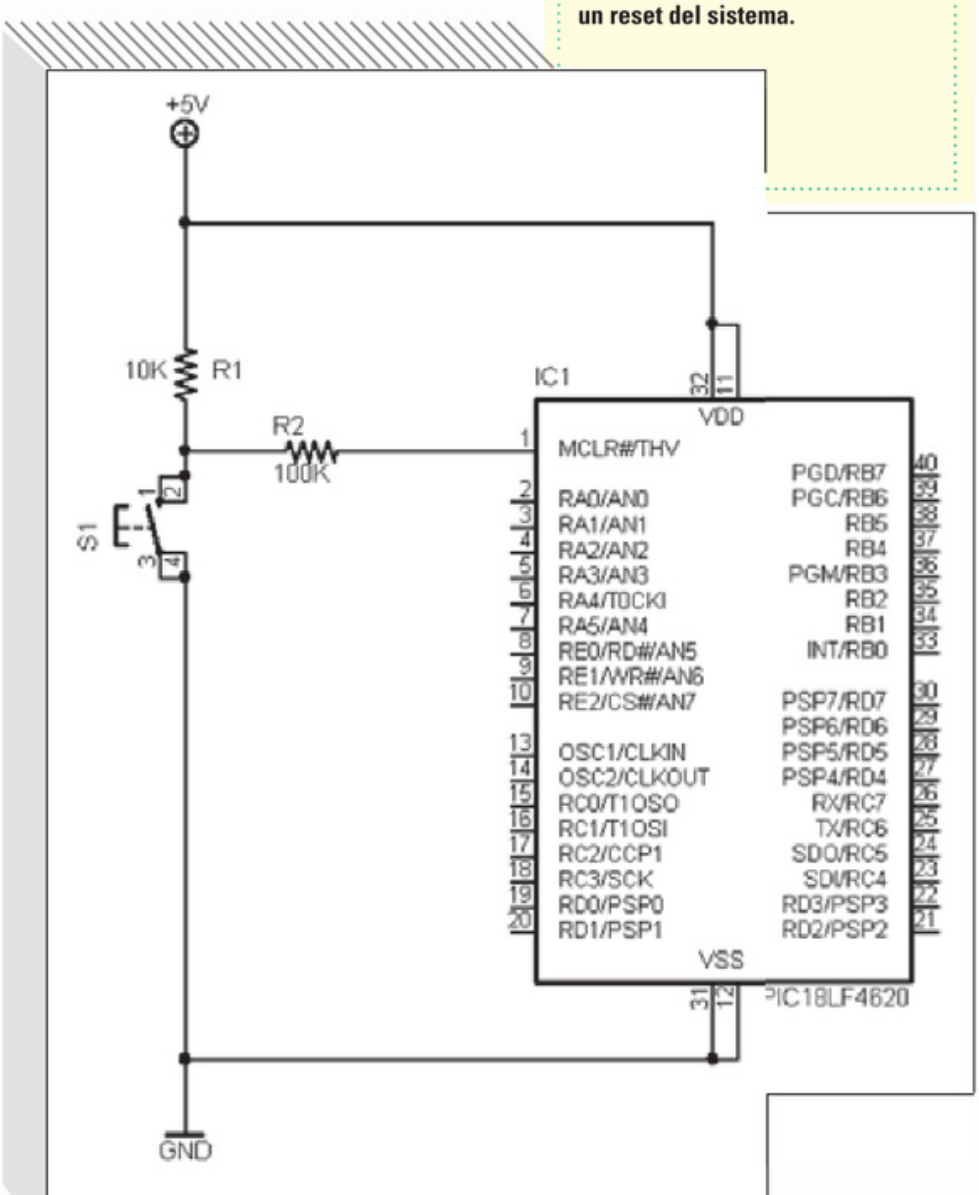
Cuando se produce un reset, el procesador se reinicializa, el programa en ejecución se abandona y el contador de programa se carga con la **dirección 0**. Es allí donde comienza el programa de la aplicación.

Las dos formas más comunes para provocar un reset son:

- Quitar y volver a establecer la alimentación del microcontrolador. Cuando se conecta la alimentación, se produce automáticamente un reset, conocido como **Power-On Reset** (POR).
- Aplicar un nivel lógico bajo en el terminal **MCLR** (Master Clear Reset), que permite provocar un reset externamente en el momento en que queramos, sin necesidad de desconectar el microcontrolador de su alimentación. Para introducir un nivel bajo en este pin, debemos colocar un pulsador que lo lleve a ese estado. En la **Figura 3,** observamos un esquema típico para el circuito de reset.

> MEJORAS TECNOLÓGICAS

Microchip mejoró el hardware de los puertos al colocar dos registros: uno es el denominado **PORT**, para manejar la entrada de datos; y el otro es el **LAT**, para controlar la salida. Esto solucionó el problema que tenía la familia PIC16 en la reconfiguración del puerto.



> Figura 3. La imagen muestra un circuito típico para el reset. Esta configuración aplica un nivel bajo en el pin MCLR, que genera un reset del sistema.

Puertos de entrada y salida

El microcontrolador **PIC18F4620** posee cinco puertos de entrada y salida: **A**, **B**, **C**, **D** y **E**. Debido a que el bus de datos es de **8 bits**, es lógico que el tamaño de los puertos sea el mismo. A diferencia de otros microcontroladores PIC, los puertos **A**, **B**, **C** y **D** son de **8 bits**, mientras que el **E** es de **4 bits**. Cada pin de cada puerto puede configurarse de manera individual como entrada o salida digital para recibir o enviar datos, según las necesidades, a excepción del **RE3**, que solo puede funcionar como entrada.

Pero no todos los puertos son solo **E/S digitales**. Para ahorrar pines en el encapsulado, estos se comparten entre los distintos periféricos.

Por ejemplo, el puerto **A** y parte del **B** pueden usarse como entrada analógica para el conversor A/D incluido en el integrado. Por lo tanto, el **PIC18F4620** tiene un total de **35 puertos I/O**, más uno que únicamente puede operar como entrada.

Sin embargo, de usarse toda la capacidad I/O, deberá emplearse el oscilador interno.

Organización de la memoria

Hay tres tipos de memoria en el **PIC18F4620**: de programa, RAM de datos y EEPROM de datos.

La primera almacena el programa que controlará el funcionamiento del PIC, y es la que grabamos normalmente con el programador de PICs durante el proceso de programación. En la memoria **RAM** se almacenan los datos manejados por el programa que cargamos en el PIC

y los del propio micro. A esta última, se la denomina **RAM del sistema o registros de funciones especiales**. Por su parte, la memoria **EEPROM** de datos es usada también para contener datos, pero a diferencia de la RAM, no necesita alimentación para retener la información, pues es no volátil. Suele usarse para realizar backups de datos.

La memoria de programa

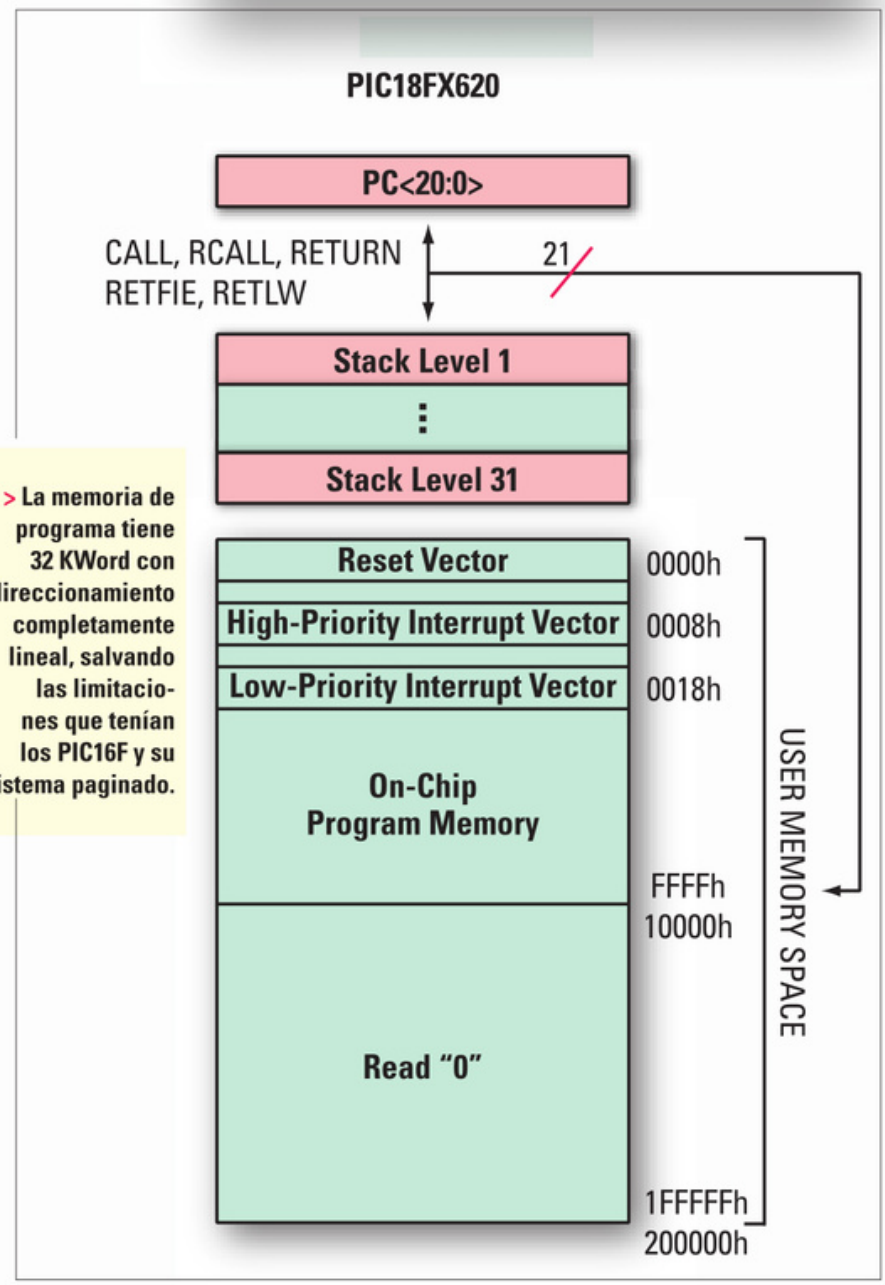
Los microcontroladores **PIC18F** implementan un contador de programa de 21 bits que es capaz de tratar 2 MB de memoria de programa. Esta memoria puede ser tratada como memoria de **16 bits** cuando se leen instrucciones, o como bytes cuando **se leen datos.** Es posible armar tablas de datos a todo lo largo de ella, y estas se encontrarán limitadas solo por la memoria física del PIC.

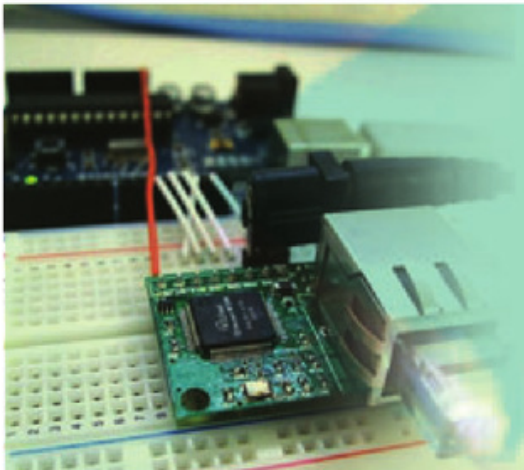
A diferencia de las familias anteriores, los **PIC 18** no tienen un esquema de memoria de programa segmentada en páginas, sino que sus direcciones son lineales, debido a la ampliación de la capacidad del contador de programa. Teóricamente, dentro de los PIC18 es posible implementar una memoria de programa de hasta **1 MWord**, enorme aun para los compiladores de alto nivel.

En este nuevo esquema, las subrutinas pueden ser llamadas desde cualquier lugar de la memoria de programa, y es posible realizar un salto dentro del programa hacia cualquier parte de él sin preocuparnos por la extensión que este deba tener.

El PIC18F4620 tiene un total de 35 puertos I/O más uno que solo puede operar como entrada.

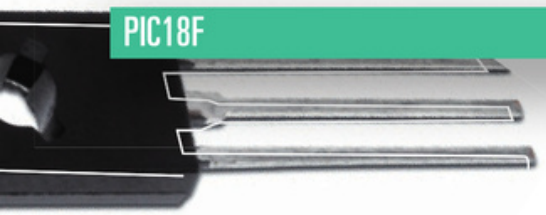
> **La memoria de programa tiene 32 KWord con direccionamiento completamente lineal, salvando las limitaciones que tenían los PIC16F y su sistema paginado.**

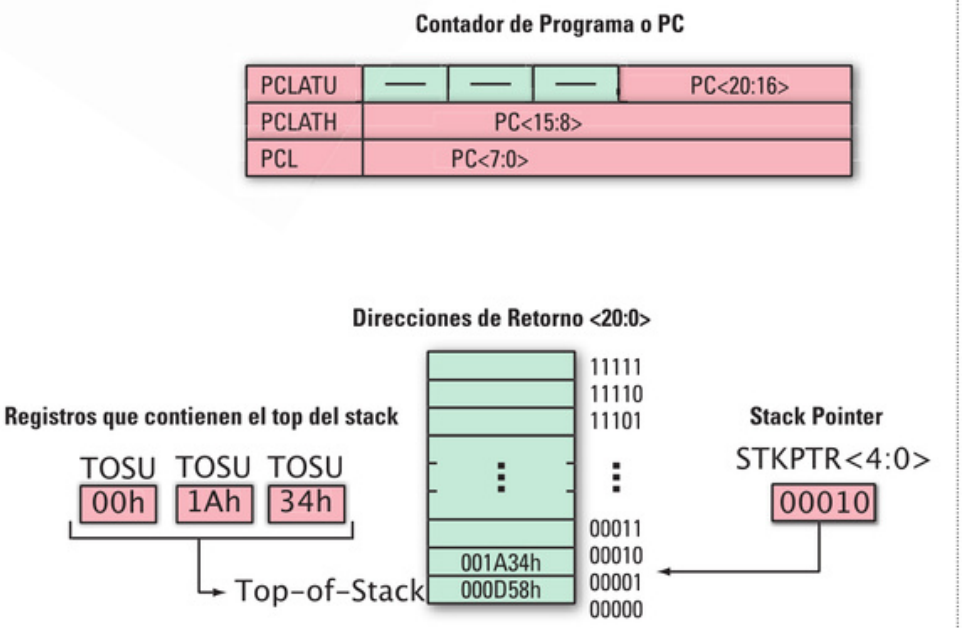




>SOBRE PIC18F

PIC18F se ha transformado en la revolución tecnológica dentro de la familia PIC. Ha servido de base para todos los microcontroladores posteriores, como PIC24, dsPICs y PIC32. Muchas de las características de los PIC18 fueron utilizadas para reformular los PIC16F y generar la nueva familia de "rango medio mejorado" o PIC16F1XXX, presentada por Microchip en el año 2009.





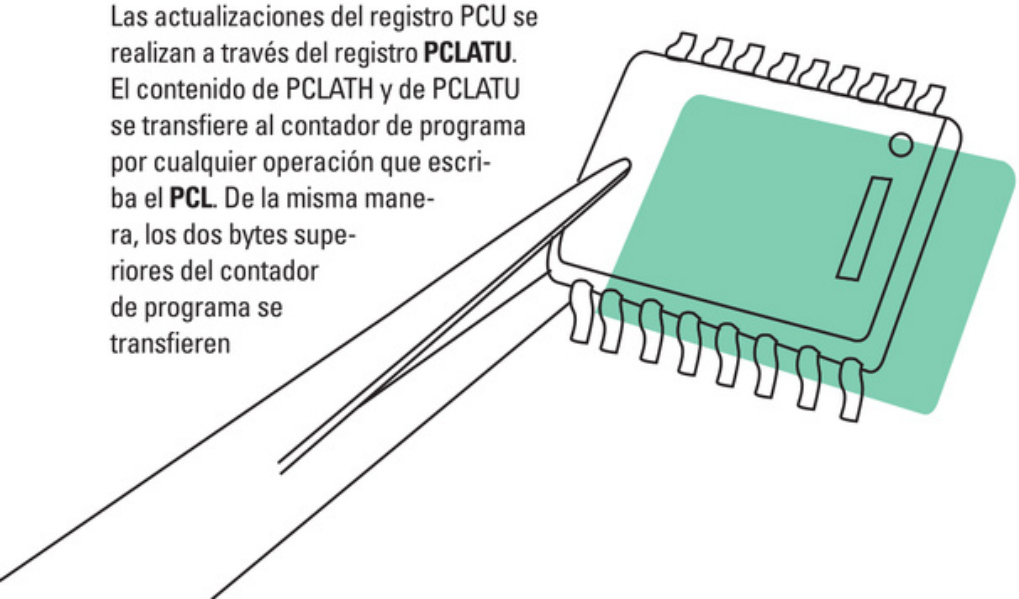
> **Contador de programa y pila o stack de 31 niveles. El contador siempre tiene la dirección de la próxima instrucción por ejecutar.**

Contador de programa

El **contador de programa** (PC) especifica la dirección de la instrucción que se va a ejecutar. Tiene **21 bits** de ancho, separados en **tres registros de 8 bits**. El byte bajo es conocido como registro PCL. El byte alto, o registro **PCH**, contiene los bits <15:8> del PC. Las actualizaciones del registro PCH se realizan a través del registro de **PCLATH**. El tercer registro o **byte superior** se llama **PCU**, y contiene los bits <20:16>.

Las actualizaciones del registro PCU se realizan a través del registro **PCLATU**. El contenido de PCLATH y de PCLATU se transfiere al contador de programa por cualquier operación que escriba el **PCL**. De la misma manera, los dos bytes superiores del contador de programa se transfieren al PCLATH y al PCLATU por cualquier operación que lea el PCL.

El contador de programa (PC) especifica la dirección de la instrucción que se va a ejecutar.



Memoria de datos

La memoria de datos en los dispositivos **PIC18** se implementa como **RAM** estática. Cada registro en la memoria de datos tiene una dirección de **12 bits**, lo que permite hasta **4096 bytes** de memoria. Se divide en **16 bancos** con **256 bytes** cada uno. Los dispositivos **PIC18F4620** tienen **16 bancos** completos, para un total de **4096 bytes**.

La memoria de datos contiene los registros de función especial (SFRs) y los registros de funciones generales (GPRs). Los primeros se utilizan para el control y el estado del regulador y de las funciones periféricas, mientras que los segundos se emplean para el almacenamiento de datos y las operaciones destinadas a guardar datos del usuario. Cuando se lee una localización no implementada, aparecen ceros.

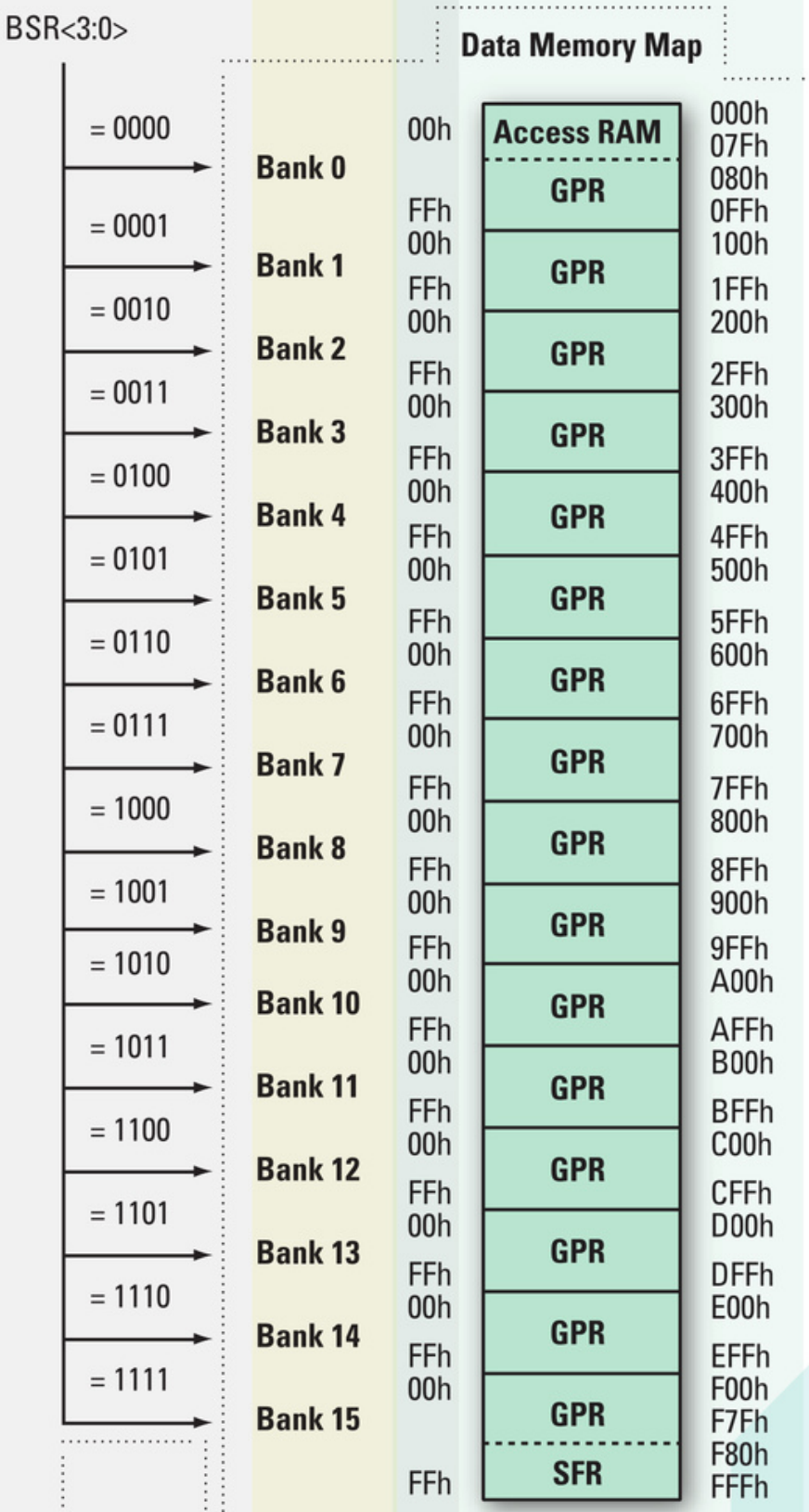
Access bank

El sistema y la arquitectura de las instrucciones permiten realizar operaciones a través de todos los bancos. Es posible tener acceso a la memoria entera de datos por modos de direccionamiento directo, indirecto o puestos en un índice.

Para asegurarse de que los registros de uso general (SFRs y GPRs) se alcancen en un solo ciclo, los dispositivos **PIC18** tienen un banco de acceso directo denominado **access bank**. Se trata de una memoria de **256 bytes** que proporciona el acceso rápido a SFRs y a la parte más baja del banco 0 de GPR sin usar BSR.

Registros de propósito general

En los dispositivos PIC18, es posible almacenar datos en el área **GPR**. Esta es la **RAM** de datos disponible para el uso de todas las instrucciones. GPR empieza en el fondo del banco 0 **(dirección 000h)** y crece hasta la última área de SFR. No se inicializa por un reset y no cambia en ningún otro.



> La memoria de datos está dividida en 16 bancos de 256 posiciones cada uno. Los registros de funciones especiales se agrupan en el último banco (banco 15).

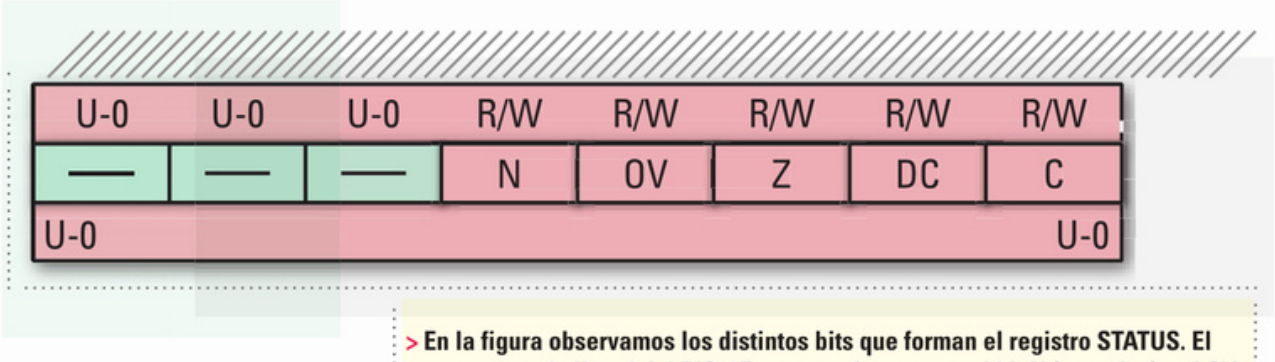
> PIC18 Y PIC16

Es importante aclarar que, a diferencia de PIC16, PIC18 tiene todos los registros de funciones especiales agrupados en las últimas 128 posiciones de la página 15 de la memoria de datos, lo cual le permite al usuario acceder a ellos de forma directa sin tener que cambiar constantemente de bancos, como sucedía en el PIC16.

Registros de funciones especiales

Los registros de función específica (SFR) son usados por la **CPU** y por los módulos periféricos para controlar el funcionamiento del dispositivo. Están implementados como **RAM estática** en la memoria de datos. Los **SFR** empiezan en la última dirección de memoria de datos y se extienden hacia abajo hasta ocupar el segmento superior del banco **15**, de **F80h a FFFh**. Pueden clasificarse en dos grupos: los asociados a la funcionalidad del núcleo del dispositivo y los relacionados con las funciones de los periféricos.





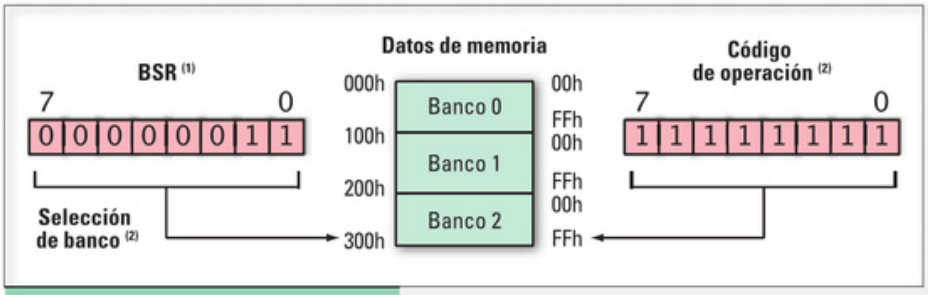
> En la figura observamos los distintos bits que forman el registro STATUS. El esquema es similar al del PIC16F, pero aquí se suman el bit N (negativo) y el OV (overflow).

El registro de estado

El registro **STATUS** contiene el estado aritmético de la ALU. En su interior podemos encontrar distintos bits que indican el estado de un producto aritmético o lógico. Entonces, como en los **PIC16F**, encontramos el bit de cero **Z**, que se pone en **uno** cuando el resultado es **cero**, y los bits de acarreo, tanto para el bit más significativo, como el bit intermedio. Pero, a diferencia de los PIC16, los PIC18 agregan el bit de overflow (**OV**, desbordamiento), que se activa cuando se produce un desbordamiento en el resultado de la cuenta. También agregan el bit Negative (**N**, negativo), cuando el resultado de la operación es negativo. Como con cualquier otro **SFR**, es posible modificarlo con cualquier instrucción. Si el registro **STATUS** es el destino de una instrucción que afecta a los bits **Z**, **DC**, **C**, **OV** o **N**, no escribirá el resultado de la instrucción. En cambio, se actualiza según la instrucción ejecutada. Por lo tanto, el resultado de una instrucción con el registro **STATUS** como destino puede ser diferente de lo previsto. Por ejemplo, **CLRF STATUS** activará el bit **Z**, y el resto de los bits no cambiará.

El registro BSR

Su sigla proviene de *Bank Select Register* (registro selector de bancos). La mayoría de las instrucciones en el sistema del PIC18 hacen uso del puntero de banco, conocido como BSR. Este SFR controla los 4 bits más significativos de la dirección de localización; la instrucción incluye los 8 bits menos significativos. Solo los 4 bits más bajos del BSR están implementados (BSR3:BSR0).



> El registro BSR mediante el cual podemos acceder a cualquier banco de la memoria RAM.



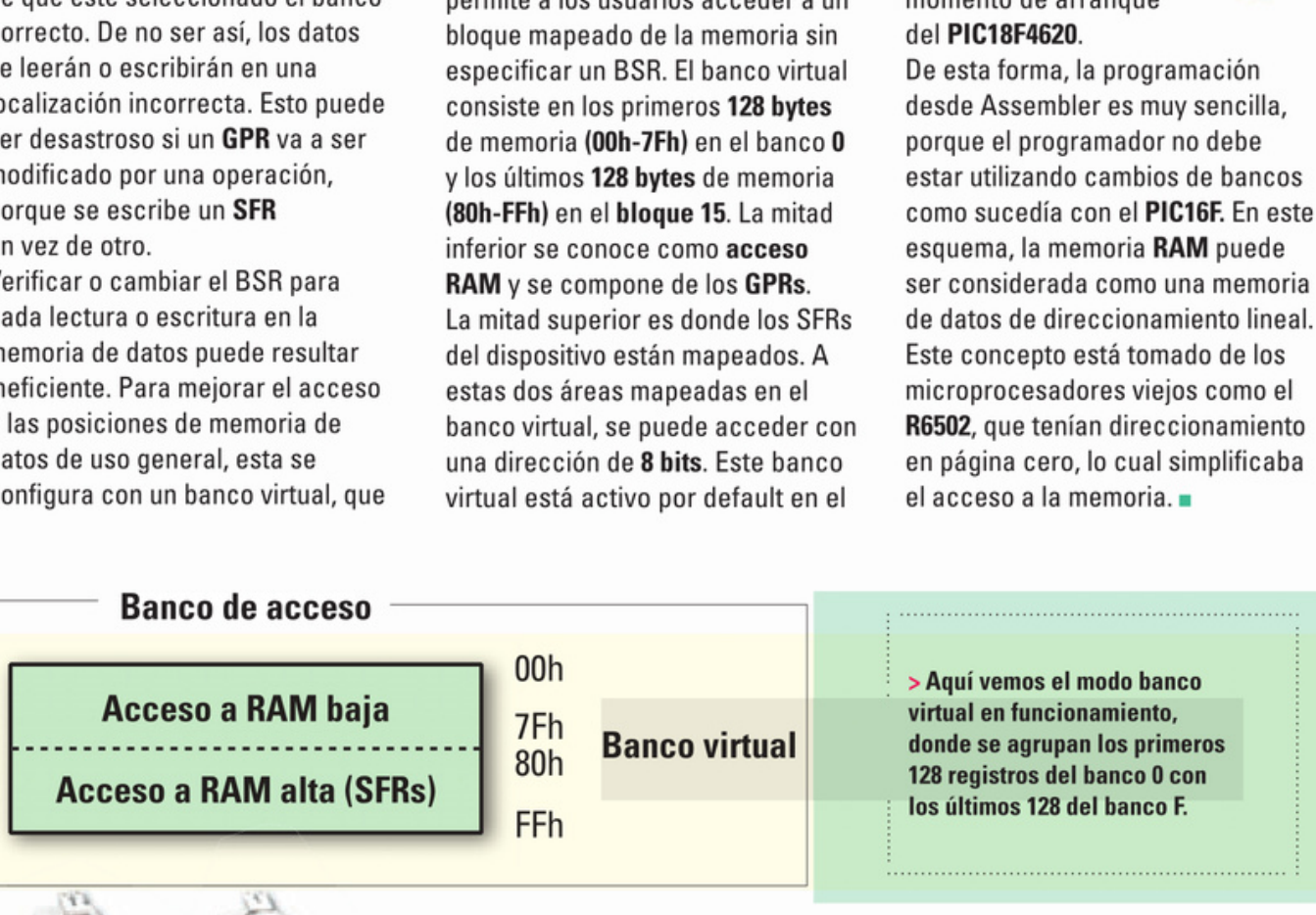


No se utilizan los 4 altos, que leerán 0 y no se pueden escribir. El BSR se puede cargar directamente usando la instrucción **MOVLB**.
El valor del BSR indica el banco en la memoria de datos. Los 8 bits de la instrucción muestran la localización dentro del banco.

Acceso a los bancos

Mientras que el uso de **BSR**, con una dirección de **8 bits**, permite que los usuarios traten la gama entera de memoria de datos, también significa que siempre hay que asegurarse de que esté seleccionado el banco correcto. De no ser así, los datos se leerán o escribirán en una localización incorrecta. Esto puede ser desastroso si un **GPR** va a ser modificado por una operación, porque se escribe un **SFR** en vez de otro.
Verificar o cambiar el BSR para cada lectura o escritura en la memoria de datos puede resultar ineficiente. Para mejorar el acceso a las posiciones de memoria de datos de uso general, esta se configura con un banco virtual, que permite a los usuarios acceder a un bloque mapeado de la memoria sin especificar un BSR. El banco virtual consiste en los primeros **128 bytes** de memoria **(00h-7Fh)** en el banco **0** y los últimos **128 bytes** de memoria **(80h-FFh)** en el **bloque 15**. La mitad inferior se conoce como **acceso RAM** y se compone de los **GPRs**. La mitad superior es donde los SFRs del dispositivo están mapeados. A estas dos áreas mapeadas en el banco virtual, se puede acceder con una dirección de **8 bits**. Este banco virtual está activo por default en el momento de arranque del **PIC18F4620**.
De esta forma, la programación desde Assembler es muy sencilla, porque el programador no debe estar utilizando cambios de bancos como sucedía con el **PIC16F**. En este esquema, la memoria **RAM** puede ser considerada como una memoria de datos de direccionamiento lineal. Este concepto está tomado de los microprocesadores viejos como el **R6502**, que tenían direccionamiento en página cero, lo cual simplificaba el acceso a la memoria. ■

> Para mejorar el acceso general, la memoria se configura con un banco virtual.



> Aquí vemos el modo banco virtual en funcionamiento, donde se agrupan los primeros 128 registros del banco 0 con los últimos 128 del banco F.



> BANCO VIRTUAL

El banco virtual es muy útil porque simplifica la programación. Sin embargo, hay que tomar en cuenta que solo tenemos 128 registros de propósitos generales para almacenar nuestros datos, ya que los otros 128 están ocupados por los registros de funciones especiales. Desde este punto de vista, PIC18 supera ampliamente a PIC16, pues el programador ya no tiene que estar agregando en su código los cambios de página. Los SFRs y los GPRs se encuentran en el mismo banco.

Registros de funciones especiales asociados a los puertos

Los puertos de los microcontroladores PIC18 poseen, como mínimo, tres registros especiales para configurar su función. En el **PIC18LF4620** ellos son:

- El registro **TRIS**
- El registro **PORT**
- El registro **LAT**

El registro **TRIS** es utilizado para indicar la dirección que tendrá cada pin del puerto, y podemos configurarlo como salida digital o como entrada digital. Todos los puertos del PIC tienen asociado este registro y, según sea el puerto, reciben el nombre de **TRISA**, para el puerto A, **TRISB**, para el B, **TRISC** para el C y, así sucesivamente.
Cada bit de este registro configura un pin del puerto; así, el bit menos significativo controla al pin 0 del puerto, y el más significativo, al pin 7. Para indicar que un puerto será empleado como salida, debemos poner a cero todos los bits del registro **TRIS** asociado a él. En cambio, si queremos que el puerto sea usado como entrada digital, debemos llevar todos los bits del registro TRIS a 1.
Por su parte, el registro **PORT** contiene el estado actual de los pines del puerto; es decir que si queremos saber cuál es el nivel lógico de un pin del puerto, tenemos que leer este registro. Al igual que sucede en el caso anterior, los registros de cada puerto se identifican como PORTA, PORTB, PORTC, PORTD y PORTE.

Los puertos de los PIC18 poseen, como mínimo, tres registros para configurar su función.

Es importante destacar que los registros **LAT** contienen el valor actual de los latch que manejan el puerto. Son utilizados en las operaciones que requieran lectura, modificación o escritura de los valores que contienen los pines del puerto.

Registros asociados al puerto A

El **PORTA** es un puerto bidireccional de **8 bits**. Su registro correspondiente para indicar el sentido de los datos es TRISA, por lo que ya sabemos que, colocando un bit de este registro en 1, configuramos el pin como entrada, mientras que si lo hacemos en 0, será como salida.
Pero, además, los pines de este puerto tienen **multiplexadas** sus funciones, dado que la mayoría puede usarse como canales de entrada del conversor analógico-digital que incorporan estos PICs. Entonces, para elegir cuál es el modo de trabajo de las líneas del **PORTA**, existe, además, el registro de control **ADCON1**, que especifica cuáles son los pines que se utilizarán como entrada al conversor, y cuáles, como líneas digitales para datos. Cuando manejemos el conversor analógico-digital del PIC, veremos en detalle el funcionamiento de este registro.

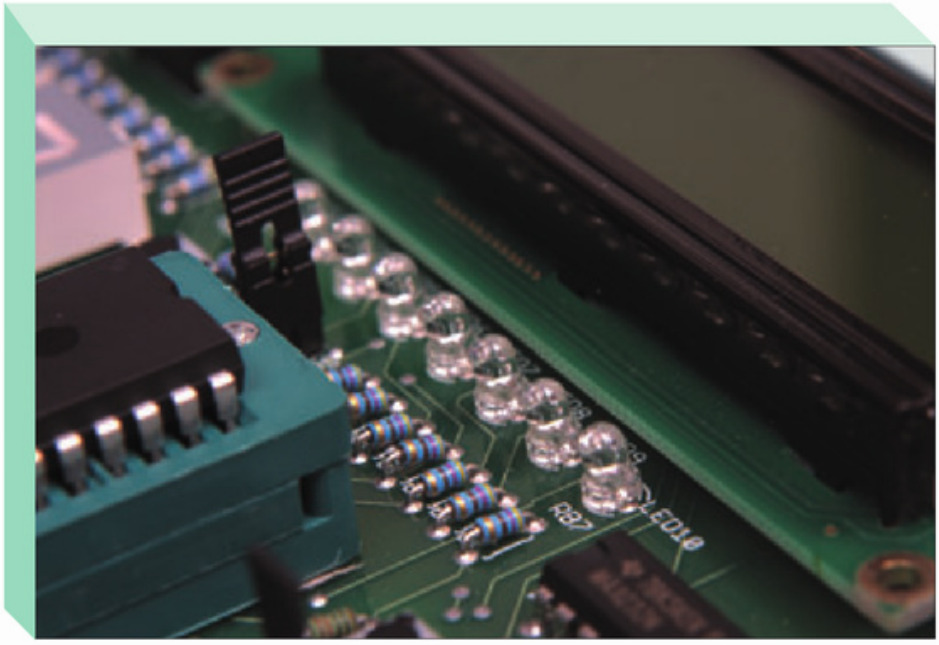
Registros asociados al puerto B

El **PORTB** es también un puerto bidireccional de 8 bits, en el que el sentido de los datos se indica mediante el registro **TRISB**. Al igual que en el PORTA, este puerto también tiene mapeado en la memoria el estado de los **latch** en el registro **LATB**.
Del mismo modo que el **puerto A**, este tiene algunos de sus pines multiplexados para otras funciones.

> Los puertos I/O son los medios utilizados por los microcontroladores PIC para comunicarse con el mundo exterior. Por eso, poseen registros propios que permiten configurar sus diferentes funcionalidades.



El terminal RB0 puede configurarse como una entrada/salida digital o para atender a una interrupción externa.



Para que el PIC pueda excitar una matriz de LEDs conectados al puerto, debemos configurarlo como líneas de salida y controlar su operación con los registros PORT.

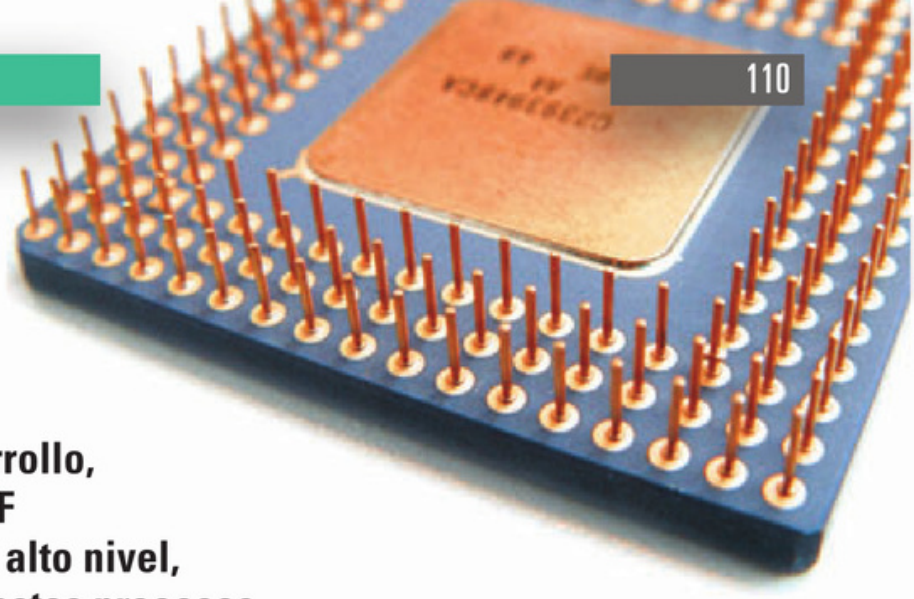
El terminal **RB0** puede configurarse como una entrada/salida digital o, también, para atender a una interrupción externa.
Además, todos los pines del PORTB tienen un **pull-up** interno que es activado por el **bit /RBPU**, que ocupa el séptimo lugar del registro **INTCON2**. Estas resistencias de pull-up son desactivadas automáticamente cuando el puerto es configurado como salida.

Registros asociados a los demás puertos

Los demás puertos del PIC –C, D y E– también tienen asociados los tres registros básicos comentados para A y B. Los **PORTC** y **D** tienen un ancho de 8 bits y, también, son direccionales. En cambio, el **PORTE** tiene un ancho de **4 bits**, y el pin **RE3** solo puede usarse como entrada. Esto quiere decir que no tiene mapeado en memoria el estado de su **latch**. Para obtener más información sobre la manera de configurar los puertos, podemos ver las hojas de datos del PIC, disponibles para descargar desde la página de Microchip.

> En la placa de la fotografía vemos la gran variedad de dispositivos que puede controlar el PIC a través de los puertos I/O. Estos pueden ser LEDs, displays, pulsadores, sensores de temperatura, pantallas LCD, y otros.





MPLAB C18

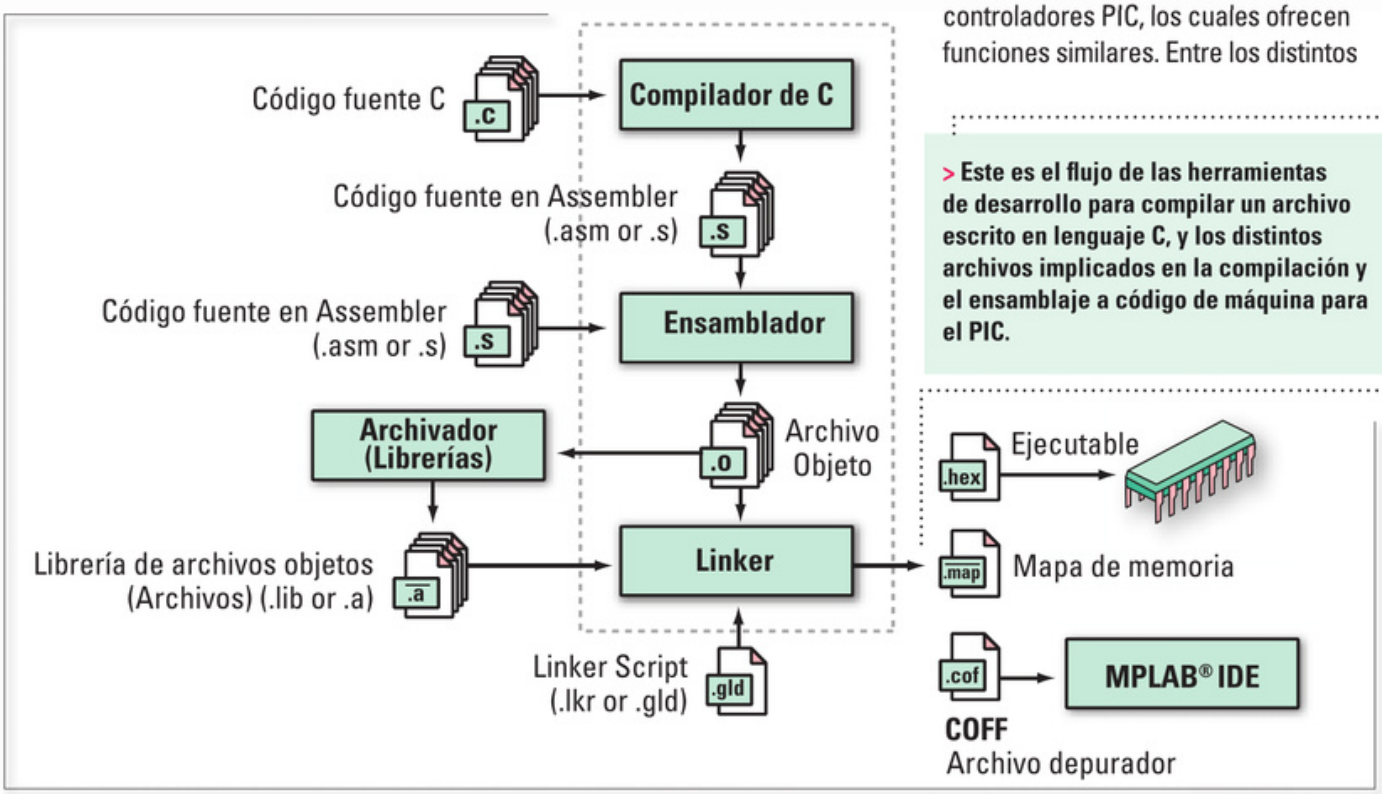
Dentro de este entorno de desarrollo, podemos trabajar con los PIC18F y programarlos en lenguajes de alto nivel, como C. Veamos cómo realizar estos procesos.

Para la programación de microcontroladores, es importante elegir cuál será el lenguaje por utilizar. La elección estará definida por distintos factores, como la velocidad deseada en tiempos de programación, el tamaño del código y la facilidad de programación.
Uno de los motivos más importantes para escoger un lenguaje de alto nivel en vez del ensamblador, es el aumento considerable de la complejidad del programa. En ese caso, es preferible utilizar un lenguaje que se acerque al humano, porque es más fácil de programar, produce código más legible y libera al programador de lidiar con los detalles del hardware, ya que el compilador se ocupa de toda esa tarea. Dentro de todos los lenguajes posibles, se encuentra uno que es muy utilizado por los profesionales: C. Esto se debe a que, normalmente, es mas fácil escribir código C en vez de ensamblador, y dentro de los lenguajes de alto nivel, C es el de más bajo nivel, lo que hace que sea soportado por distintos compiladores para microcontroladores con diversas arquitecturas.

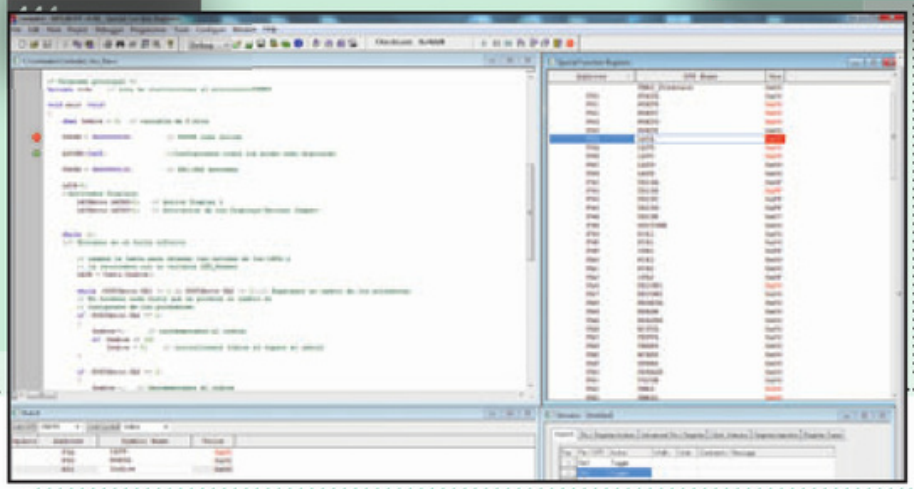
Los compiladores de C para PIC más importantes son los fabricados por CCS, HiTech y Microchip.

Compiladores C para PIC

En la actualidad, existen varios compiladores de C para los microcontroladores PIC, los cuales ofrecen funciones similares. Entre los distintos



> **Este es el flujo de las herramientas de desarrollo para compilar un archivo escrito en lenguaje C, y los distintos archivos implicados en la compilación y el ensamblaje a código de máquina para el PIC.**



Aspecto del entorno de MPLAB cuando trabajamos en lenguaje C. Este software nos permite tener varias ventanas abiertas para seguir el estado del PIC en las simulaciones.

compiladores que podemos encontrar en el mercado, los más destacados son los fabricados por **CCS, HiTech y Microchip**.

El programa CCS C Compiler es un compilador muy potente que soporta todas las familias de PICs y posee su propio entorno de trabajo, desde el cual podemos generar proyectos, simularlos y depurarlos. Este compilador cuenta, además, con una amplia librería de funciones.

Por su parte, **HiTech C Compiler** para PIC18 dispone de un plugin para incorporar en el entorno del **MPLAB IDE**. Si bien es menos conocido, no deja de ser una alternativa viable.

Compilador MPLAB C18

El **MPLAB C18** es el compilador de Microchip para sus **PIC18** y tiene un lenguaje bastante similar al **C** convencional, excepto que se le han agregado diversas adaptaciones para volverlo más apropiado para el ambiente de programación de los PIC. Se maneja perfectamente dentro del entorno visual de MPLAB, y sus características son:

- Solo se utiliza para los PIC18.
- Posee las funciones estándares del lenguaje C.
- Permite la inclusión de lenguaje ensamblador (Assembler).
- Está optimizado para la arquitectura de los PIC18.
- Contiene librerías para comunicaciones **SPI**, **I2C**, **USART** y periféricos externos como LCD inteligentes.
- Ofrece una versión estudiantil gratuita (freeware).

Todas estas características hacen del compilador **MPLAB C18** una herramienta ideal para los desarrollos que podemos realizar con los PIC18.

Tipo de almacenamiento

El **lenguaje C** maneja los tipos de datos básicos, como **int**, **char** y **float**. El lenguaje orientado a los microcontroladores del MPLAB **C18** también los soporta. En la **Tabla 1,** podemos ver los tamaños en bits que ocupan en la memoria. Esto es importante porque el tamaño de las variables puede cambiar de un compilador a otro, de modo que siempre que se escoge un compilador determinado para la programación, es conveniente verificar los tamaños de los tipos de datos que se utilizarán.

Los tipos de datos **char** almacenan un carácter simple, tienen un tamaño de **8 bits** y son ideales para trabajar con los registros del PIC, ya que poseen la misma longitud. Las variables **int** tienen un tamaño de **16 bits**; son adecuadas para operaciones aritméticas con números enteros. Por su parte, para operaciones con mayor precisión tenemos las variables **float**, con un tamaño de 32 bits.

TIPOS DE DATOS ENTEROS

TIPO	TAMAÑO	MÍNIMO	MÁXIMO
Char	8 bits	-128	127
Signed Char	8 bits	-128	127
Unsigned Char	8 bits	0	255
Int	16 bits	-32768	32767
Unsigned Int	16 bits	0	65535
Short	16 bits	-32768	32767
Unsigned Short	16 bits	0	65535
Short Long	24 bits	-8.388.608	8.388.607
Unsigned Short Long	24 bits	0	16.777.215
Long	32 bits	-2.147.483.648	2.147.483.647
Unsigned Long	32 bits	0	4.294.967.295

Tabla 1. Tipos de datos enteros del MPLAB C18.
El compilador permite realizar operaciones matemáticas entre datos tipo char; el resultado también puede ser un char.

```
// Ejemplo de tratamiento de cadenas en RAM y ROM

#include <string.h>
// almaceno mi cadena en ROM
const rom unsigned char micadena[ ]="hola mundo";

// almaceno copia en RAM
char strbuffer[21];

void main(void)
{
    strcpy(strbuffer micadena); // incorrecto
    strcpypgm2ram(strbuffer, micadena); // correcto
    for (;;);
}
```

> **Ejemplo de cómo declarar cadenas de string en memoria de programa y de cómo almacenarla en un buffer en la memoria RAM de datos.**

Cadenas de string en memoria ROM y RAM

Ya vimos los distintos tipos de datos que podemos manejar en el lenguaje, de modo que ahora nos concentraremos en las cadenas de string o caracteres. Esto nos permite manejar frases o mensajes en nuestro PIC con todas las funcionalidades que posee el **lenguaje C**. Recordemos que en **C**, un **string** es un caso especial de **array**, y que en la declaración puede asignarse o no una dimensión del **string**. Si se opta por **no** asignar una dimensión, recordemos que el array de strings siempre finaliza con un **carácter nulo**.

La cadena de string puede ser almacenada en la memoria de programa (**ROM**) o en la de datos (**RAM**). El lugar donde se la guarda se indica con los calificadores **rom** o **ram**. Veamos un ejemplo de cómo almacenar un string constante en la memoria de datos:

```
Const rom unsigned char micadena[]
= "hola mundo"
```

Este ejemplo crea una cadena de caracteres constantes en la memoria **ROM**, y si bien no indicamos una longitud, sabemos que el último carácter será un **null**.

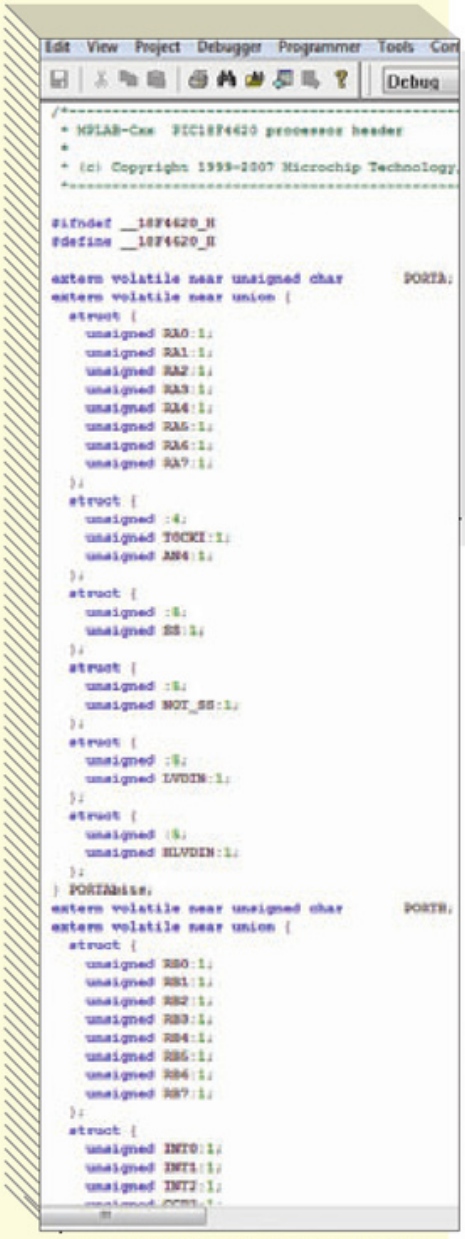
Del mismo modo en que utilizamos los calificadores **far** y **near** para variables, también podemos usarlos para crear cadenas de caracteres en las posiciones de memoria que queramos.

Estructuras y uniones

Si queremos agrupar variables de diferentes tipos, podemos utilizar las **estructuras** y las **uniones**. El compilador **C18** permite declarar estructuras anónimas dentro de las uniones. La utilidad principal de esta característica es la declaración de variables. Por ejemplo:

```
Union tpuerto{
        Unsigned char valor
        struct{
                unsigned bit0:1;
                unsigned bit1:1;
                unsigned bit2:1;
                unsigned bit3:1;
                unsigned bit4:1;
                unsigned bit5:1;
                unsigned bit6:1;
                unsigned bit7:1;
        };
}byte;
```

Esta es la declaración de una unión que, dentro, contiene una **estructura anónima**, porque no posee nombre. Esto permite utilizar la variable en el plano de los bytes y también a nivel de bits. Así, resulta similar al estándar en C para los campos de bits (bitfields), pero el hecho de que la estructura esté dentro de la unión permite modificar la variable, tanto en el nivel de



> **Vemos cómo el compilador utiliza las uniones y las estructuras para definir los registros especiales (SFR) del PIC. En este extracto del archivo p18f4620.h, vemos de qué modo se definen los bits del PORTA.**

los bytes como para acceder a los bits. Entonces, si queremos escribir un valor en el byte, utilizamos la variable de la siguiente forma:

```
// byte
byte.valor = 0xFF;
```

Del mismo modo, podemos acceder a los bits de esa variable:

```
// bit
byte.bit0 = 0;
byte.bit7 = byte.bit0;
```

Este recurso es muy utilizado para definir los registros que maneja el microcontrolador.
Vemos cómo el compilador utiliza las uniones y las estructuras para definir los registros especiales (SFR) del PIC. En este extracto del archivo p18f4620.h, vemos de qué modo se definen los bits del PORTA.

Punteros de memoria

Los punteros son variables que contienen la dirección a una zona de memoria donde reside un determinado tipo de dato. En el compilador C18 podemos tener punteros hacia la memoria de datos o hacia la de programa. Un puntero a la memoria de datos tiene una longitud de 16 bits y se declara de la siguiente manera: **char *puntero**. Este puntero apunta a variables del tipo **char** en la memoria de datos. Un puntero a memoria de datos no puede apuntar a un dato que se encuentre en la memoria de programa; para eso, debemos definir un puntero hacia esta memoria, por ejemplo: **near rom char *rom_puntero**. Este contiene direcciones de datos que están en la memoria de programa y solo tiene acceso a las direcciones de memoria inferiores a 64 K. Si queremos usar punteros para direcciones mayores, debemos cambiar el prefijo **near** por **far**.

TAMAÑO DE PUNTEROS

TIPO DE PUNTERO	EJEMPLO	TAMAÑO
Puntero a memoria de datos	Char * ptr;	16 bits
Puntero a memoria de programa cercano	rom near char * n_rom_ptr;	16 bits
Puntero a memoria de programa lejano	rom far char * f_rom_ptr;	24 bits

Ejemplos de distintos tipos de punteros con respecto a la memoria de datos y de programa, y las longitudes en bits de cada uno de ellos.

Directivas

Al igual que el lenguaje C, el compilador C18 posee directivas al procesador, que permiten realizar el control del código. La directiva **#pragma** permite a los compiladores definir sus propias directivas. El compilador C18 la utiliza para definir algunas secciones de memoria.
El compilador C18 divide las secciones de memoria en memoria de programa o memoria de datos. A su vez, estas secciones están subdivididas en dos zonas:

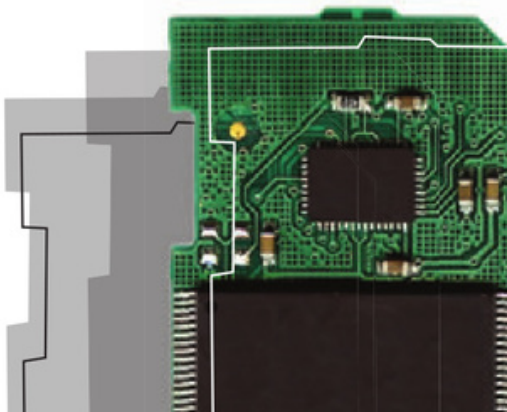
Memoria de programa
- **code:** instrucciones en memoria de programa.
- **romdata:** datos en memoria de programa.

Memoria de datos
- **udata:** variables de usuario estáticas sin inicializar.
- **idata:** variables de usuario estáticas inicializadas.

Para definir las rutinas de interrupciones, debemos utilizar la directiva **#pragma interrupt**, mediante la cual le indicamos al compilador que las instrucciones escritas a continuación comenzarán en el vector de interrupciones.

> DIRECTIVA #PRAGMA CONFIG

Permite acceder a la zona de los bits de configuración del PIC. Los bits de configuración se utilizan para configurar muchas funcionalidades del PIC; por ejemplo, podemos establecer el tipo de oscilador, activar protecciones a la memoria de programa y realizar muchas otras funciones que poseen los PIC18F.



Primer programa en C

Aprenderemos el entorno de programación en C a través de un ejemplo práctico. Luego, realizaremos una simulación con el **MPLAB SIM**.

Introducción al ejercicio

Para nuestro primer programa en C, vamos a realizar un contador decimal, que posee dos pulsadores para incrementar o disminuir la cuenta. El estado de la cuenta será visualizado por medio de un **display de 7 segmentos**.

En el esquemático del ejercicio (**Figura 4**) podemos observar los pulsadores, uno de los cuales tendrá la función de incrementar la cuenta, y el otro, de disminuirla. El **display de 7 segmentos** estará conectado al puerto **B** del **PIC18LF4620**.

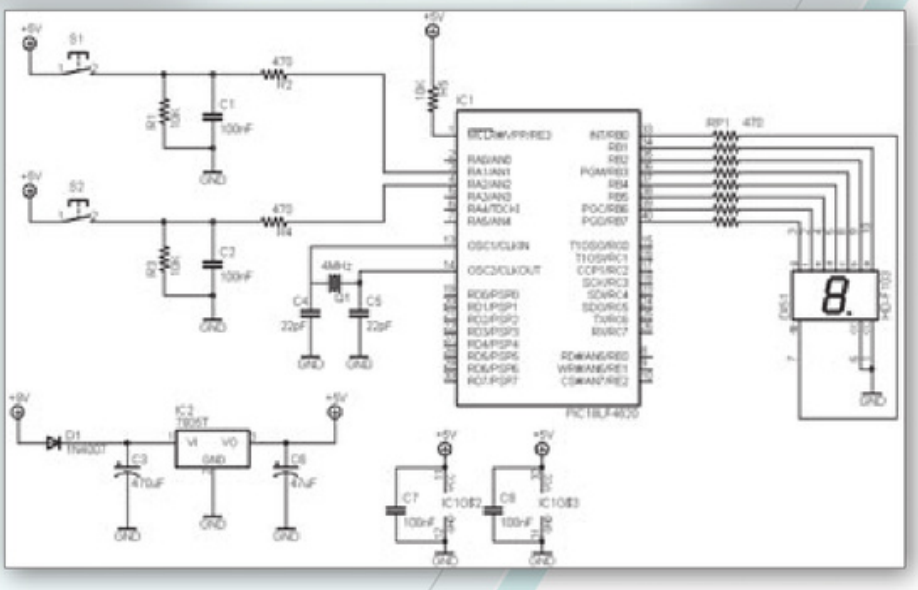
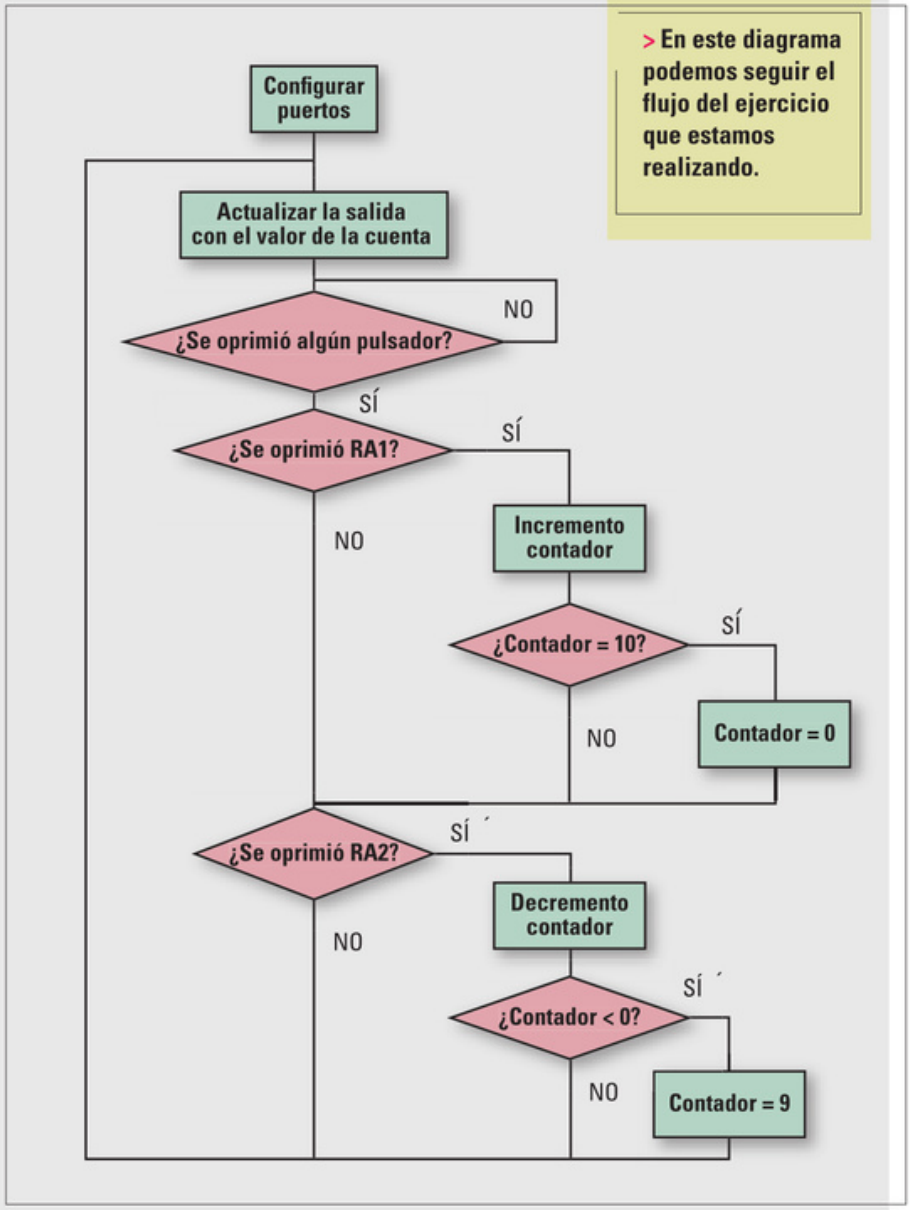


Figura 4. Este es el esquemático que debemos tomar como referencia para el desarrollo de la práctica.

> **En este diagrama podemos seguir el flujo del ejercicio que estamos realizando.**



Descripción del código

En la **Figura 4** se muestra el diagrama de flujo del ejercicio, a partir de cuyo código nos guiaremos para resolver el problema.

El código comienza configurando los puertos del PIC; las entradas de los pulsadores serán **RA1** y **RA2**, y la salida del programa será el puerto **B**. Luego, entramos en un bucle infinito, donde actualizamos el valor de los latchs del puerto B con el valor actual de la cuenta en formato de **7 segmentos**.

A continuación, nos quedamos esperando el cambio en los pulsadores. Basta con oprimir uno de ellos para que la condición sea verdadera, y el programa pase a interrogar cuál es el que se presionó. Una vez identificado, se realiza la función correspondiente de incrementar o disminuir el estado de la cuenta. Si la cuenta se excede de su valor máximo, inicializamos el contador; pero si es menor que 0 (cero), retornamos con el valor **9**.

Cuando pasamos al código del

```
/* Includes */
#include "p18f4620.h" // Incluimos archivo con los registros del PIC

/* Declaraciones */
// declaramos datos constante en memoria de programa comenzando en la
// direccion 0x180
#pragma romdata Tabla_Display = 0x180
const rom unsigned char Tabla[10] = {          0b00111111, //0
                                                0b00000110, //1
                                                0b01011011, //2
                                                0b01001111, //3
                                                0b01100110, //4
                                                0b01101101, //5
                                                0b01111101, //6
                                                0b00000111, //7
                                                0b01111111, //8
                                                0b01100111};//9


/* Programa principal */
#pragma code  // zona de código
void main (void)
{
char Indice = 0; // variable de 8 bits

        TRISB = 0b00000000; // PORTB como salida

ADCON1=0x0F;    //Configuramos todos los pines como digitales

        TRISA = 0b00000110;                             // RA1,RA2 entradas

        while (1)
        {// Entramos en un bucle infinito

                // usamos la tabla para obtener las salidas en los LEDs y
                // la recorremos con la variable Indice
                LATB = Tabla[Indice];

        while (PORTAbits.RA1 != 1 && PORTAbits.RA2 != 1);
                // No hacemos nada hasta que se produce un cambio en
                // cualquiera de los pulsadores

                if (PORTAbits.RA1 == 1) // Se pulso RA1?
                {
                        Indice++;  // incrementamos el indice
                if (Indice == 10)
                Indice = 0; // inicializamos Indice
                }

                if (PORTAbits.RA2 == 1)      // Se pulso RA2?
                {
                Indice--; // Decrementamos el indice
                        if (Indice < 0)
                                Indice = 9;// para la cuenta regresiva
                }
  }
}
```

ejercicio, creamos una tabla con la conversión de la cuenta a un display de 7 segmentos, que declaramos en la memoria de programa. Una variable **Índice** recorre las posiciones de la tabla para devolvernos su valor en código de 7 segmentos; la utilizaremos como el contador de nuestro programa.

Para detectar el cambio de los pulsadores, realizamos un **AND lógico** entre los estados **RA1** y **RA2** dentro de la expresión del **while**. Entonces, basta con oprimir uno de los pulsadores para que la expresión entre paréntesis sea falsa, y el programa salte a la próxima instrucción.



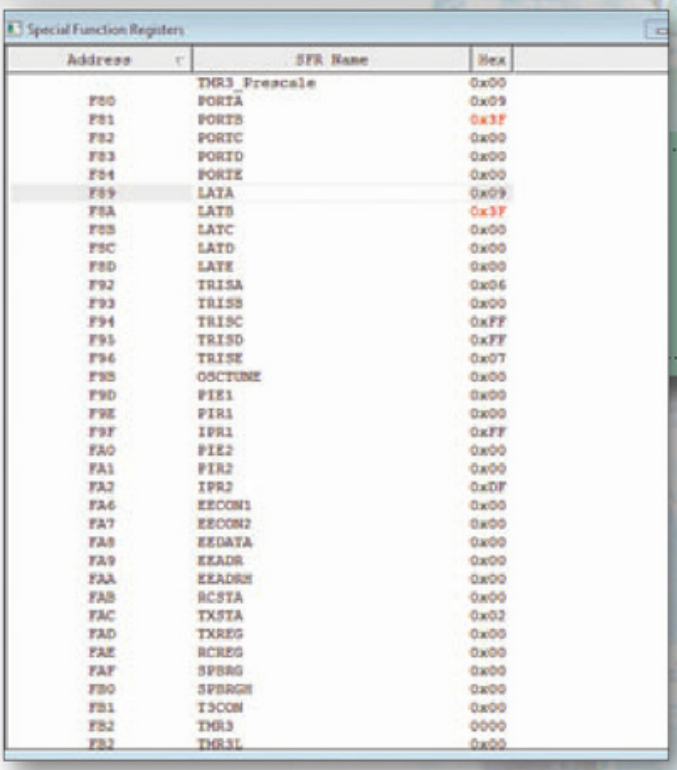
Código resultante

Vamos a mostrar el código resultante del ejercicio, que, luego, podemos simular en **MPLAB SIM** para verificar su funcionamiento. Es importante comentar todas las líneas de código para no olvidar cuál es la función de cada sentencia. Observemos también la posición de memoria que ocupa la tabla: la colocamos bien alejada del código de programa para que no se pisen. Siempre debemos controlar estas posiciones de memoria para no tener problemas.

Las entradas de los pulsadores serán RA1 y RA2, y la salida del programa será el puerto B.

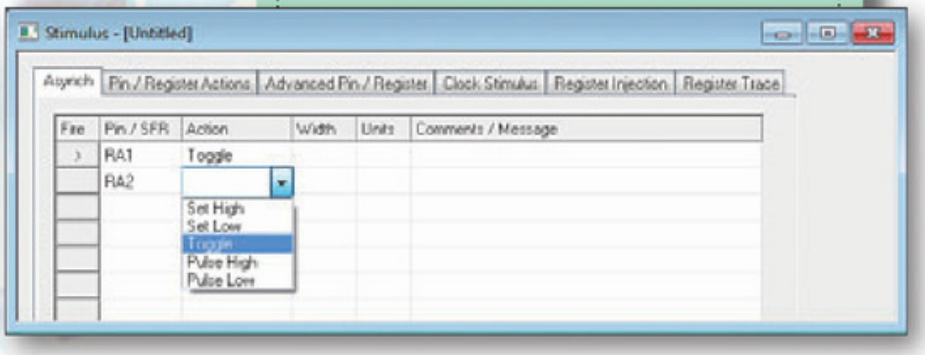
Con este primer ejercicio, aprendimos cuál es la estructura de un programa en lenguaje de alto nivel para los microcontroladores, en el que las sentencias son fáciles de seguir. El próximo paso es compilar el proyecto y realizar la simulación.





> En MPLAB, las modificaciones de los registros se visualizan en la ventana de SFR. Para ver los registros más importantes, configuramos la ventana Watch.

> Desde la ventana Stimulus podemos simular cambios asincrónicos en los pines del PIC. Esto es muy útil cuando simulamos entradas digitales. En la imagen vemos cómo configurar la entrada de los pulsadores en el programa.



Simulación de programas en MPLAB SIM

Vamos a utilizar el **MPLAB SIM** para simular nuestro programa. Pero antes debemos compilarlo, por lo que debemos crear un nuevo proyecto en **MPLAB**.

Comenzamos seleccionando **Project/Project Wizard** en el entorno de **MPLAB**. Luego, elegimos el PIC con el que vamos a trabajar; en este caso, es el **PIC18F4620**.

En el siguiente paso, debemos escoger el lenguaje de programación, para lo cual nos dirigimos a **Microchip C18 Toolsuite**.

Indicamos el nombre del nuevo proyecto y la ubicación que tendrá en la PC. Cuando el programa nos pida agregar un archivo, colocamos el código fuente que acabamos de crear y, por último, presionamos **Finalizar**.

Solo resta un paso más, que es poner el archivo **linker** necesario para que el **C18** pueda compilar. Para hacerlo, vamos a la ventana **Project** y, en la carpeta **Linker Script**, hacemos clic con el botón secundario del mouse y elegimos **Add Files**. Buscamos el archivo **18f4620i.lkr** en la carpeta **LKR**, dentro del directorio **MCC18**.

Una vez hecho todo esto, compilamos el proyecto, seleccionando **Build All**.

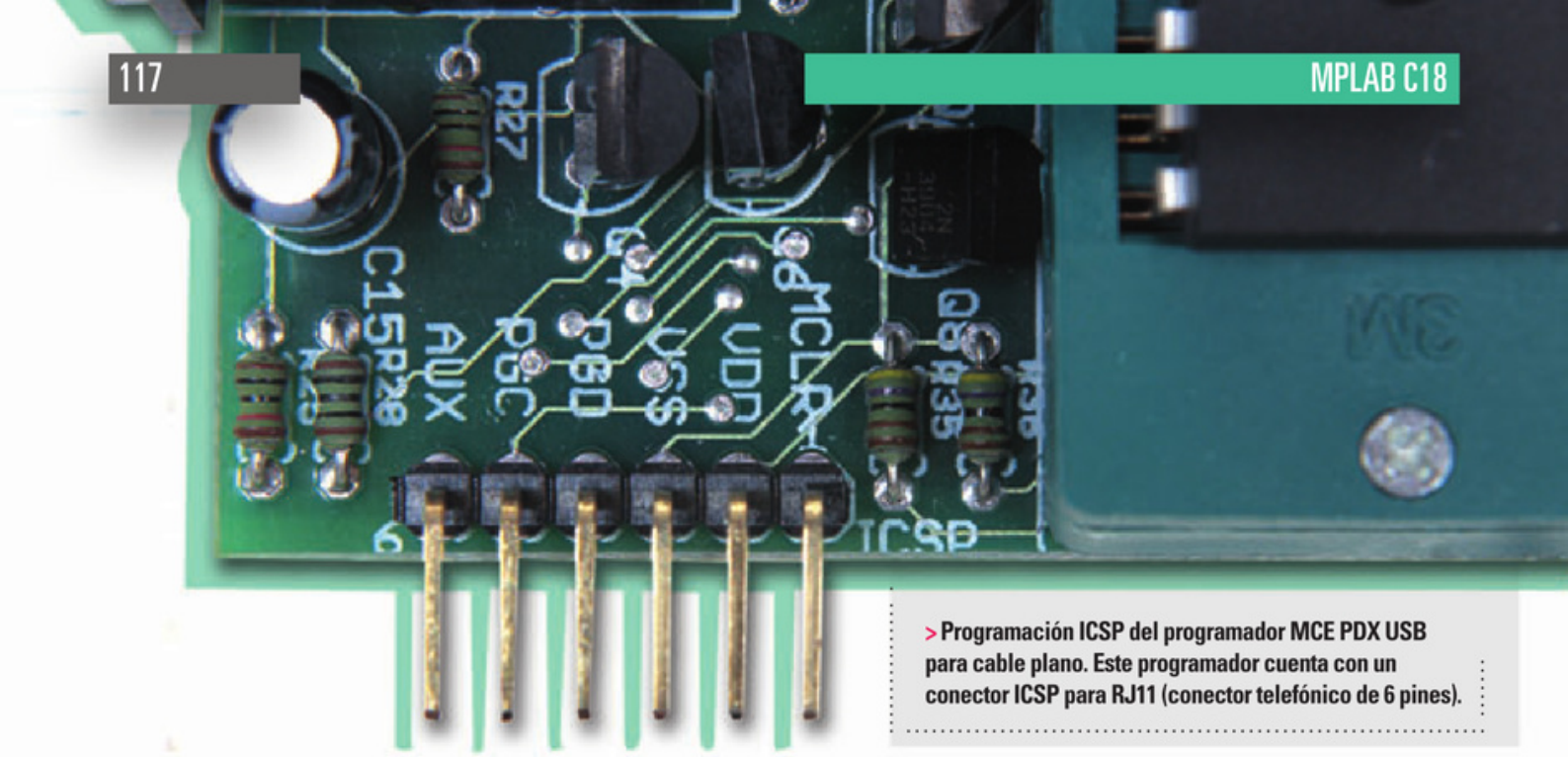
Para comenzar con la simulación, vamos a **Debugger/Select tool** y elegimos **MPLAB SIM**. Veremos que aparecen nuevos iconos en la barra de herramientas, que sirven para controlar el proceso de simulación.

Para evitar pasar por todos los archivos de arranque, podemos colocar un **breakpoint** en la primera instrucción de la función **main** y correr el programa con la tecla **run** hasta detenerse allí. Es posible simular los pulsadores en la ventana **Debugger/Stimulus**, seleccionando **New Workbook**. En la solapa **Asynch** buscamos la columna **Pin/SFR** y elegimos, por ejemplo, el pin **RA1**, mientras que en **[Action]** elegimos la acción que realizará el pulsador, como poner un nivel alto. Cada vez que oprimamos **[Fire]**, durante la simulación, se llevará a cabo la acción seleccionada.

Programador MCE PDX USB

En el mercado es posible encontrar muchas herramientas de desarrollo para los microcontroladores PIC. Los programadores comerciales más populares de Microchip son actualmente **MPLAB ICD2 y PicKit 2**, que no solo graban el PIC, sino que también permiten realizar la depuración de código en circuito (*debugger in-circuit*).

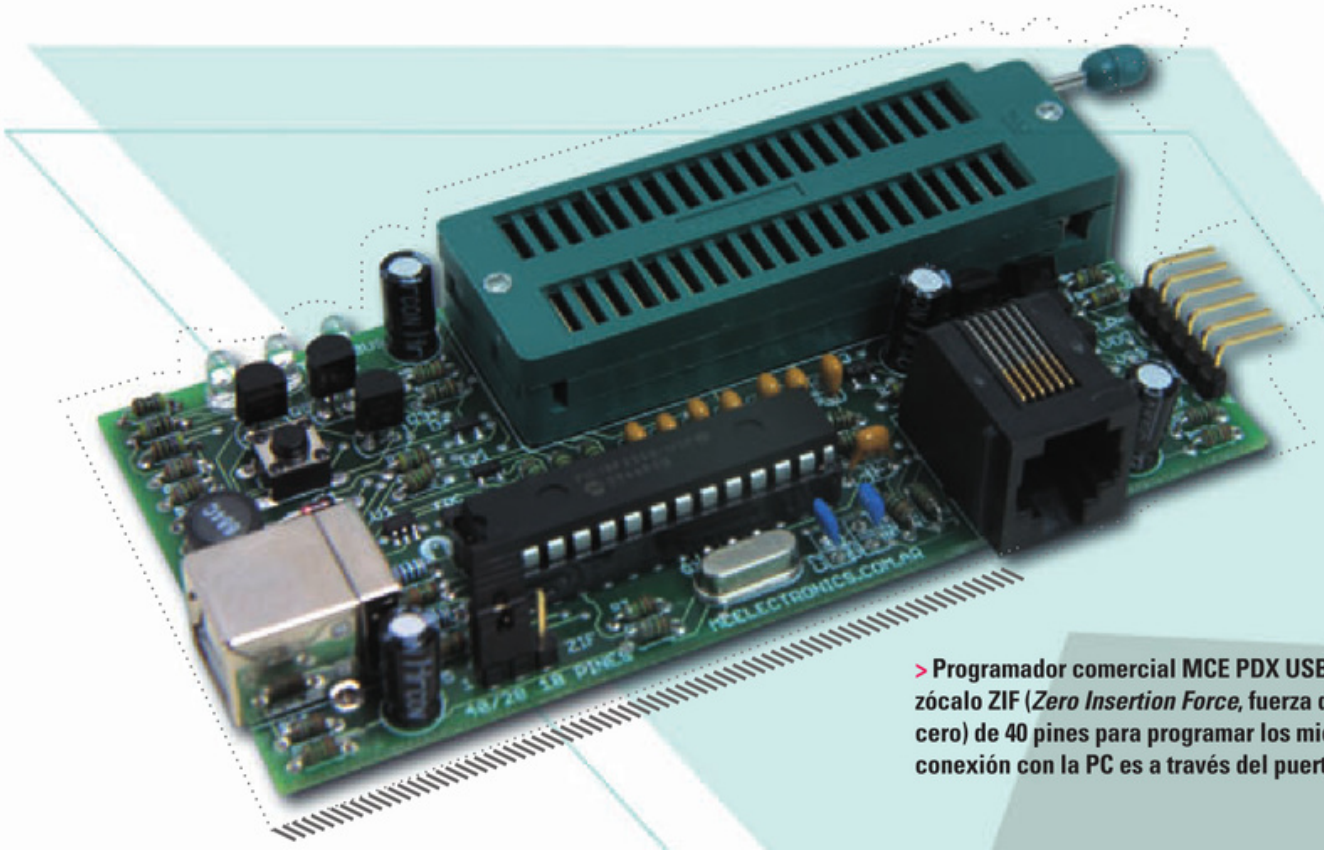
Una de las ventajas de poder depurar código en el circuito es detectar esos errores que son ajenos al programa. Si bien estos programadores resultan muy útiles, su costo es elevado, pero



> Programación ICSP del programador MCE PDX USB para cable plano. Este programador cuenta con un conector ICSP para RJ11 (conector telefónico de 6 pines).

en el mercado podemos encontrar herramientas con las mismas funciones a un costo mucho menor. Este es el caso del **MCE PDX USB**, de la empresa **MC Electronics**. Esta versión comercial es una buena alternativa a las de Microchip, porque además de poseer la funcionalidad de depurar código en circuito, también agrega funciones alternativas para detectar errores. Puede usarse, además, como un simple analizador de estados lógicos de tres canales. Soporta una gran cantidad de microcontroladores PIC y es totalmente compatible con el entorno de programación MPLAB. El programador MCE PDX se conecta a la PC a través del puerto USB y no necesita alimentación externa, ya que la toma directamente del puerto. También posee un regulador de tensión interno de 2,5 Volts a 5 Volts, lo que permite grabar microcontroladores que se alimenten a 3,3 Volts, muy comunes en estos días. Entonces, una herramienta como esta nos será de gran ayuda en nuestro taller si queremos profundizar en el conocimiento de los microcontroladores PIC.

El programador MCE PDX se conecta a la PC a través del puerto USB y no necesita alimentación externa.

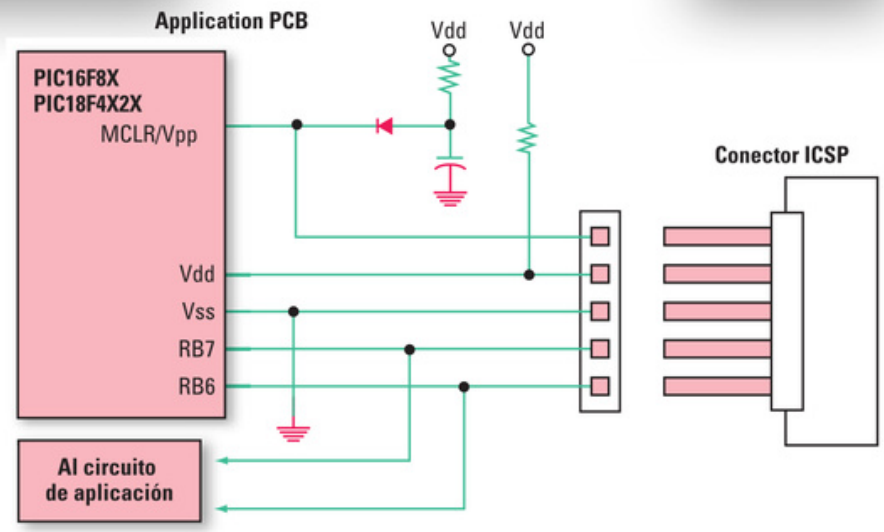


> Programador comercial MCE PDX USB. Posee un zócalo ZIF (*Zero Insertion Force*, fuerza de inserción cero) de 40 pines para programar los micros. La conexión con la PC es a través del puerto USB.

Depuración en circuito

Muchos de los microcontroladores PIC incorporan **ICSP** (*In Circuit Serial Programming*, programación serie en circuito) o **LVP** (*Low Voltage Programming*, programación a bajo voltaje). Estas características permiten programar el PIC en el circuito, sin necesidad de retirarlo. Solo hay que agregar en el **PCB** un conector con las cinco líneas para la programación del PIC.

En la programación **ICSP**, utilizamos los pines **RB6** y **RB7** como reloj y datos, y el pin **MCLR** para activar el modo programación, por medio de una tensión de entre 12 y 13 Volts. Pero esta comunicación serie que tenemos con el PIC no solo nos permite programarlo, sino que también nos da la posibilidad de efectuar una depuración del código en circuito (*In-Circuit Debugging*).

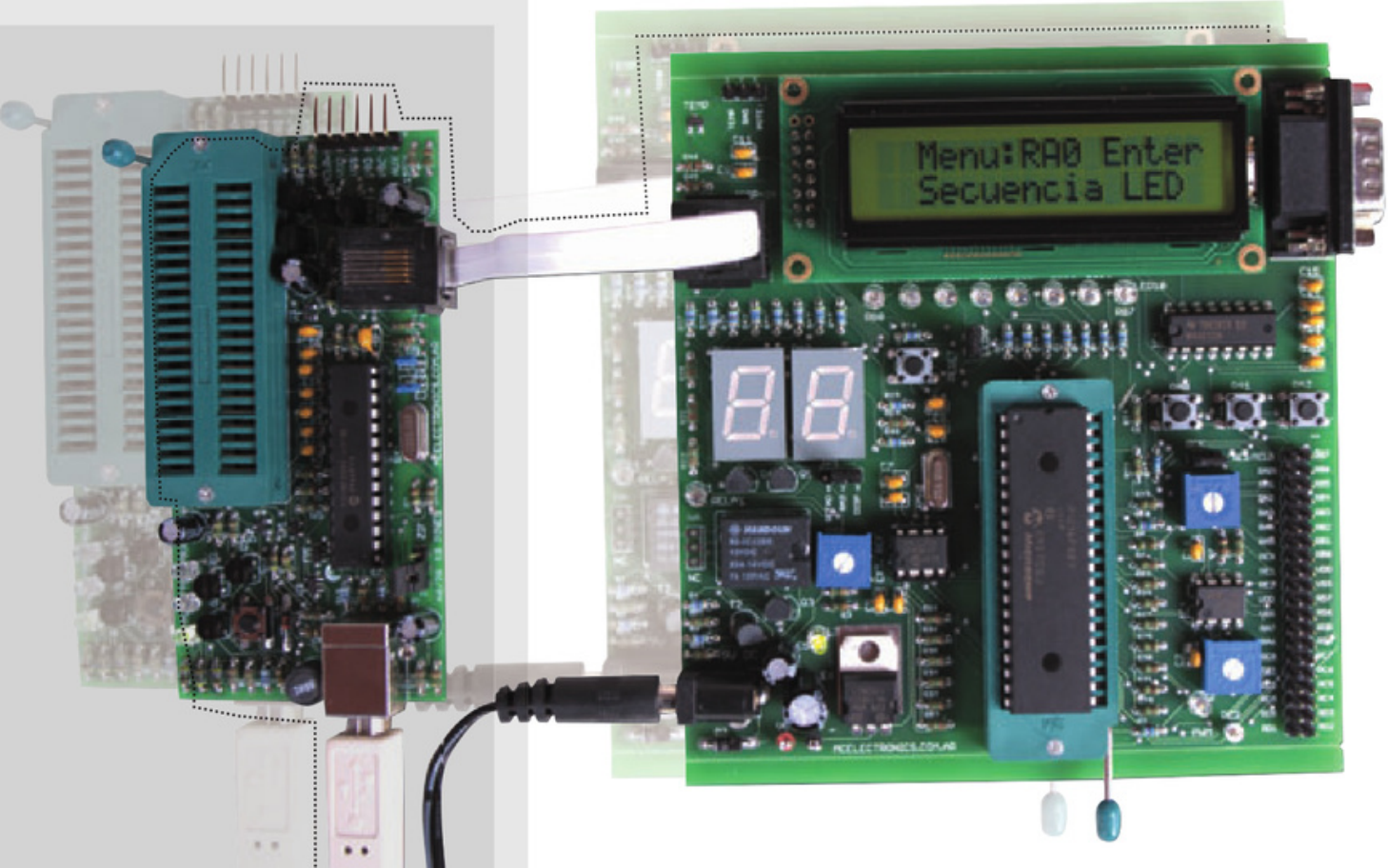


> Esta es la conexión circuital que debemos agregar a nuestras placas para realizar la programación y la depuración en circuito.

ICSP y LVP son características que permiten programar el PIC en el circuito, sin necesidad de retirarlo.

La herramienta para hacerlo es el programador **MCE PDX USB**, que ya tiene disponibles conectores externos para la programación y depuración de código en circuito. Lo importante es que este programador es compatible con la interfaz del **MPLAB**, y las herramientas de depuración son las mismas que analizamos con el simulador **MPLAB SIM**.

> Vemos la conexión ICSP del programador con la placa experimental. En este caso, se utilizó una placa entrenadora comercial para simular los ensayos.



Depuración in-circuit del programa en C

Para realizar la depuración del código, debemos contar con un programador que nos permita hacer este proceso en circuito, como el **MCE PDX**, y conectarlo a nuestra placa experimental a través de las cinco líneas de programación.

En este caso, utilizamos una placa comercial que posee pulsadores y display de 7 segmentos para efectuar la depuración del código.

Una vez que tenemos conectados el programador y la placa experimental, nos resta enchufar el cable USB del programador a la PC.

Ejecutamos el **MPLAB** y abrimos nuestro primer proyecto en C, el contador ascendente/descendente. Luego, elegimos la herramienta de depuración en el **MPLAB**, para lo cual nos dirigimos al menú **Debugger/Select Tool** y elegimos **7 PicKit 2**.

En la ventana **Output** del **MPLAB**, debemos visualizar el estado de la conexión del programador y verificar que no se produjeron errores.

Observaremos que aparecen las mismas herramientas para la depuración de código que teníamos con **MPLAB SIM** y que, además, se agregaron otras (propias del programador).

Antes de comenzar, debemos compilar el código haciendo clic en **Project/Build All**. Luego, programamos el micro desde **Debugger/Program**, para que el micro se grabe con el código y nos permita efectuar la depuración.

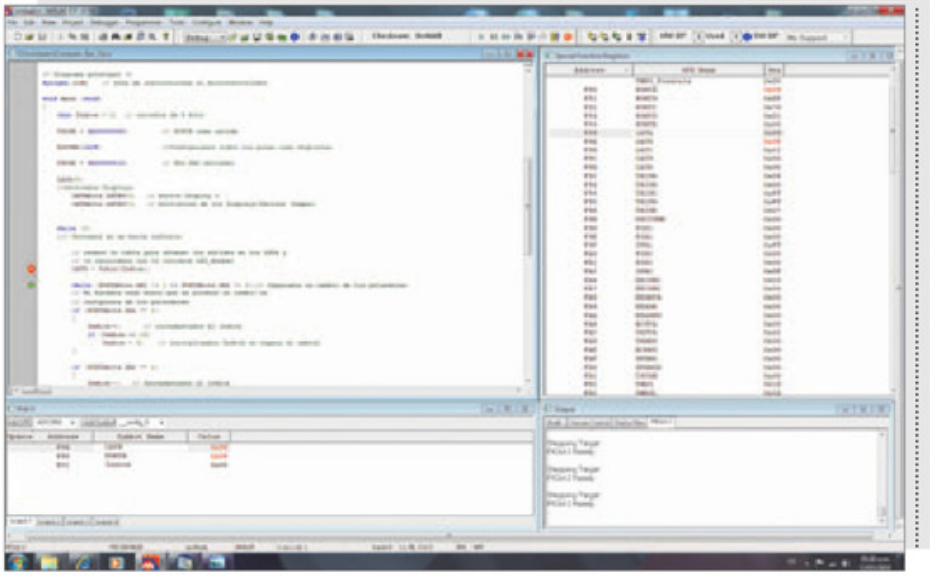
Ahora podemos comenzar con la simulación del código. Es importante destacar que este microcontrolador nos permite poner hasta **tres breakpoints** o puntos de detención de programa.

Si corremos la depuración con **run**, veremos cómo se ejecuta el programa en la placa y podremos detenerlo en cualquier instante, oprimiendo el icono **halt**. También tenemos la opción de ejecutar el código línea por línea y ver cómo se modifican los registros de funciones especiales (SFR). La ventana para visualizar los SFR se selecciona desde **View/Special Functions Registers**.

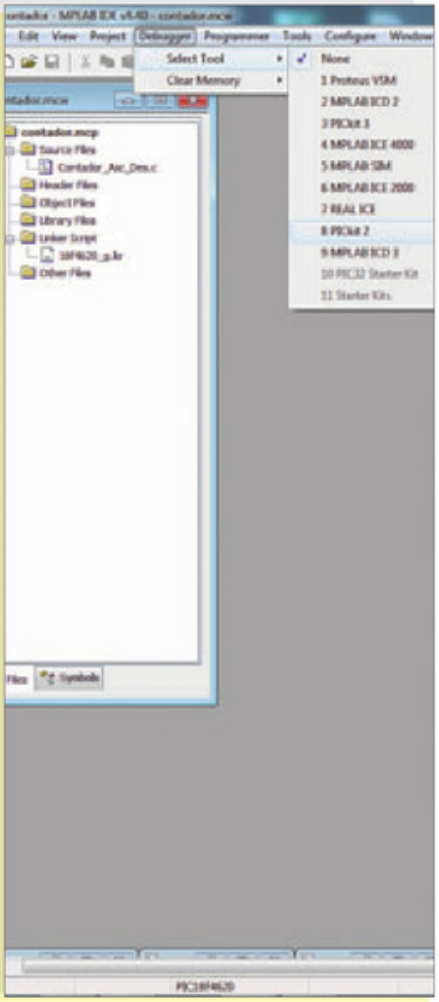
En el momento de llegar al **while**, que espera el cambio de los estados de los pulsadores, podemos oprimir cualquiera de ellos en la placa y ver de qué manera se comporta el código ante ese estímulo que le imponemos.

Veremos que esta forma de buscar errores en el código es más efectiva, porque nos permite observar el comportamiento del PIC con los periféricos externos, y encontrar si algún error es producto del código o del hardware.

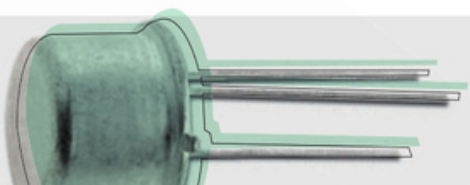
Cuando estemos en el modo depuración, notaremos que los pines **RB6** y **RB7** se utilizan para la comunicación, por lo que no podremos simularlos como entradas o salidas digitales. ■



> En la depuración en circuito, podemos colocar hasta tres breakpoints para este microcontrolador y recorrer el código línea por línea, a la vez que los registros de memoria se van actualizando.



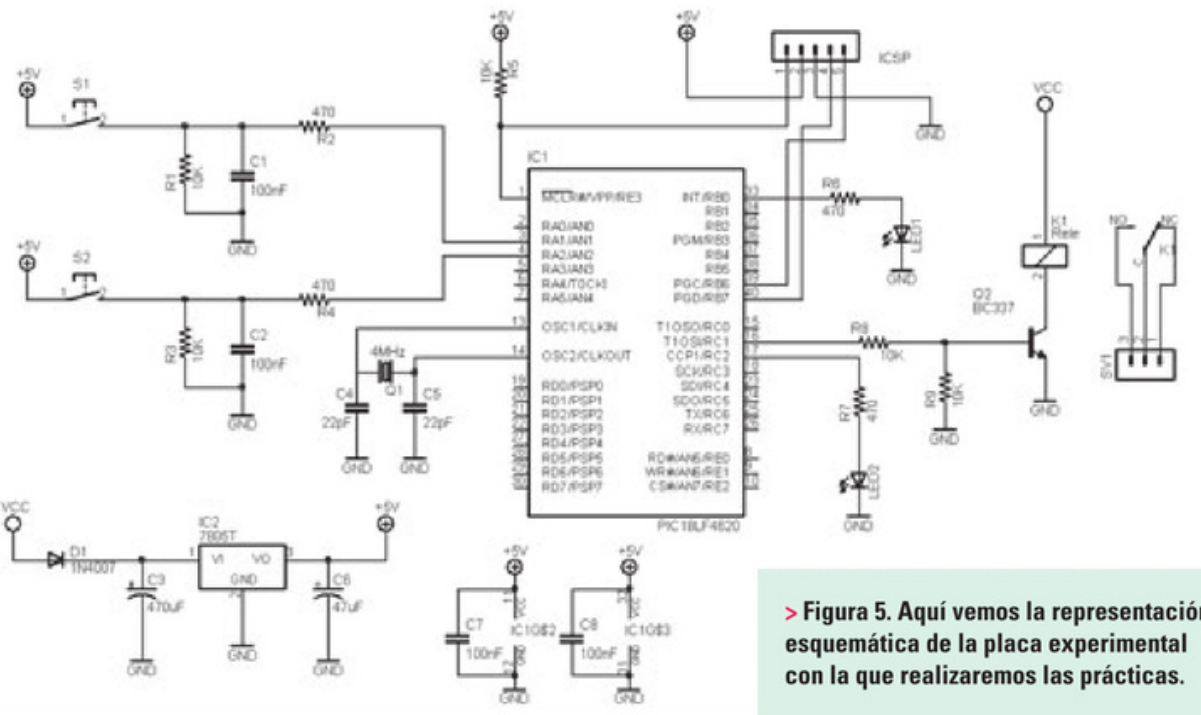
> Podemos ver aquí cómo seleccionar la herramienta de depuración para el programador MCE PDX USB.



PLACA EXPERIMENTAL PARA PIC18LF4620

En las siguientes páginas veremos cómo armar una placa experimental para el PIC18LF4620 y, así, realizar las prácticas correspondientes. Veamos de qué se trata este tema.

En la **Figura 5,** vemos el esquemático de la placa que armaremos. Podemos notar que el centro del esquema circuital es el microcontrolador **PIC18LF4620**, al que se le acoplan distintos dispositivos, como pulsadores, diodos LED y la interfaz para el control de un relay. A un costado, distinguimos el circuito de alimentación, que utiliza un regulador de 5 V, y el LM7805 con sus capacitores de filtrado.
A esta placa, se le agrega un conector de cinco pines para realizar una programación **ICSP** (*In-Circuit Serial Programming*, o programación serie en circuito), ya que será utilizada para experimentar con diferentes códigos. Este conector nos resultará de gran utilidad para programar el PIC sin necesidad de retirarlo de la placa.
Los dispositivos externos que agreguemos en la placa serán usados para entrenarnos en la programación del PIC, en tanto que los pulsadores servirán como entradas de datos. En el esquemático, observamos también que, cuando el pulsador se encuentra abierto, el circuito impone un cero lógico, y cuando lo oprimimos, pasa al estado lógico alto. Colocamos diodos LED para indicar el estado de los pines **RB0** y **RC2** del PIC. Los LEDs son muy útiles para esta función, ya que se encenderán cuando coloquemos ese pin en estado alto. El del pin **RC2** será utilizado en las prácticas de PWM (modulación de ancho de pulso), donde podremos variar su intensidad por medio de la técnica de modulación. ■



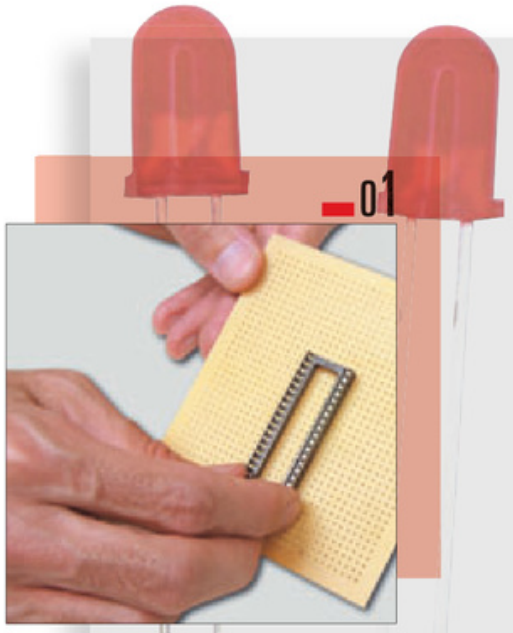
> Figura 5. Aquí vemos la representación esquemática de la placa experimental con la que realizaremos las prácticas.

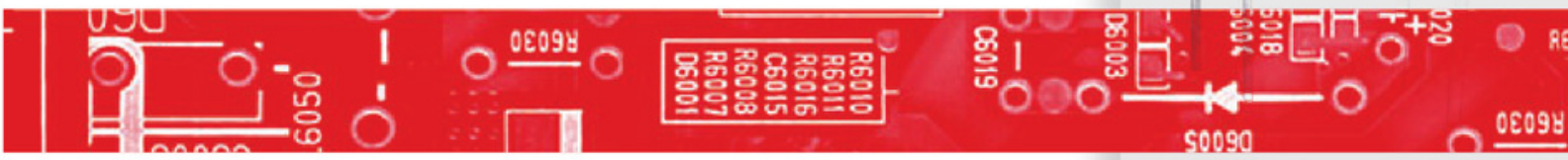
PASO A PASO

CONSIDERACIONES DE ARMADO

Veremos algunos aspectos que debemos tener en cuenta en el momento de armar una placa experimental.

Al diseñar y armar una placa experimental, debemos tener presentes algunos consejos que daremos en estas páginas para asegurar un armado exitoso. Estas recomendaciones también nos servirán en el armado de futuras placas que cuenten con algún microcontrolador. ■

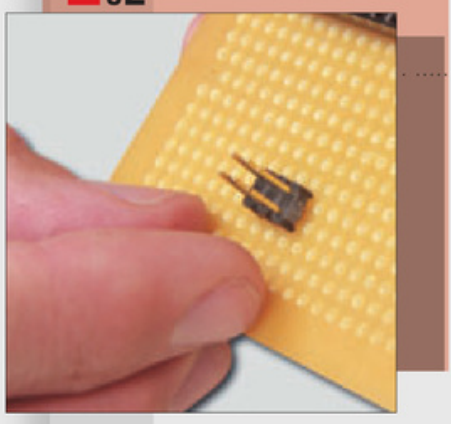


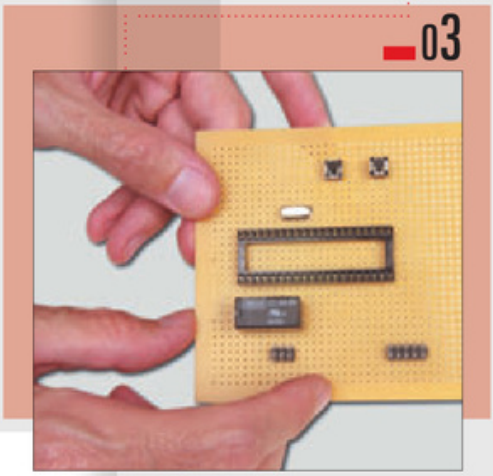


01. Una vez que tenemos todos los materiales necesarios, comenzamos a armar la placa. Procedemos de la misma forma que con placas universales. Empezamos colocando el zócalo DIP de 40 pines para, luego, poder orientarnos en el momento de instalar los demás componentes.

02. Luego de insertar el zócalo, ponemos los demás componentes, guiándonos con el esquemático del circuito. Es importante colocar los componentes más chicos, como conectores y pulsadores, para que el armado no resulte incómodo. Si empezamos con los más grandes, será difícil manipular la placa.

03. En el momento de elegir en dónde colocar el cristal resonador, recomendamos ubicarlo lo más cerca posible del PIC, ya que el circuito oscilador es la parte de la placa que trabaja a mayor frecuencia. Si mantenemos las pistas del circuito oscilador lo más chicas posible, evitaremos muchos problemas, considerando que esta zona es muy sensible a interferencias externas.

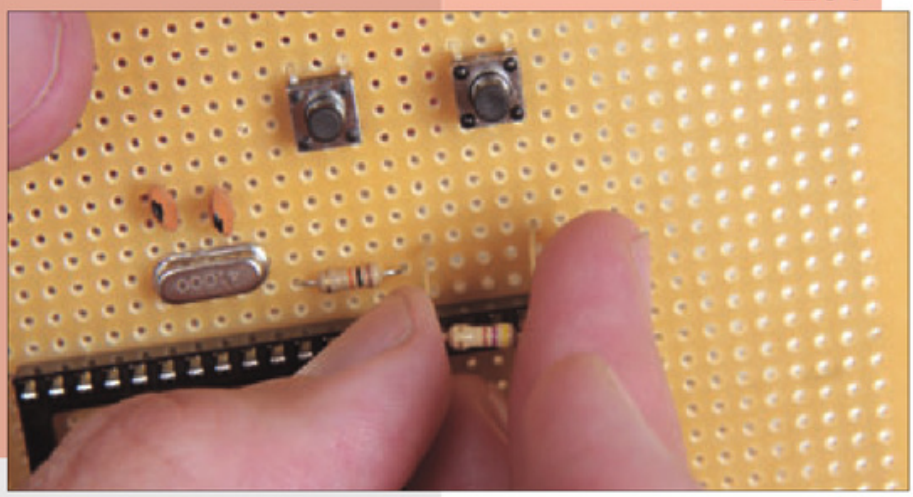








04



NPO (Negative-Positive zero), por ser más estables a variaciones de temperatura, y muy útiles en circuitos oscilantes y filtros. Se reconocen porque poseen una marca negra en la parte superior.

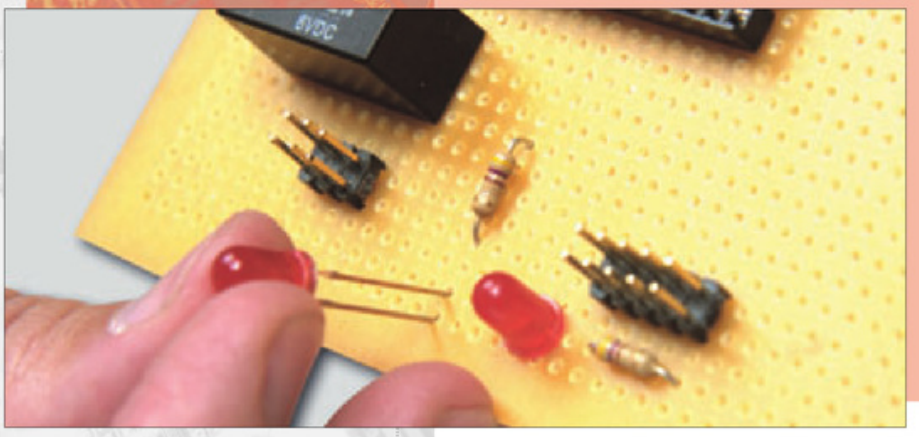
04. Colocamos las resistencias y los capacitores. Para el oscilador, además, recomendamos utilizar capacitores de 22 pF del tipo

05. Debemos tener especial cuidado al colocar los LEDs indicadores, dado que tienen polaridad y, si los ponemos al revés, no encenderán. Recordemos que el borde achatado del cuerpo marca el cátodo del dispositivo, que debe ser conectado, en este caso, a la masa del circuito.

Al colocar los LEDs debemos tener mucho cuidado, ya que poseen polaridad.

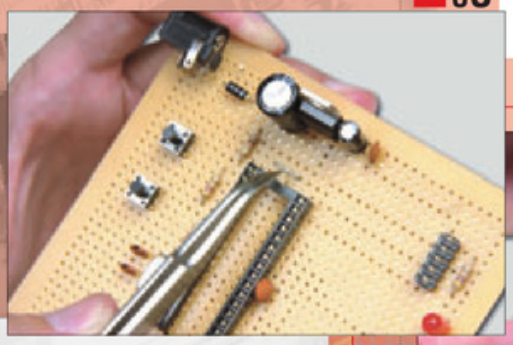


05

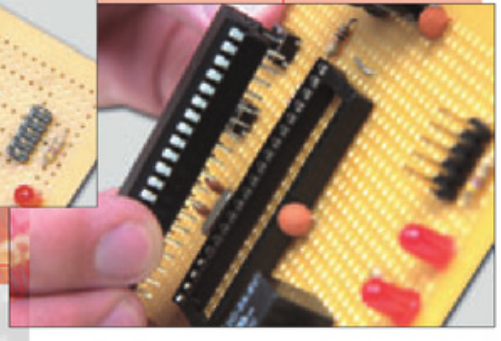


06. En el momento de comenzar a armar el circuito de alimentación, hay que colocar primero el regulador LM7805. Lo posicionamos en un extremo de la plaqueta, para poder agregarle un disipador de calor más tarde. Ubicamos los capacitores electrolíticos a los lados del regulador y verificamos que su polaridad sea correcta.

06

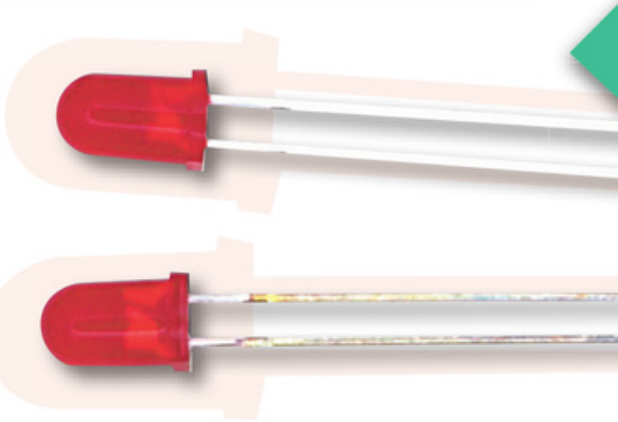


07



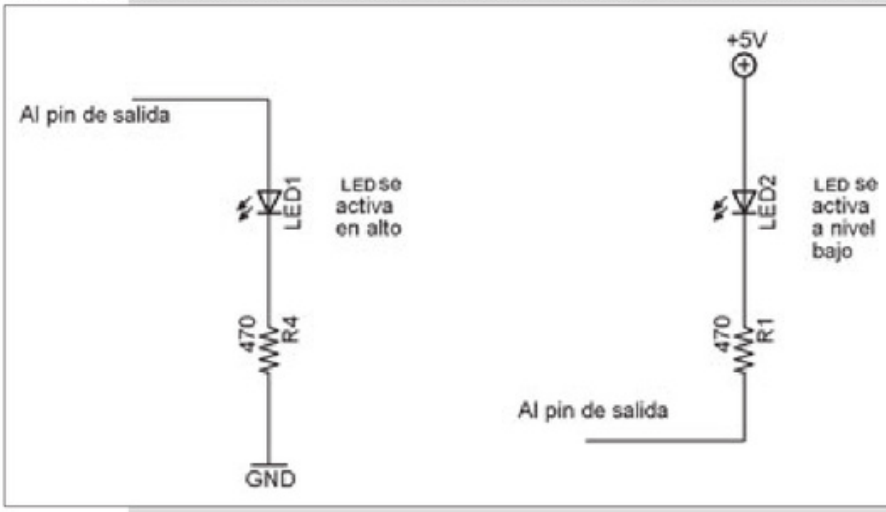
07. Cuando terminamos de soldar y probar la placa, podemos colocar el PIC en el zócalo DIP de 40 pines. También podemos utilizar un zócalo adicional para evitar romper alguna patita cuando tengamos que maniobrar con él, por ejemplo, cuando lo sacamos del zócalo para grabarlo en el programador.

PERIFÉRICOS EXTERNOS



Los puertos del microcontrolador pueden enviar o recibir datos. En estas páginas veremos la manera de conectar los periféricos externos más usuales al PIC.

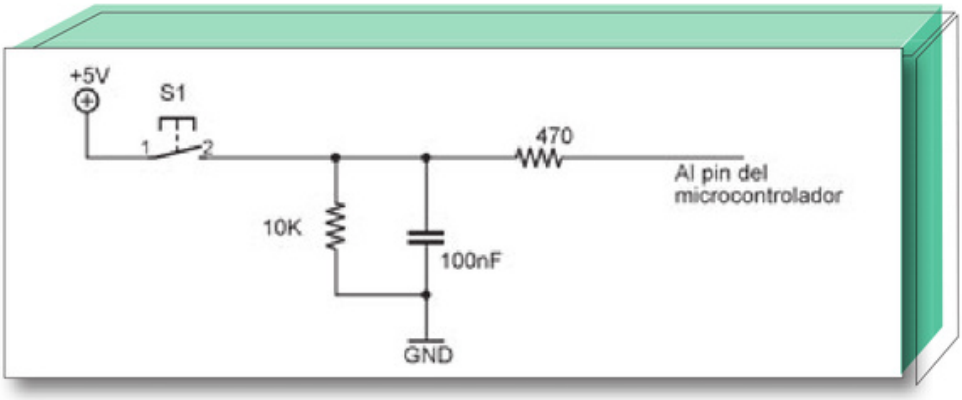
Los LEDs son los indicadores luminosos más simples que podemos conectar al PIC. En la **Figura 6** podemos ver dos modos de conectar el LED al pin del microcontrolador: una de las configuraciones excita al LED con una salida en alto en el pin, y la otra, con un nivel bajo. Para dimensionar la resistencia utilizamos la fórmula: **R = (Vf – Vled) / Iled**, donde **Vf** puede ser la tensión de salida del pin o la tensión de la fuente; **Vled** es la tensión del LED que el fabricante nos da como dato; e **Iled** es la corriente del LED, que puede estar entre **5 mA y 20 mA**, según la intensidad que deseamos tener (colocando una resistencia de **470** Ω, obtenemos un indicador lumínico aceptable).



> **Figura 6. Distintos modos de conectar un LED a un microcontrolador. Una de las configuraciones activa el LED en un nivel alto en el pin, y la otra lo hace a nivel bajo.**

Pulsadores

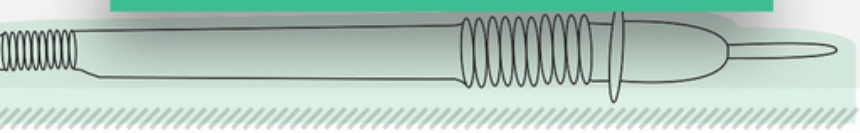
La manera más sencilla de enviar información hacia nuestro microcontrolador es a través de los pulsadores. En la **Figura 7** vemos una conexión típica de un pulsador al cual se le coloca un capacitor para eliminar el efecto rebote que tiene todo contacto mecánico. Entonces, podemos enviar un **1** o un **0** al cerrar o abrir el interruptor o el pulsador.

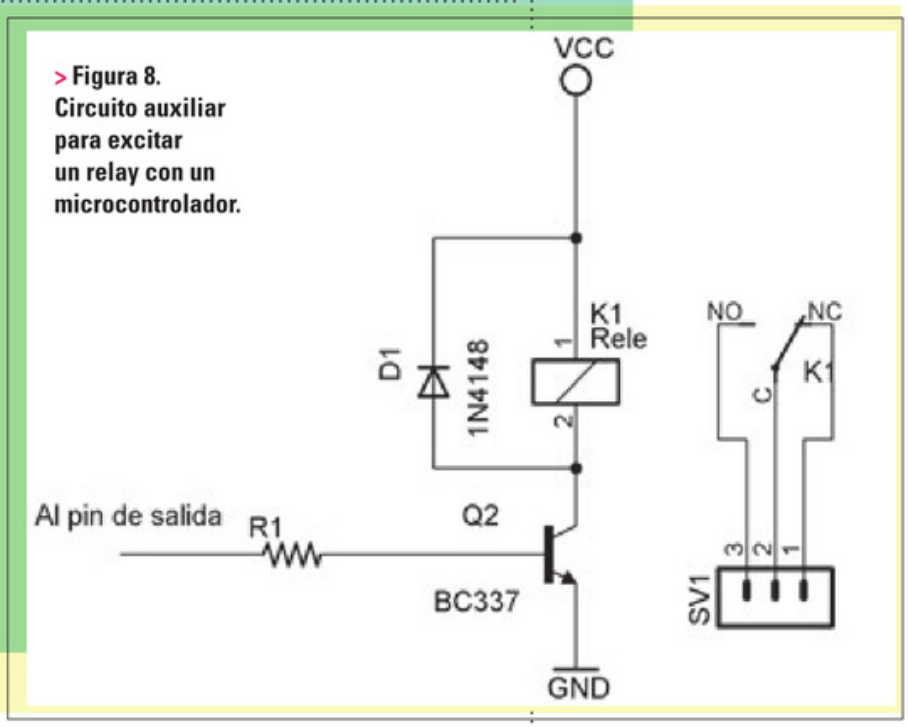


> **Figura 7. Circuito auxiliar para conectar un pulsador al microcontrolador.**

Relay

El relay (o relé) es un dispositivo electromecánico que actúa como un interruptor, controlado por una bobina que acciona un juego de uno o varios contactos. Estos permiten abrir o cerrar circuitos electrónicos independientes de nuestro microcontrolador. Como los contactos de salida del relay pueden conectarse a circuitos que son independientes del circuito de excitación, podemos controlar circuitos de salida con mayor potencia que de entrada.





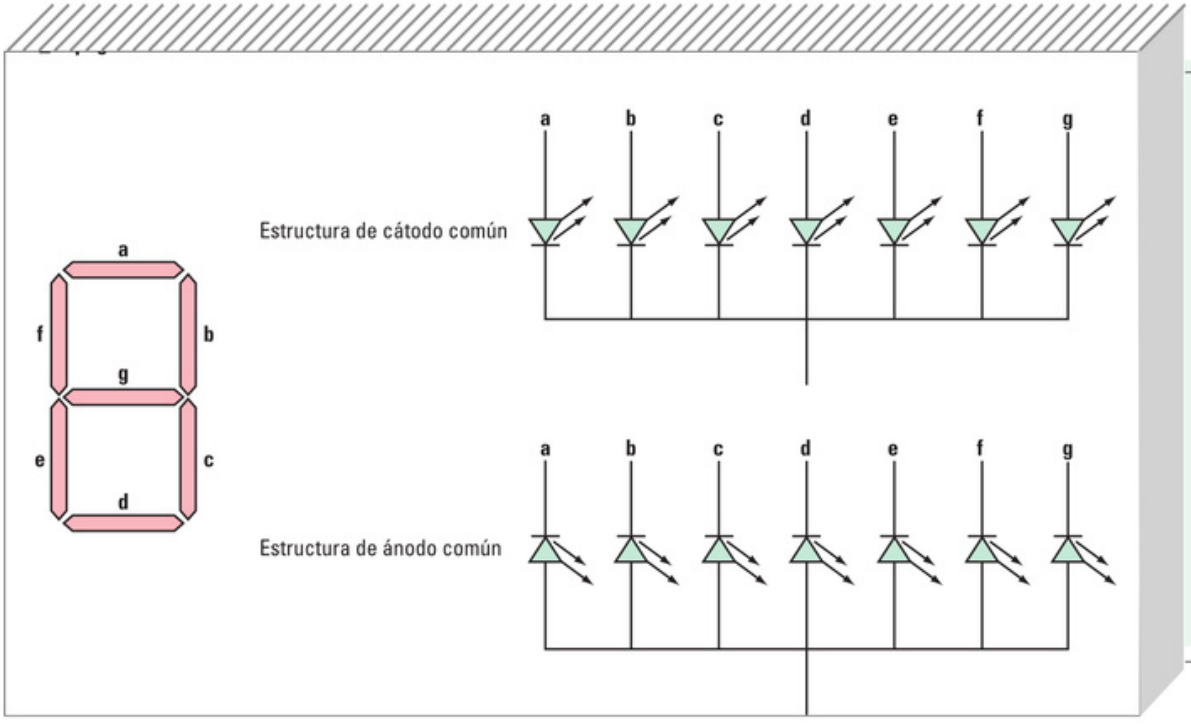
> **Figura 8. Circuito auxiliar para excitar un relay con un microcontrolador.**

Con los relays, podemos controlar circuitos de salida con mayor potencia que de entrada.

Los contactos de un relay pueden ser normalmente abiertos (**NA**) o normalmente cerrados (**NC**). Los NA se cierran cuando se excita la bobina del relay, en tanto que los NC se abren en ese caso. En la **Figura 8** vemos cómo conectar el relay a nuestro PIC. Para hacerlo, utilizamos un transistor **BC337**, porque puede drenar mayor corriente entre su colector y emisor que las salidas del microcontrolador. También hay que colocar un diodo de protección –llamado de **rueda libre**– entre la bobina del relay, para evitar que los transitorios de corriente destruyan el transistor.

Display de 7 segmentos

Los displays de 7 segmentos serán de utilidad en circuitos donde haya que mostrar números. En el mercado podemos conseguir displays de ánodo común y de cátodo común, según la conexión de los LEDs internos. Dependiendo de la configuración de estos LEDs internos, las barras se encenderán con un **1** (cátodo común) o con un **0** (ánodo común).
Para colocar una hilera de varios displays –con el fin de representar números de más de un dígito–, podemos utilizar la técnica de **multiplexado**. Esta enciende los displays de manera secuencial a una velocidad tan rápida que el ojo humano no puede percibir su apagado; a su vez, nos permite ahorrar pines del PIC. Veamos ahora el código en C:



> **Los displays de 7 segmentos son simplemente barras de LEDs que encendemos de manera selectiva para formar un número. Existen los de cátodo común y los de ánodo común, según la disposición de los LEDs.**

```
#include <p18f4620.h> //este archivo denominado de
                                    //cabecera incluye las definiciones
                                    //de todos los SFRs del PIC18F4620
#include <delays.h> // este archivo incluye librerias de tiempo
#pragma config OSC = XT //definimos que usaremos un
                                    //oscilador a cristal estandar

#pragma config WDT = OFF //definimos que no se usará el
                                    //Watchdog Timer

#pragma config MCLRE = ON //definimos que el pin MCLR se
                                    //usará como RESET y no como
I/O

void main (void)            //definimos la función principal
{
        unsigned int contador = 0;
        ADCON1 = 7;  //configuramos el PORTA como digital
        CMCON = 7;  //desactivando las funciones analógicas
LATB = 0;                   //ponemos el LATB en 0
        TRISAbits.TRISA1 = 1; //configuramos RA1 como entrada
        TRISB = 0; // configuramos todo el PORTB como salida
        while(1)  // creamos un bucle que se repite siempre
        {
                if (PORTAbits.RA1==1)//Chequeamos si RA1=1
                        {                   //Si es así entonces...
                        contador = contador + 1; //incrementa el contador
                        if (contador > 9)// chequeamos si paso de 9
                                {
                                contador = 0; // si paso reseteamos el
contador
                                }
                        }
switch (contador)//leemos el valor del contador
                                {       // y lo decodificamos a 7 segmentos
                                case 0:
                                        PORTB = 0b00111111; //escribe 0
en PORTB
                                        Break ;
                                case 1:
                                        PORTB = 0b00000110; //escribe 1
en PORTB
                                        Break ;
                                case 2:
                                        PORTB = 0b01011011; //escribe 2
en PORTB
                                        Break ;
                                case 3:
                                        PORTB = 0b01001111; //escribe 3
en PORTB
                                        Break ;
                                case 4:
                                        PORTB = 0b01100110; //escribe 4
en PORTB
                                        Break ;
                                case 5:
                                        PORTB = 0b01101101; //escribe 5
en PORTB
                                        Break ;
                                case 6:
                                        PORTB = 0b01111100; //escribe 6
en PORTB
                                        Break ;
                                case 7:
                                        PORTB = 0b00000111; //escribe 7
en PORTB
                                        Break ;
                                case 8:
                                        PORTB = 0b01111111; //escribe 8
en PORTB
                                        Break ;
                                case 9:
                                        PORTB = 0b01100111; //escribe 9
en PORTB
                                        Break ;
                                }
                        Delay1KTCYx(200);           //espera un tiempo
        }
}
```

Puertos I/O

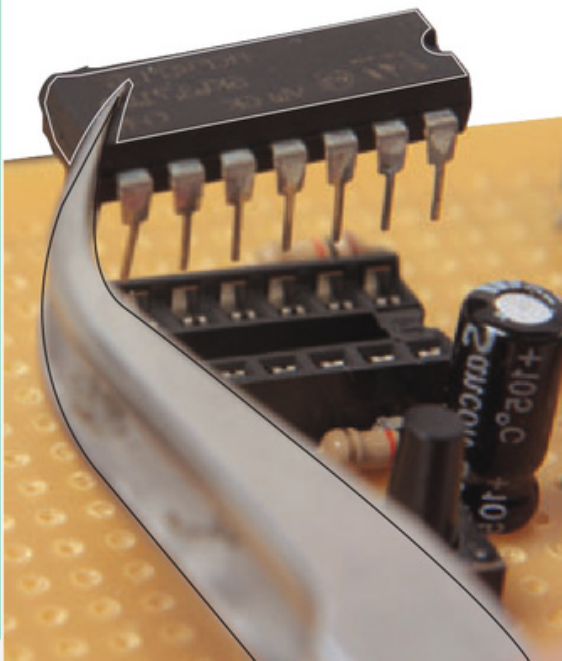
Los dispositivos expansores de puertos sirven para agregar uno o más puertos de **8 bits** de manera externa. Podemos recurrir a un dispositivo como este cuando los puertos de nuestro PIC son insuficientes.
Uno de los expansores más utilizados es el **integrado PCF8574**, que contiene un puerto de entrada/salida de 8 líneas. Se conecta al PIC a través del protocolo de comunicación I2C.

Programa de un contador con display de 7 segmentos

Veamos ahora un pequeño programa en **lenguaje C** en el que implementaremos una lectura de pulsador y el incremento de un contador de un dígito.

Detalles del programa

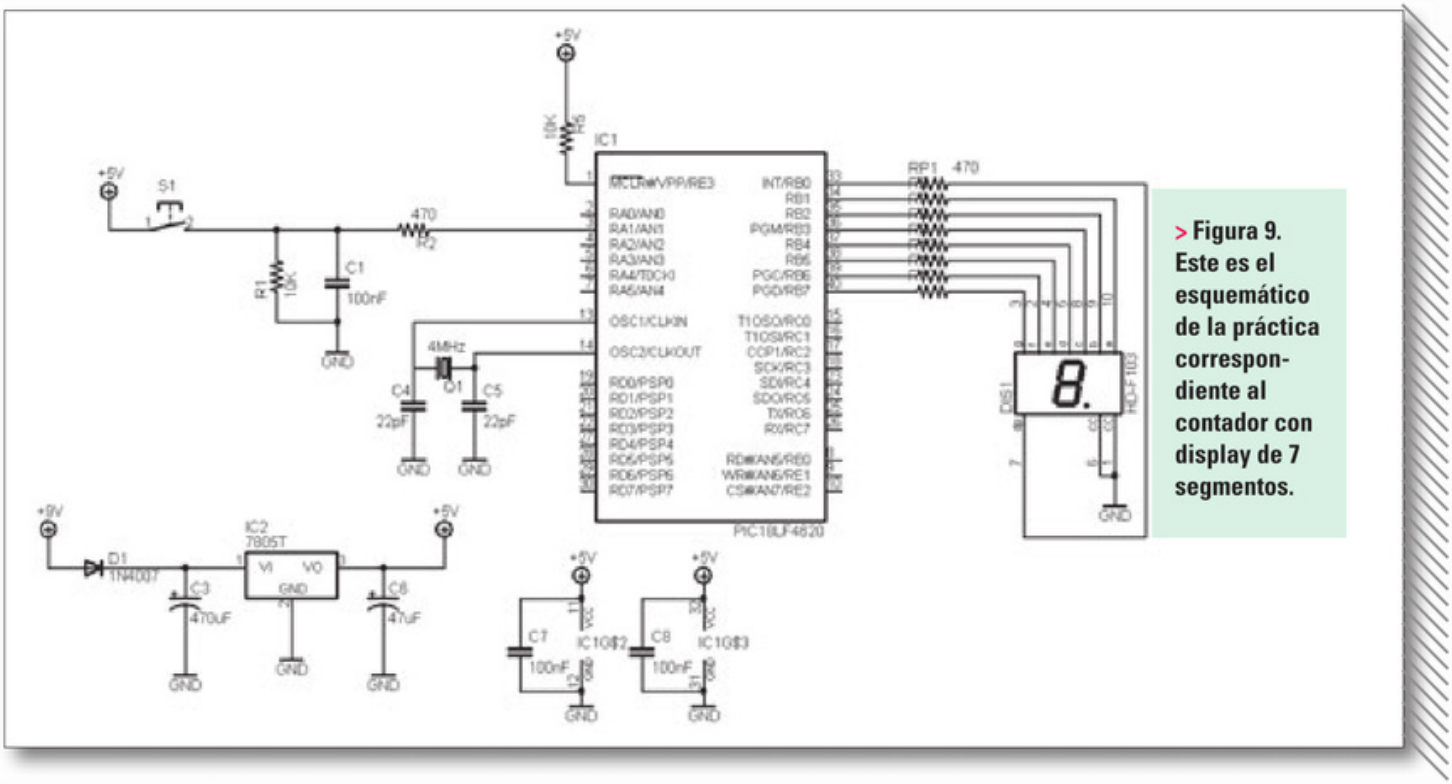
Para nuestro programa usaremos librerías básicas. Su función principal será explorar el estado del pulsador colocado en el puerto **RA1**. Si este se activa, entonces incrementaremos el valor de un contador, que solo cuenta de **0** a **9**. Luego, el valor será decodificado a 7 segmentos y enviado al **PORTB** para presentarlo en un display de cátodo común. ■





CONTADOR PWM

En estas páginas comenzamos por inicializar las variables del programa y los puertos del microcontrolador. Luego, aprenderemos a utilizar uno de los módulos internos del PIC.



> **Figura 9. Este es el esquemático de la práctica correspondiente al contador con display de 7 segmentos.**

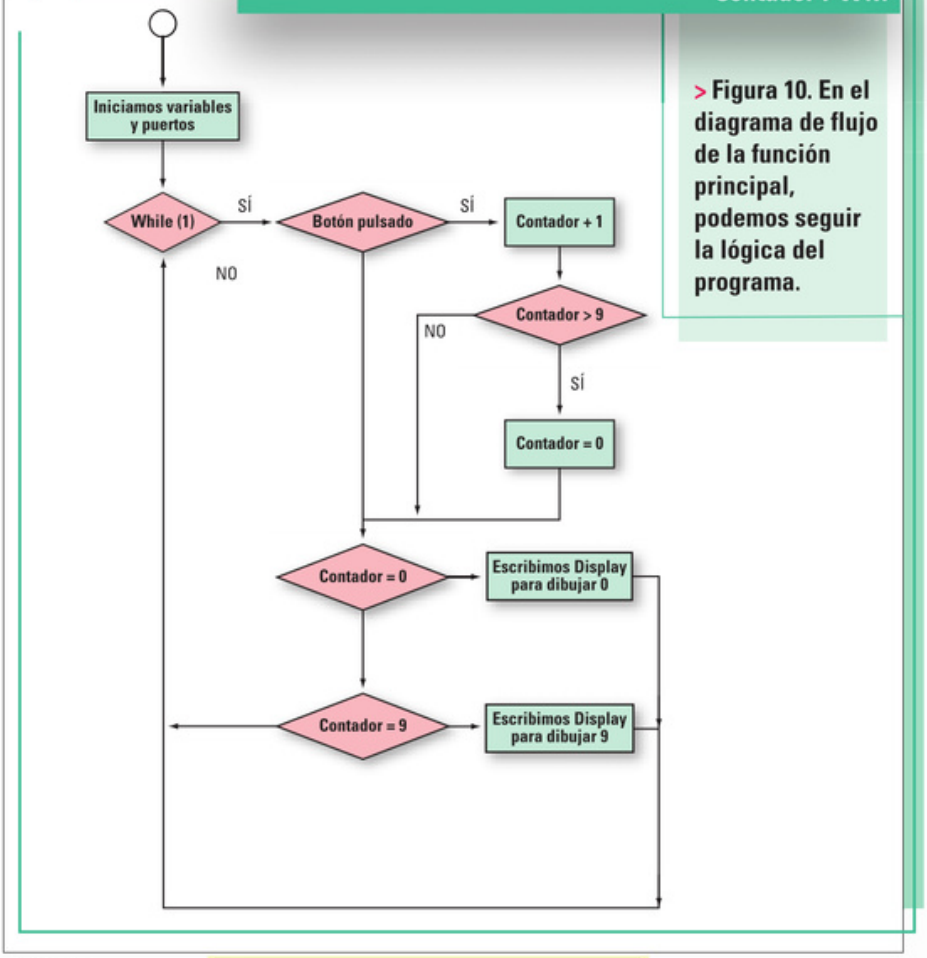
En el esquemático de la **Figura 9** vemos que el pulsador de entrada se conecta al pin **RA1**, y controlamos el display de 7 segmentos con el **puerto B**. Entonces, la configuración inicial será indicarle al PIC que el pin RA1 es una entrada de datos, y que el PORTB es un puerto de salida.

Una vez que configuramos el microcontrolador, entramos en un bucle infinito generado por la sentencia **While** (1); donde constantemente preguntamos cuál es el estado del pin RA1. Si se oprimió el pulsador, el programa incrementará la variable contador, y si este es mayor a 9, la cuenta se reseteará. Luego, según la cuenta del contador, convertimos ese valor para que sea visualizado en el display de 7 segmentos.

La directiva #pragma config permite configurar algunos recursos del PIC, como elegir el tipo de cristal.

En el diagrama de flujo de la **Figura 10** se presenta, de manera lógica, cuál es el procedimiento de la conversión. Primero, preguntamos si el contador es igual a 0; si lo es, ponemos en el puerto de salida el valor que coloca un 0 en el display. Si no lo es, pasamos a interrogar si el contador vale **1** y, así, sucesivamente hasta llegar a **9**.

En el diagrama de flujo, solo colocamos dos valores en la interrogación, para que el dibujo no se haga complicado. Luego, cuando pasamos estas interrogaciones al código, lo hacemos a través de la sentencia **switch-case**, donde, dependiendo del valor de la variable contador, escribimos el puerto **B** con

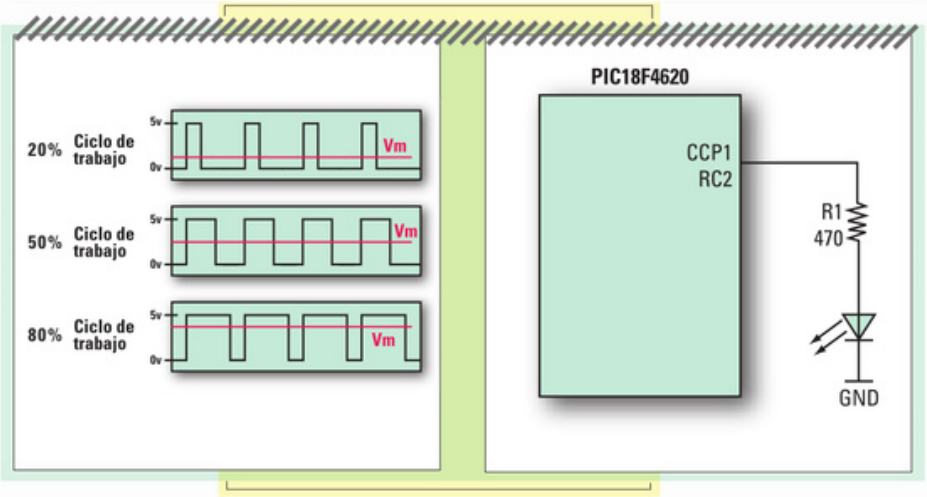


> Figura 10. En el diagrama de flujo de la función principal, podemos seguir la lógica del programa.

el dígito en 7 segmentos.
Observemos que, en el encabezado del código, además de incluir el archivo que define los registros de nuestros PICs, utilizamos la directiva **#pragma config** para configurar algunos recursos del PIC. En esas líneas le indicamos al microcontrolador que emplearemos un cristal externo del tipo **XT** para el PIC, desactivamos el **"perro guardián"** (WDT) y activamos el pin de **Reset**.

Práctica PWM

En esta práctica aprenderemos a utilizar uno de los módulos internos del PIC para generar una señal de ancho de pulso variable (PWM). Veamos de qué se trata.
Los microcontroladores PIC son capaces de producir pulsos de anchos variables en el tiempo por medio de un módulo interno llamado **CCP** (*Compare/Capture/PWM*, o comparación/captura/modulación de ancho de pulso). Las señales **PWM** permiten realizar diversas tareas, desde **dimmers** para LEDs (variación de la intensidad del LED), hasta controlar la velocidad de un motor eléctrico de corriente continua.
Todas estas aplicaciones están basadas en un principio de señales llamado **PWM**, donde estas modulan su ancho de pulso alterando su ciclo de trabajo o **Duty Cycle**, que es la relación entre el tiempo en que la señal se mantiene a nivel alto y el período de la señal, expresado en forma porcentual. A medida que el ciclo de trabajo aumenta, también se incrementa la potencia entregada por la señal. En la **Figura 11** vemos cómo aumenta el valor medio de la señal cuando crece el ciclo de trabajo, y se realiza de manera lineal con la siguiente fórmula:
Vm = CT (Ciclo de trabajo) x Vmax.



> Figura 11. Es posible variar el ciclo de trabajo de una señal, con lo cual se verán cambios en la tensión media.

> Figura 12. El pin del PIC es el RC2, el cual es la salida del módulo CCP que utilizaremos como PWM.

> EJERCICIO

Realizaremos un dimmer para LED, es decir que modificaremos la intensidad de este elemento con la técnica de PWM. El LED se encuentra conectado al pin RC2 del PIC por medio de una resistencia de 470 Ω, cuyo objetivo es limitar la corriente que pasa por él. Entonces, modulando la señal del pin donde está el LED, se modificará la corriente promedio que lo atraviesa, y cambiará la intensidad. Dos pulsadores conectados en RA1 y RA2 actuarán como control de luminosidad; usaremos uno de ellos para aumentar el brillo y el otro, para disminuirlo. La señal que module la intensidad del LED debe ser mayor a los 100 Hz.



Módulo PWM

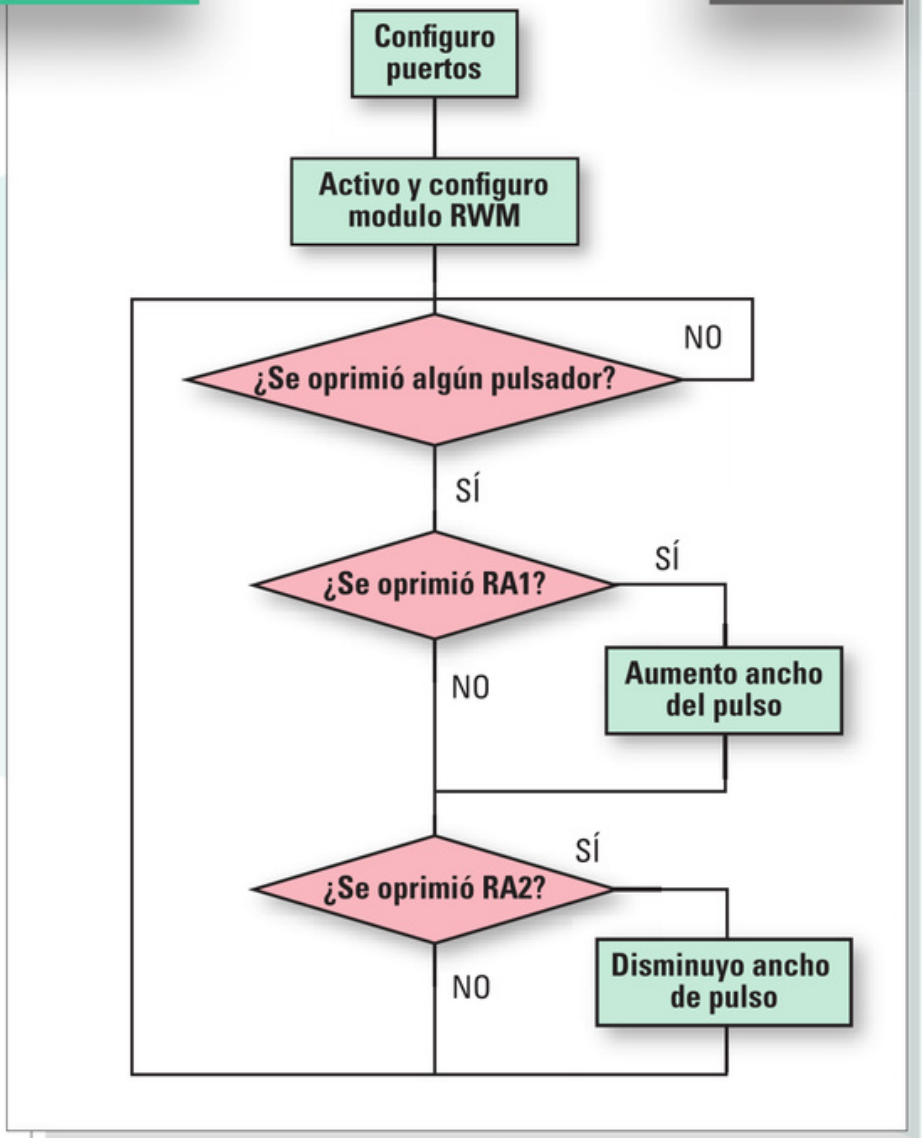
Vamos a utilizar el Módulo PWM. El PWM es una de las tres funcionalidades que posee el módulo CCP (comparación/captura/PWM) de los microcontroladores PIC. Este es controlado por uno de los registros especiales (SFR) del PIC, el CCP1CON. Los bits de este registro configuran al módulo para utilizarlo como PWM, haciendo que el pin RC2 actúe como la salida de la señal

Las señales PWM permiten realizar dimmers y controlar la velocidad de un motor

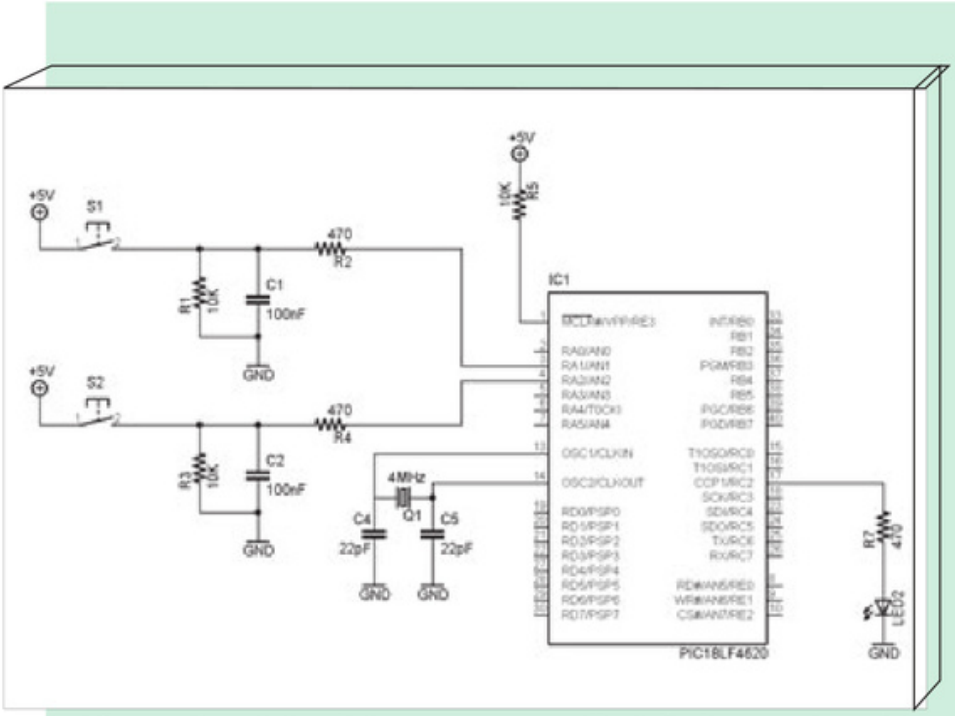
modulada; por esta razón, colocamos un LED allí. Cuando inicializamos el módulo y cargamos con un valor en el registro PR2, se realiza una comparación entre el valor del PR2 y un contador interno del PIC llamado Timer2. Si los valores son iguales, la señal se pone en 1, y esta operación marca el período de la señal. Luego, con el registro CCPR1L se establece el ciclo de trabajo de la señal cuando la comparación con el contador es la misma.

Código

En la **Figura 13** vemos el diagrama de flujo que resuelve el ejercicio, donde se comienza configurando los puertos del PIC. El pin **RC2** se configura como salida (allí encontramos el LED por controlar), y los pines **RA1** y **RA2**, como entradas digitales. El próximo paso es activar y configurar el módulo PWM. No entraremos en detalles con respecto a este tema, sino que recomendamos



> Figura 13. Este es el diagrama de flujo de la práctica.



> El esquemático de la práctica muestra una sección de la placa experimental que construimos, donde encontramos los pulsadores y el LED.

```
/* Includes */
#include "p18f4620.h"

#pragma code  // Zona de memoria de programa

void main (void)
{
        unsigned char brillo = 0;

        TRISCbits.TRISC2 = 0;       //RC2 salida
        ADCON1=0x0F;    //Configuramos todos los pines como digitales
 TRISA = 0b00000110;        // PORTA bit 1 y 2 entrada (Aqui estan
                                    // conectados los pulsadores)

        //Configuracion Timer 2
        // La configuracion de los temporizadores se vera en detalle en la
proxima
        // clase
        T2CON = 0b00000111;// Prescale = 1:16, timer on

        // Periodo de la señal
        PR2 = 249;
        // Con esta configuracion generamos un periodo de 4mS, este
periodo no
        // se modificara en todo el programa

        //Duty Cycle o ciclo de trabajo
        CCPR1L=brillo;
        // La variable brillo se encargara de modificar el ancho de pulso
        // de la señal, observar que tenemos un ciclo de trabajo que tiene
        // un resolucion de 8 bits.

        // Configuramos el modulo CCP del PIC para que trabaje en modo
PWM
        // modificando los bits del Registro CCP1CON
        CCP1CON = 0b00001100;


        while(1)
        {// Nos quedamos esperando a que se presionen los pulsadores
                while(PORTAbits.RA1 != 1 && PORTAbits.RA2 != 1);


                if(PORTAbits.RA1 == 1) // Se presiono RA1?
                {
                        brillo = brillo + 2;// Aumentamos brillo en multiplo
de 2
                        CCPR1L = brillo;// modificamos en ancho de pulso
                }
                if(PORTAbits.RA2 == 1) // Se presiono RA2?
                {
                        brillo = brillo - 2;// Decrementamos brillo
en multiplo de 2
                        CCPR1L = brillo;// modificamos en ancho de pulso
                }

        }
}
```

leer la hoja de datos del PIC para configurarlo correctamente. Con el registro **PR2** definimos el período de la señal en **4 mseg** y modificamos el ancho de pulso por medio del registro **CCPR1L**. Cuando terminamos de configurar los recursos del PIC, el programa se queda interrogando si se produce algún cambio en los pulsadores. Si lo detecta, pasa a interrogar cuál es el pulsador que se oprimió: si fue **RA1**, se aumenta el ancho de pulso de la señal modificando el registro CCPR1L con la variable brillo. En cambio, si fue **RA2** el que se oprimió, se hace disminuir el ancho de pulso y, luego, se vuelve a realizar todo otra vez.

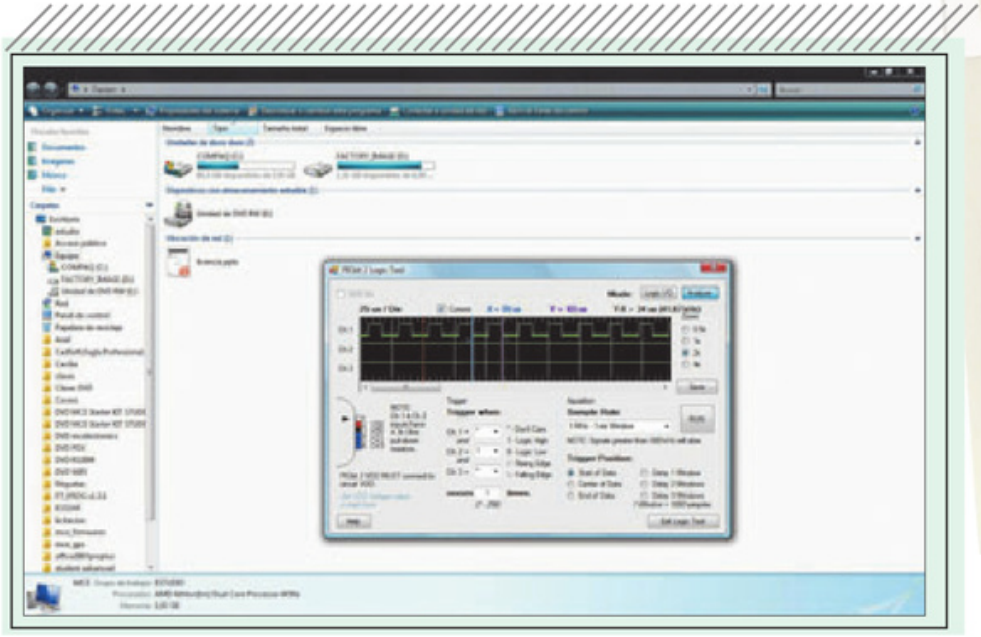
A partir del diagrama de flujo, obtenemos el siguiente **código en C** para el enunciado de esta práctica. En los comentarios, se explica cada línea de la configuración del PIC y el proceso de monitoreo de los pulsadores para aumentar o disminuir la intensidad del LED. Observemos que la variable **brillo** controla el ancho de pulso de la señal y, de ese modo, la intensidad en el LED. La configuración del temporizador del PIC no es una tarea simple, pero el módulo **PWM** lo necesita para su operación.

Una vez que tenemos el código fuente, generamos un nuevo proyecto en el **MPLAB** y simulamos el código con el **MPLAB SIM** para verificar que el programa no tiene errores y resuelve el enunciado del ejercicio. Luego, podemos grabarlo en el PIC y correr el programa en nuestra placa experimental.



Medición del ancho de pulso

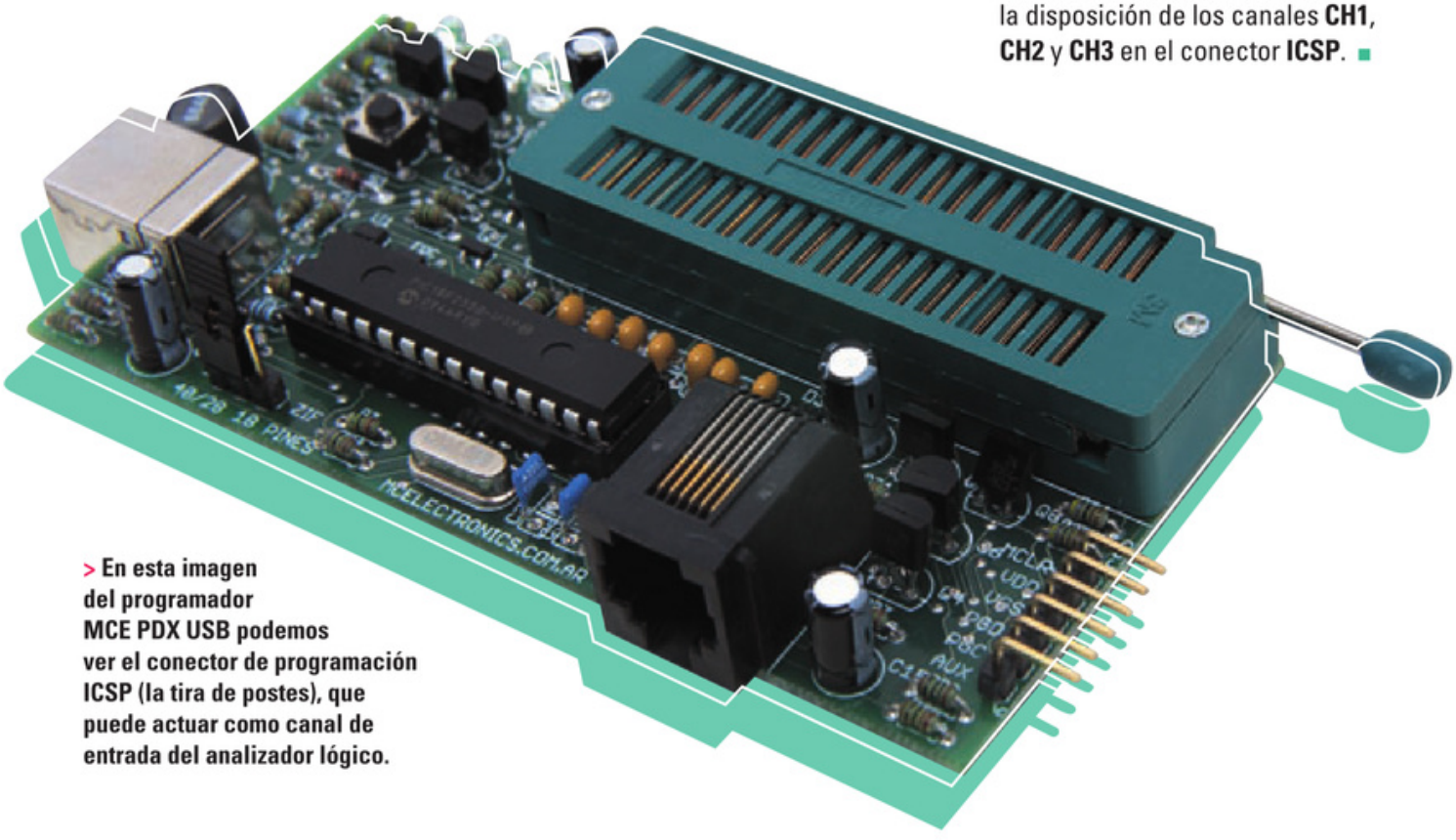
En nuestra placa experimental, podemos apreciar el funcionamiento de la señal **PWM** por medio del LED de prueba, donde se ve cómo varía la intensidad cada vez que oprimimos los pulsadores. También es posible visualizar la forma de onda de la señal que excita al LED en un osciloscopio, el que nos permite calcular la frecuencia y el ciclo de trabajo de la señal. Si no disponemos de un osciloscopio, debido a su elevado costo, existen herramientas alternativas que permiten visualizar la señal en la computadora. Una de ellas es el programador de **PICs MCE PDX USB**, que puede ser utilizado, en su función como analizador lógico, para observar los cambios de estado de una señal. También nos será de utilidad para visualizar la señal PWM que genera nuestro PIC en la práctica que acabamos de hacer. Este se comporta como un analizador lógico de tres canales, con una frecuencia máxima de muestreo de **1 MHz**; es decir que es capaz de tomar **1 millón** de muestras por segundo en cada canal.



> **Figura 14. La pantalla típica del analizador lógico del programador nos permite configurar la medición. También tenemos la posibilidad de utilizar cursores para medir el ancho de pulso de la señal PWM.**

Las conexiones de los canales del analizador están multiplexadas con el conector de programación **ICSP** que tiene esta herramienta, y desde allí podemos conectar cables a nuestra placa experimental y realizar las mediciones.

Para utilizar el programador como analizador lógico, debemos ejecutar el software **PicKit 2 programmer**, incluido en el programador **PDX**, y activar la herramienta lógica desde **Tool/Logic Tool**. Seleccionamos el modo **Analizador** para observar la forma de onda de la señal que genera el PIC. Veremos una pantalla como se muestra en la **Figura 14**, desde la cual podemos analizar la disposición de los canales **CH1**, **CH2** y **CH3** en el conector **ICSP**. ■



> **En esta imagen del programador MCE PDX USB podemos ver el conector de programación ICSP (la tira de postes), que puede actuar como canal de entrada del analizador lógico.**

PERIFÉRICOS DEL PIC18LF4620

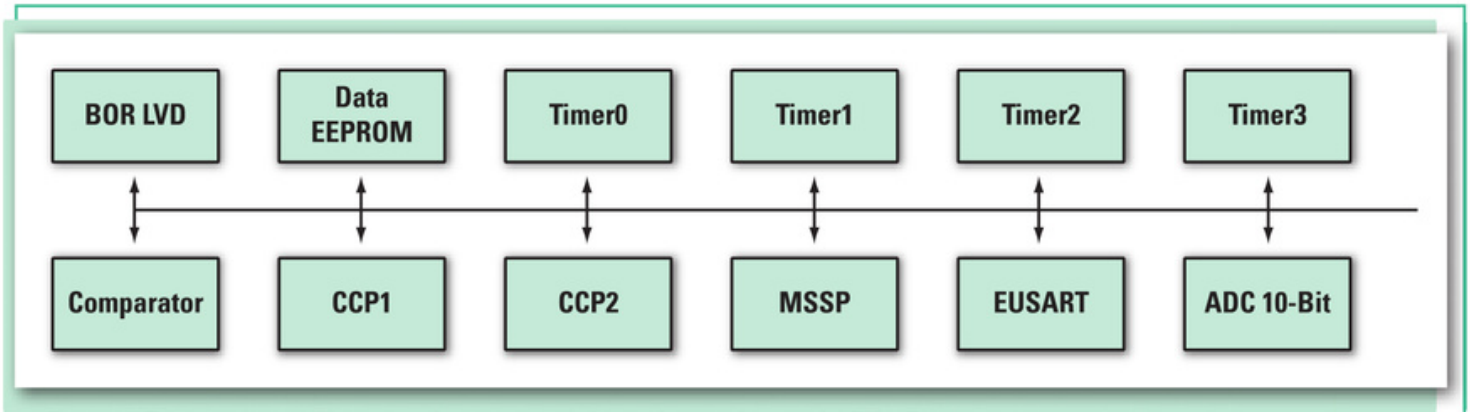
Los PIC18LF4620 incorporan todos los periféricos necesarios para realizar la mayoría de los sistemas de control que se usan en electrónica industrial.

Microchip dotó a su microcontrolador PIC18LF4620 de todos los periféricos necesarios para efectuar cualquier sistema de control de forma embebida. De esta manera, encontramos dentro de él todos los que hallaríamos en un PIC16F877, pero mejorados. El PIC18LF4620, además de los puertos I/O, cuenta con los siguientes elementos:

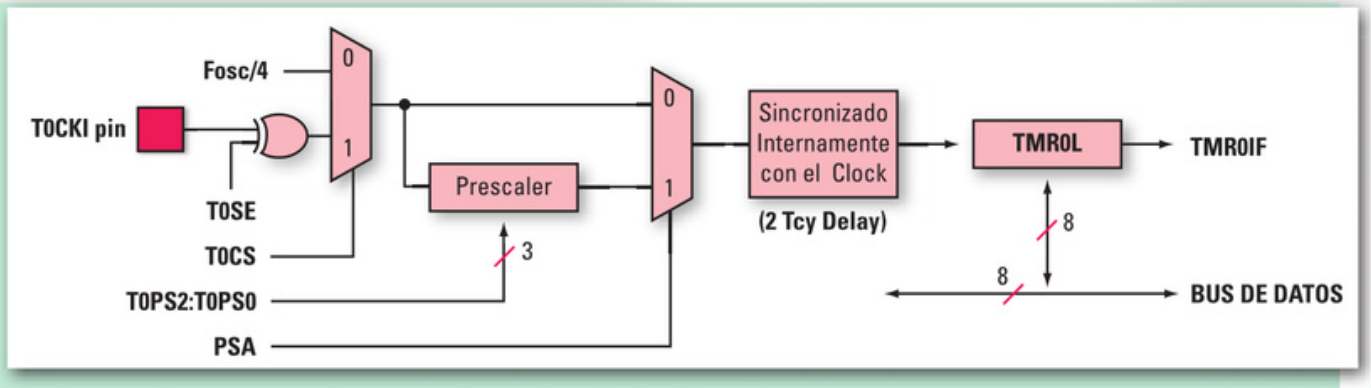
- Timer 0, que puede trabajar en modo de 8 o 16 bits.
- Timer 1, de 16 bits.
- Timer 2, de 8 bits.
- Timer 3, de 16 bits.
- Conversor A/D tipo SAR de 10 bits y 13 canales.
- Puerto serie sincrónico maestro (MSSP) para comunicaciones I2C y SPI.
- Unidad de comunicaciones serie asíncronas mejorada (EUSART) con autodetección de velocidad de comunicaciones (*Auto Baud Rate*) y soporte para RED LIN 2.0.
- Módulo captura/comparación/PWM mejorado (ECCP), que incorpora control PWM para puente completo, con control de estrangulación de PWM automática y control de tiempos muertos (*Dead Time Control*).
- Puerto paralelo esclavo (PSP), para comunicaciones con microprocesadores de 8 bits.
- Memoria EEPROM para datos con capacidad para almacenar hasta 1024 bytes.

El PIC18LF4620 tiene los periféricos necesarios para realizar cualquier sistema de control.

Estos periféricos nos permiten realizar cualquier sistema de control de forma embebida, es decir, usando parte de los componentes internos y sin tener que recurrir a ninguno externo. Por ejemplo, podríamos realizar un control de velocidad para un motor de corriente continua. En este caso, usaríamos el módulo **ECCP** funcionando como **PWM** en modo **full bridge** (puente completo). Por medio del conversor A/D, en un canal colocaríamos un potenciómetro para fijar la velocidad; en otro, un preset para setear el torque, finalmente, en otros dos canales podríamos tener un monitoreo de la corriente del motor y la fuerza contra electromotriz que este genera para conocer la velocidad del motor.



> Figura 15. Estos son los periféricos que forman parte de la arquitectura interna del PIC18F4620.



> Figura 16. Observamos el diagrama en bloques funcional del timer 0.

Los timers

Los temporizadores o timers son contadores binarios que pueden contar pulsos internos o externos. En el primer caso se los denomina temporizadores, mientras que en el segundo se los llama contadores de eventos. Estos timers pueden tener 8 o 16 bits. En el PIC18LF4620 se han integrado cuatro timers, conocidos como 0, 1, 2 y 3.

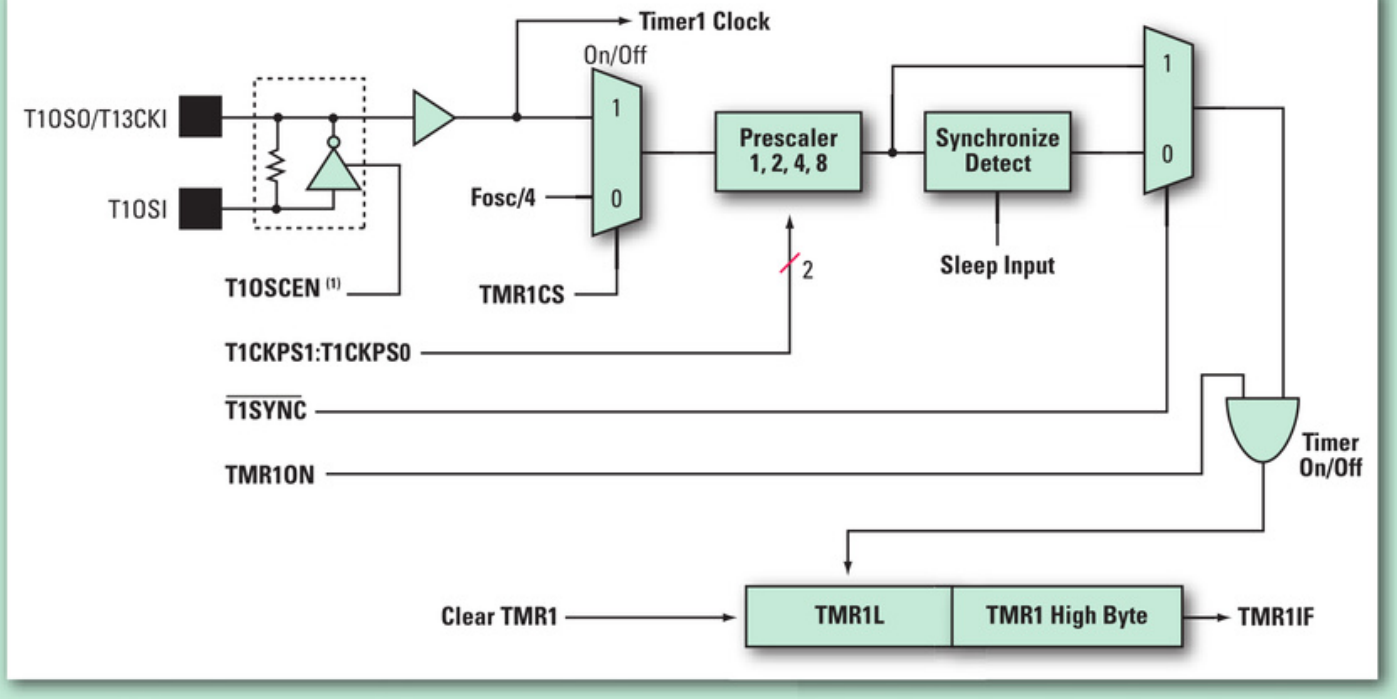
El timer 0

Este temporizador puede trabajar a 8 o 16 bits, y está preparado para contar pulsos externos o internos. Cuando trabaja excitado por pulsos externos, cuenta aquellos que llegan al teminal T0CKI. A este timer se le puede agregar un módulo divisor de frecuencia programable (prescaler) para que el avance sea de forma más lenta. El resultado de la cuenta se almacena en los registros TMR0L y TMR0H. Cuando el timer opera en modo de 8 bits, solo se usa el registro TMR0L. Sin embargo, cuando opera en 16 bits, se usan los dos registros.

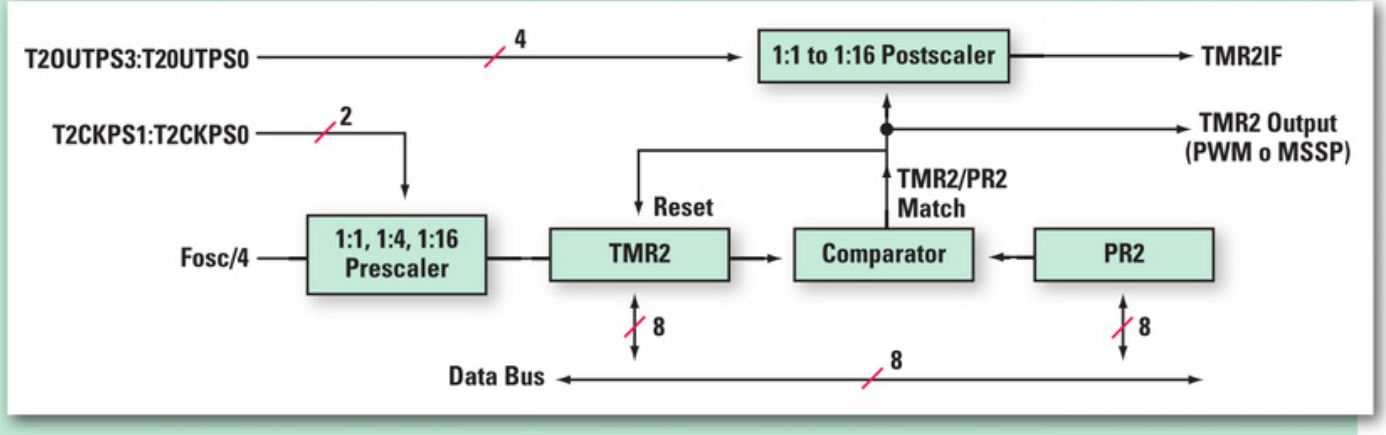
Los temporizadores o timers son contadores binarios que pueden contar pulsos internos o externos.

Timers 1 y 3

Ambos son de 16 bits y, estructuralmente, iguales; solo existe una leve diferencia entre ellos: el timer 3 tiene la posibilidad de un modo de disparo especial por evento. En los dos casos, el contador está formado por dos registros de 8 bits, que pueden ser disparados por eventos externos o internos. Incluso, cuentan con la posibilidad de activar un driver inversor para configurar un oscilador de baja frecuencia realimentado externamente por medio de un cristal cuya frecuencia típica es de 32.768 KHz. Mediante este cristal, es posible crear una base de tiempos para armar un reloj de tiempo real dentro del mismo PIC.



> Figura 17. Podemos apreciar el diagrama en bloques funcional del timer 1.

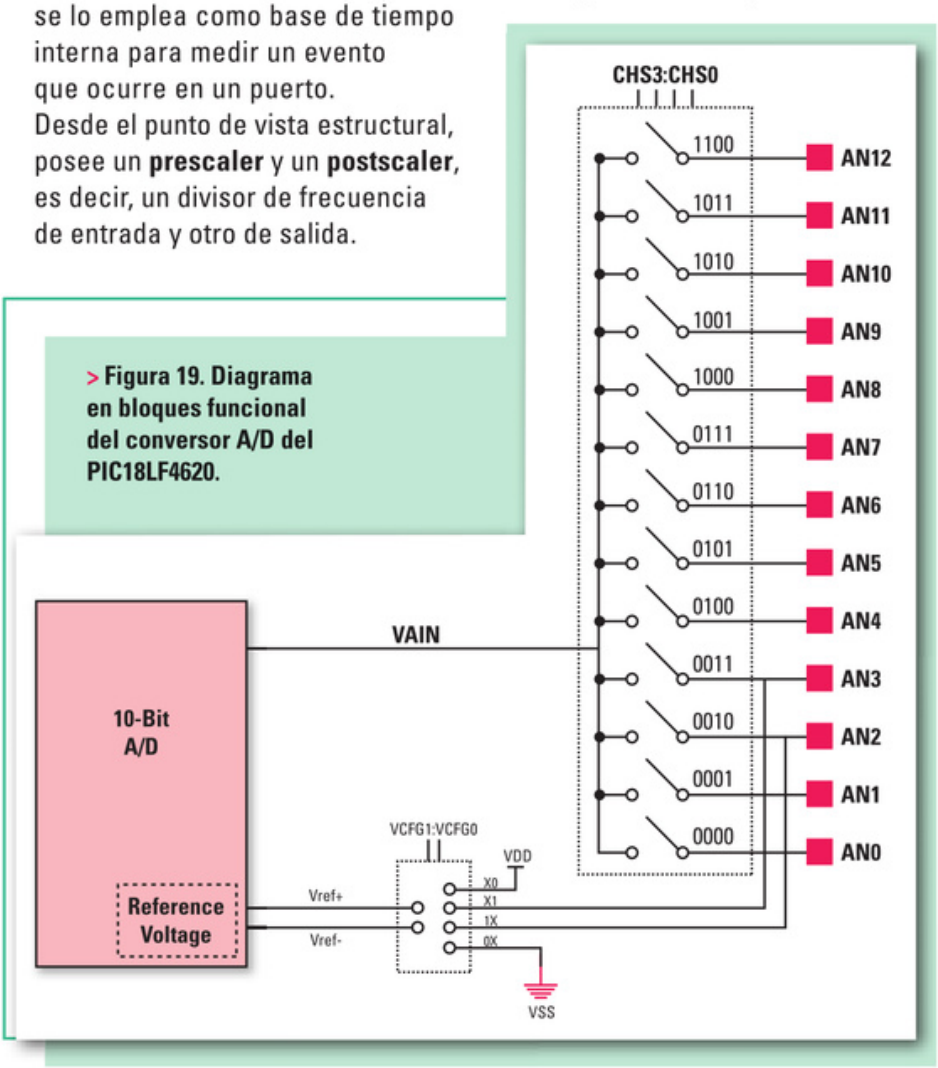


> Figura 18. Diagrama en bloques funcional del timer 2.

Timer 2

El timer 2 es de 8 bits, a diferencia de los anteriores, y no tiene la posibilidad de ser excitado por una señal externa, sino que solo puede hacerlo internamente por la señal del oscilador principal **FOSC/4**.
Es usado como base de tiempo para la generación de la señal PWM y, en muchas aplicaciones, se lo emplea como base de tiempo interna para medir un evento que ocurre en un puerto.
Desde el punto de vista estructural, posee un **prescaler** y un **postscaler**, es decir, un divisor de frecuencia de entrada y otro de salida.

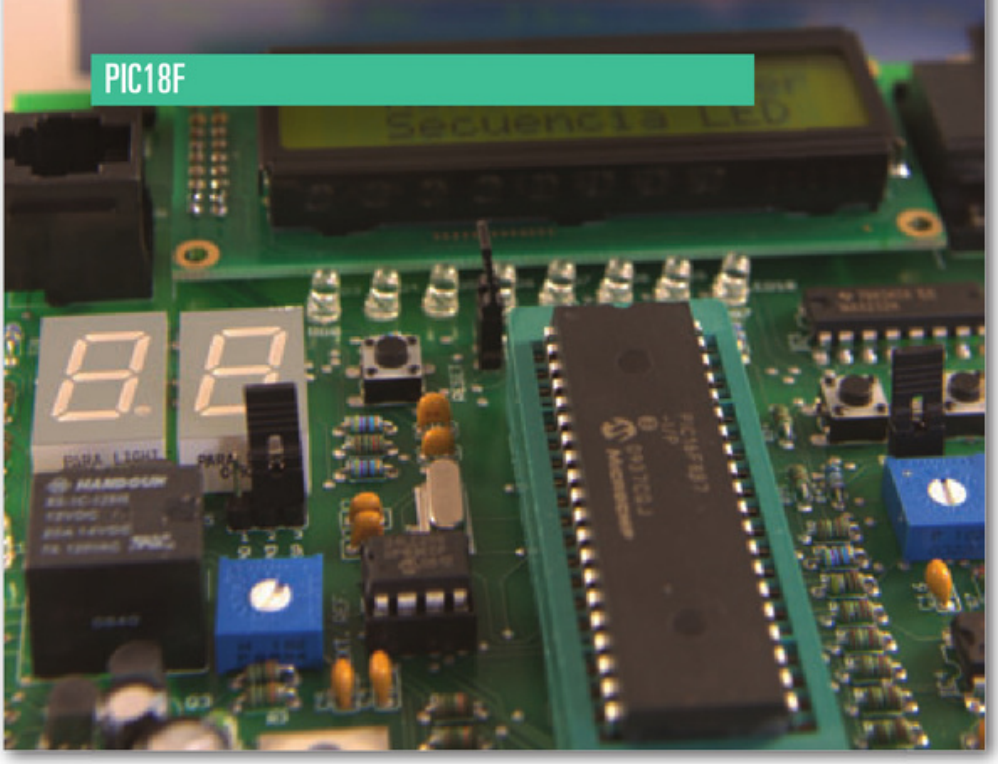
El **timer2** trabaja comparándose siempre contra el contenido de un registro denominado **PR2**. De esta manera, cuando alcanza a tener el mismo valor que hay en el registro PR2, el módulo comparador, encargado de comparar ambos registros, genera un pulso de salida. Dicho pulso es usado para reiniciar el timer 2 a 0 y para ser enviado hacia el PWM, el MSSP. El **postscaler**, luego de dividir la señal, pondrá en 1 la bandera de interrupciones TMR2IF.
El módulo PWM usará esta salida del comparador como base de tiempo para hacer funcionar el PWM y, en el caso del MSSP, se podrá usar como base de tiempo para generar la señal **SCK** o **SCL**.



> Figura 19. Diagrama en bloques funcional del conversor A/D del PIC18LF4620.

El conversor analógico-digital

Los PIC18F, al igual que algunos modelos de PIC 16F, están equipados con un conversor A/D de 10 bits, cuya precisión es más que suficiente para realizar cualquier instrumento para control industrial.
El conversor es del tipo **SAR**, por lo tanto, es muy rápido, y en el caso del PIC18F4620, posee 13 canales de conversión (AN0-AN12), lo cual nos permite monitorear hasta esa cantidad de señales analógicas mediante el método de multiplexión de canales.
Para controlar el funcionamiento del conversor existen tres registros de control, denominados **ADCON0**, **ADCON1** y **ADCON2**.
El resultado de la conversión se lee desde los registros **ADRESL** y **ADRESH**. A diferencia del PIC16, el conversor equipado en el PIC18 puede seguir operando aunque el microcontrolador esté en modo SLEEP.



La placa de entrenamiento nos permite manejar los periféricos básicos que aprendimos a utilizar en este libro.

Placas entrenadoras

Para entrenarnos en el uso de los microcontroladores PIC existen en el mercado placas específicas en las cuales aprender a utilizar todos los recursos disponibles que estuvimos viendo hasta el momento. Muchas de ellas cuentan con LEDs indicadores, pulsadores, potenciómetros, displays de 7 segmentos, pantallas LCD y bornes de salidas. En algunas ocasiones, es más económico comprar una placa entrenadora que armar una cuando esta cuenta con integrados SMD, por ejemplo. Entonces, estas herramientas, junto con un programador, son ideales para quienes desean iniciarse en el mundo de los microcontroladores. Una tarjeta de este tipo es la MCE STARTER KIT STUDENT de MCElectronics. ■

Para hacer funcionar el conversor, vamos a configurarlo del siguiente modo:

- Cargamos los puertos que serán analógicos o digitales.
- Seleccionamos el canal de entrada que queremos leer.
- Seleccionamos el tiempo de adquisición adecuado.
- Seleccionamos el clock para el conversor en cuestión.

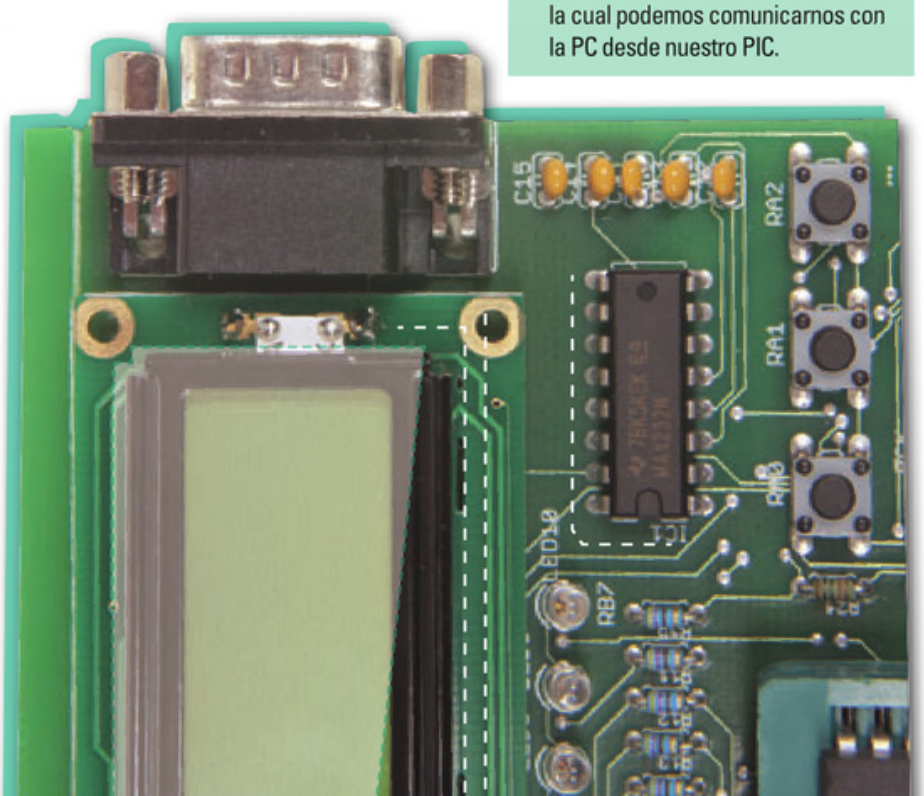
Si trabajamos por interrupciones, deberemos realizar la siguiente configuración:

- Borrar la bandera de interrupciones del conversor A/D, bit ADIF.
- Poner en 1 el bit ADIE para habilitar la interrupción por final de conversión y, luego, el GIE mediante el cual habilitamos el sistema general de interrupciones.

El conversor equipado en el PIC18 puede seguir operando aunque el microcontrolador esté en modo SLEEP.

- Esperar el tiempo de adquisición que sea necesario.
- Iniciar la conversión poniendo en 1 el bit GO/DONE.
- Si no usamos la interrupción, revisar el estado que adquiere el bit GO/DONE, ya que, al finalizar el proceso de conversión, este será puesto en 0 por el hardware del conversor, lo cual nos indica que el valor analógico ya fue convertido a formato digital.
- Leer el valor convertido desde los registros ADRESL y ADRESH, y si se usó la interrupción, borrar el bit ADIF.
- Para reiniciar la conversión, esperar como mínimo dos tiempos de adquisición (Tad).

La placa MCE Starter Kit Student cuenta con una conexión RS232, con la cual podemos comunicarnos con la PC desde nuestro PIC.



MANEJO DEL DISPLAY LCD INTELIGENTE

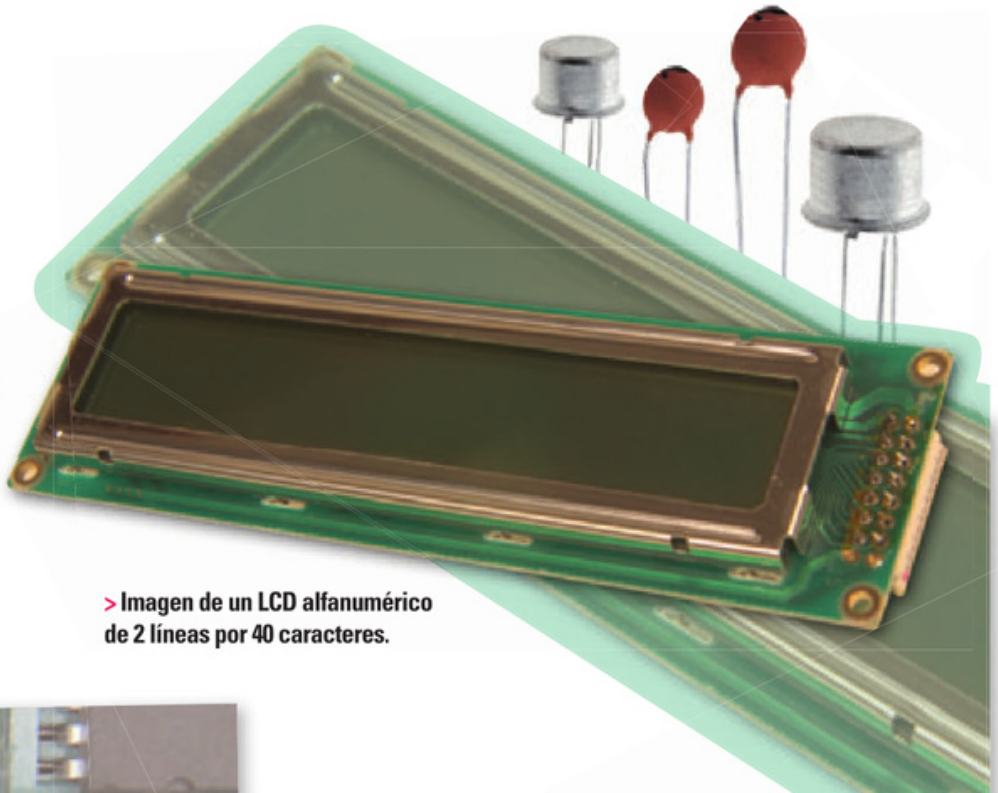
Estos displays son usados para la implementación de interfaces de usuario. Su bajo costo y facilidad de manejo los ha transformado en uno de los periféricos preferidos.

Los LCDs inteligentes son dispositivos periféricos usados para mostrar información. En ellos podemos representar todo tipo de mensajes conformados por letras, números e, incluso, caracteres gráficos programados por el usuario. Este tipo de display se denomina inteligente porque incluye sus propios controladores sobre el PCB que soporta a todo el sistema. Tiene un bus de datos capaz de operar en 8 o en 4 bits para comunicar la información, y cuenta con un bus de control formado por tres líneas. Son considerados periféricos en sí mismos.
Estos componentes surgieron a partir de mediados de los años 80, cuando la firma Hitachi desarrolló un controlador específico para tal fin denominado **HD44780**, el cual fue mejorado luego por Samsung. Este controlador permitió la contracción de distintos modelos de LCDs comerciales; tanto es así que hoy podemos encontrar muchas ofertas de distintos fabricantes, como Winstar o Truly, por nombrar solo dos de los muchos que hay. En el mercado las ofertas disponibles son:

- 2 líneas x 8 caracteres en cada una, con back light (luz de fondo).
- 1 línea x 16 caracteres, con y sin back light (opcional).
- 2 líneas x 16 caracteres cada una, con y sin back light.
- 2 líneas x 40 caracteres cada una, con y sin back light.
- 2 líneas x 20 caracteres cada una, con back light.
- 4 líneas x 20 caracteres cada una, con back light.

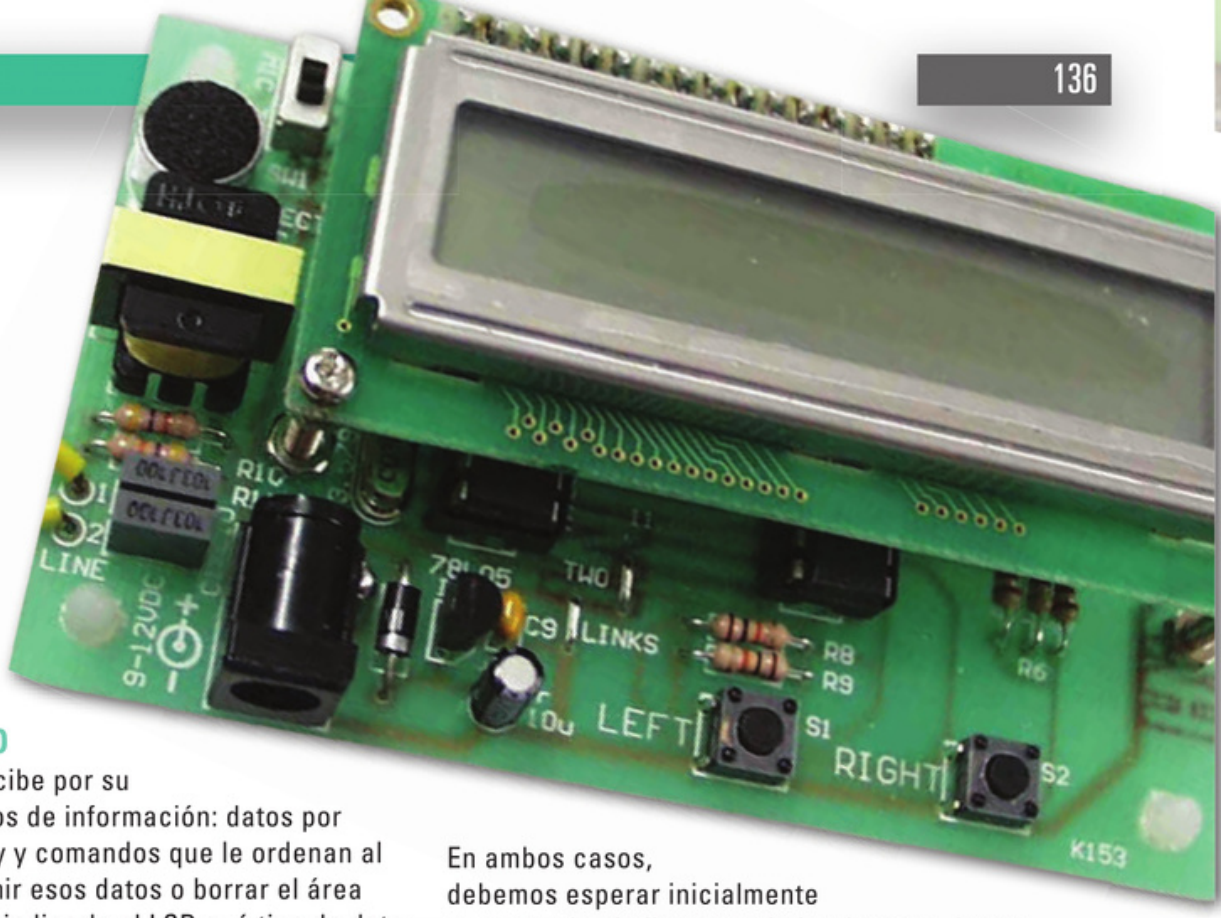
Estos dispositivos se alimentan con **5 Volts**, tienen un control de contraste, y el back light o luz de fondo puede mostrar distintos colores, siendo los más típicos los amarillos con letras en negro, los verdes con letras en negro, y los azules con letras en blanco. También los hay con back light inverso, es decir, con pantalla negra y letras en amarillo.

Según el estado lógico que apliquemos, el LCD interpretará si el dato enviado es un carácter por imprimir o un comando.



> Imagen de un LCD alfanumérico de 2 líneas por 40 caracteres.





Comandos del LCD

El LCD inteligente recibe por su bus de datos dos tipos de información: datos por imprimir en el display y comandos que le ordenan al display dónde imprimir esos datos o borrar el área visible del LCD. Para indicarle al LCD qué tipo de dato le estamos enviando, existe un terminal denominado RS. Según el estado lógico que apliquemos en él, el LCD interpretará si el dato enviado es un carácter por imprimir o un comando. De este modo, si queremos imprimir un carácter, habrá que poner en 1 el terminal RS; para indicarle que el dato enviado es un comando, debe estar en 0.

Los comandos más representativos que podemos enviarle al LCD se muestran en la **Tabla 1**. Además de ellos, existen otros que permiten controlar el modo de operación del display en 4 u 8 bits, así como también señalar si el cursor aparecerá titilante o no.

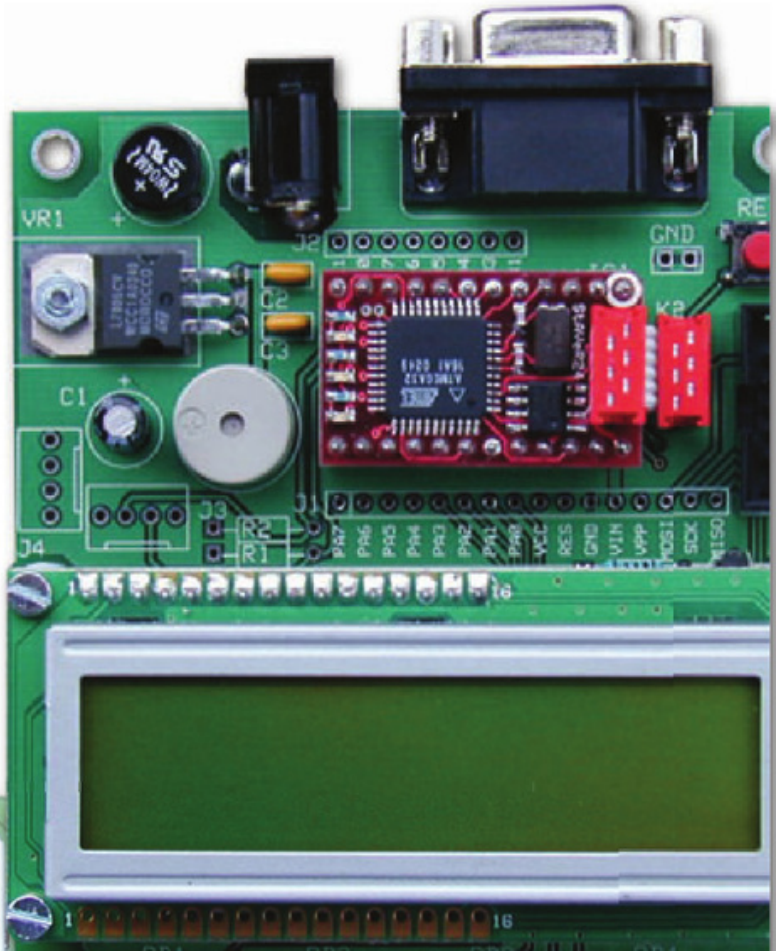
Para que el display funcione, debe ser inicializado, y esto se logra mandándole una secuencia de comandos, variable según el modo de operación de este (es decir, si está conectado en 8 o en 4 bits).

COMANDOS PARA EL LCD	
Función	**Comando**
Borrar pantalla	(01H)
Posicionar cursor al principio	(02H)
Encender pantalla	(0EH)
Apagar pantalla	(0AH)
Desplazar el cursor a la izquierda	(10H)
Desplazar el cursor a la derecha	(14H)
Desplazar la pantalla a la izquierda	(18H)
Desplazar la pantalla a la derecha	(1CH)

> **Tabla 1. La letra H al final del número indica que el valor está expresado en hexadecimal.**

En ambos casos, debemos esperar inicialmente unos 15 min al alimentar el circuito para, luego, activar la rutina de inicialización.

Si bien todo esto parece complejo de entender, los fabricantes de compiladores han desarrollado una serie de librerías en los lenguajes de alto nivel que nos permiten, de manera muy simple, inicializar el LCD, y enviarle caracteres, mensajes completos y comandos de posicionamiento. En el caso, por ejemplo, del compilador C18, Microchip provee de una librería denominada XLCD.h, muy completa y fácil de manejar.



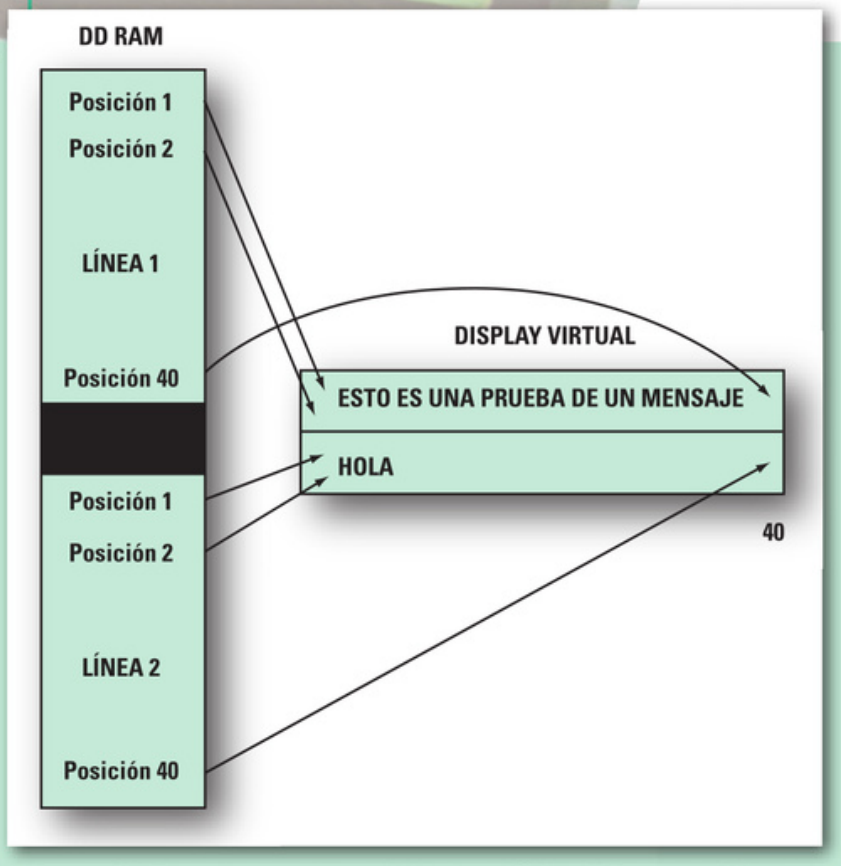
Caracteres del LCD

Hemos mencionado anteriormente que al LCD podemos enviar comandos y caracteres imprimibles. En este caso, los caracteres se mandan en formato ASCII, y todos los establecidos por el código ASCII estándar se imprimirán. Sin embargo, si bien ha existido un acuerdo entre todos los fabricantes en cuanto a los primeros 128 símbolos, no sucedio lo mismo con los códigos que van desde el 128 hasta el 255 (conocidos como tabla ASCII extendida).
En esta tabla, por ejemplo, los fabricantes orientales suelen colocar los símbolos de su alfabeto.
Para decirle al display que le enviamos un carácter imprimible, debemos poner en **0** el terminal **RS**. Los caracteres que enviamos se almacenan en la **memoria RAM** de video del controlador del LCD, denominada **DDRAM**. Esta puede albergar un total de **80 caracteres**. Sin embargo, solo serán visibles los que permita la ventana del LCD. Por ejemplo, en un LCD tipo 2 x 16 (2 líneas de 16 caracteres por línea), solo veremos 32 caracteres ubicados en las direcciones de la memoria RAM, que van desde:

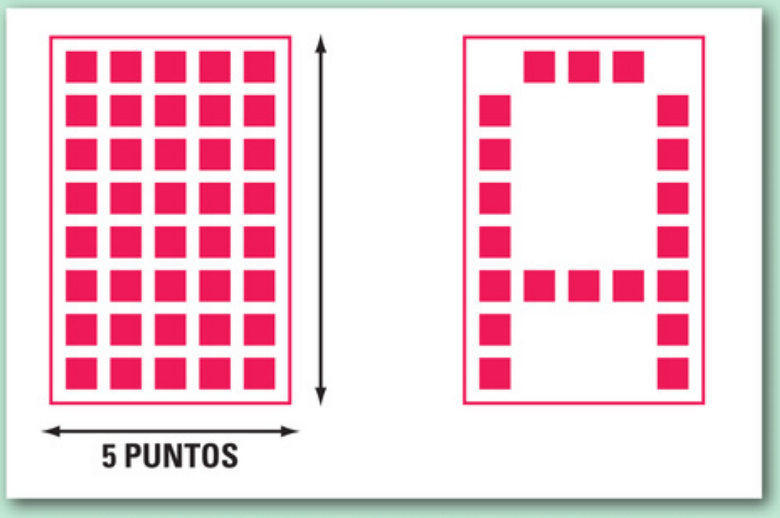
Primera línea: 00-0FH
Segunda línea: 40-4F

Esto solo es cierto para los **LCD tipo 2 x 16**, ya que los **2 x 20** tienen una mayor área visible. Por lo tanto, siempre deben consultarse las hojas de datos del controlador que tenga el LCD.
Para construir un carácter, el controlador del LCD lo dibuja sobre una matriz de puntos, que puede tener dos tamaños: **5 x 7** o **5 x 10** (esto es programado por el usuario). Además, el usuario tiene la posibilidad de armar sus propios caracteres, por ejemplo, para introducir un logo en el LCD. En este caso, es posible programar hasta 8 caracteres de 5 x 7 o hasta 4 caracteres de 5 x 10. Estos se almacenarán en una RAM distinta, denominada RAM del generador de caracteres o **CGRAM**.

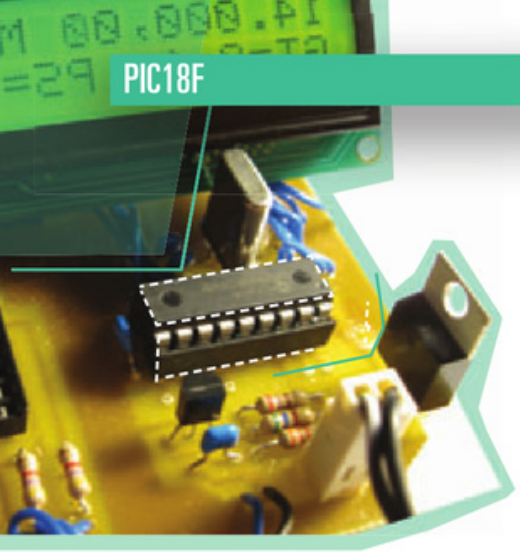
Para construir un carácter, el controlador del LCD lo dibuja sobre una matriz de puntos.



> Figura 20. Mapa de la memoria RAM de un LCD de 2 líneas por 16 caracteres.



> Figura 21. Las dos matrices de caracteres que es posible configurar por medio del envío del comando apropiado.



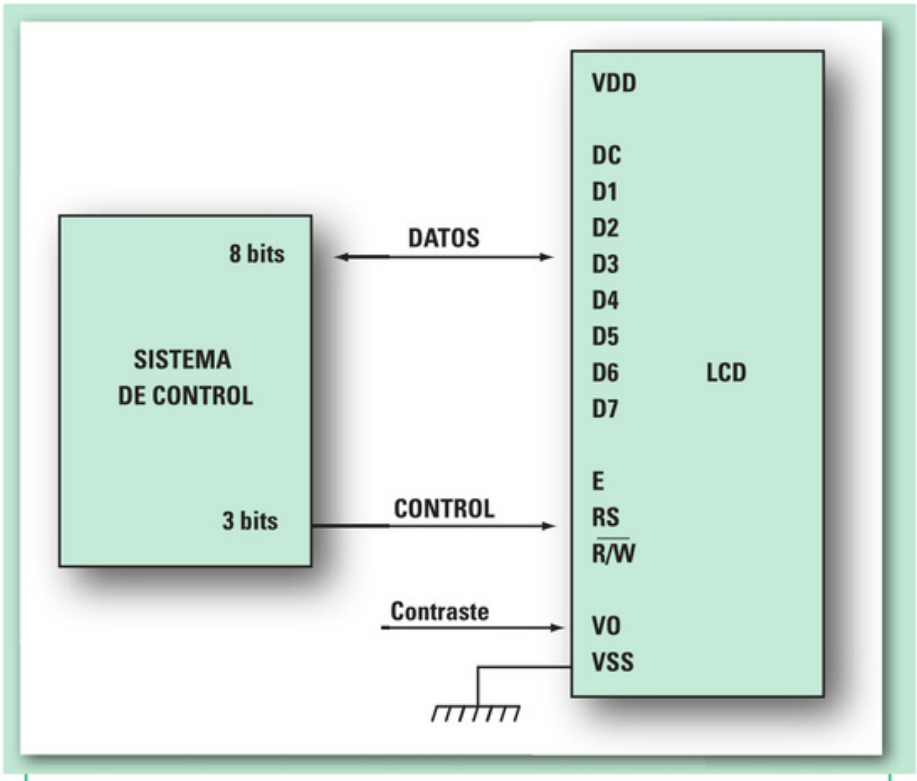


Figura 22. Observamos cómo debe conectarse el LCD en 8 bits al microcontrolador.

Conexión del LCD en 8 bits

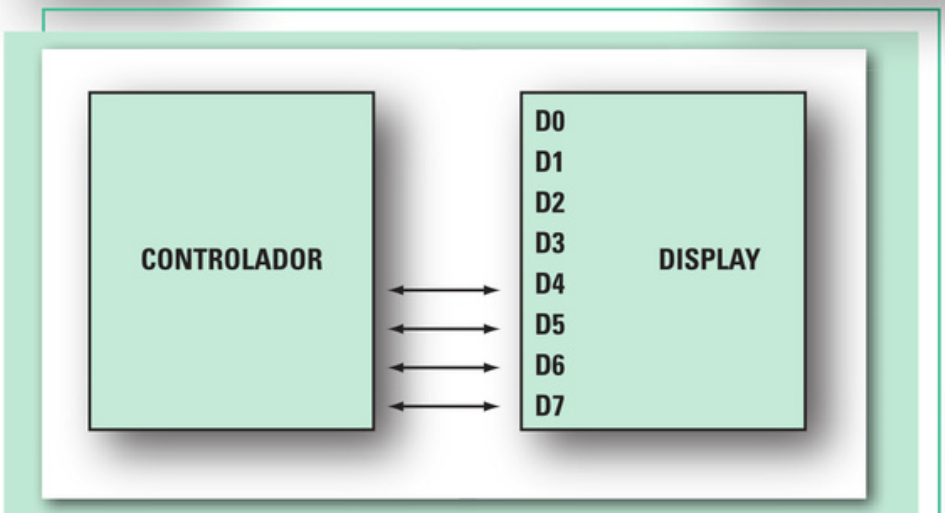
Este es el modo normal de operación de los LCD, y donde obtenemos el mayor rendimiento en cuanto a velocidad de comunicaciones. En este caso, debemos conectar el puerto del microcontrolador –por ejemplo, el **PORTD**– al bus de datos del **LCD**, haciendo corresponder el número de bits en la conexión. Además, necesitaremos un par de puertos auxiliares para controlar la comunicación, lo cual se consigue manejando las líneas de control del LCD (**Tabla 2**).

Para leer el estado interno, además de poner el terminal R/W en 1, también debemos poner en 1 el terminal RS.

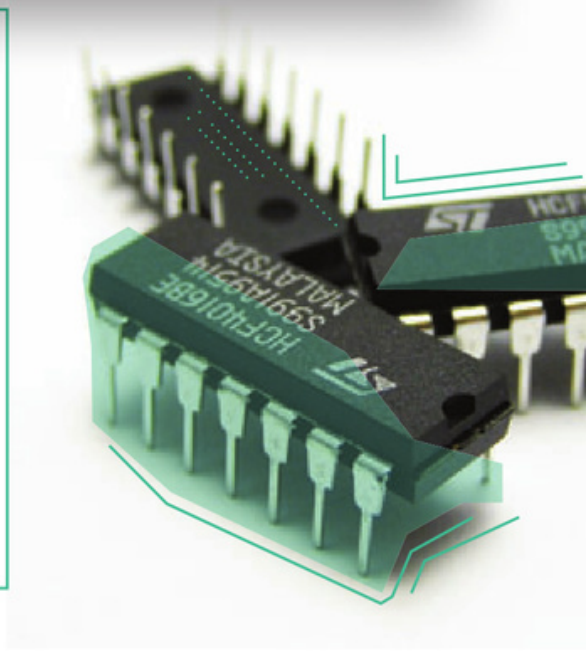
Existen dos maneras de enviar información al LCD. La primera es hacerlo de a un carácter por vez, esperando unos 10 milisegundos entre cada envío hasta terminar con el mensaje. Si bien este método, denominado de fuerza bruta, es lento, nos permite ahorrar un puerto, ya que nunca leemos el LCD. En este sistema, el terminal **R/W** debe ir conectado directamente a masa.
La otra manera es más inteligente, porque usamos el terminal **R/W** para leer el LCD antes de enviar un dato, para saber si está ocupado. Si no lo está, mandamos el dato; en caso contrario, esperamos un tiempo y repetimos la lectura.
Este método nos permite comunicarnos de forma más rápida, pero necesitamos un puerto más del microcontrolador, por lo cual no es muy utilizado.
En este caso, el puerto del microcontrolador debe reconfigurarse como entrada –cuando leemos el estado interno del LCD– y como salida –cuando escribimos un dato–, dado que la comunicación es bidireccional.
Para leer el estado interno, además de poner el terminal **R/W** en **1**, también debemos poner en 1 el terminal **RS**. El LCD enviará su estado interno sacando un byte, donde su bit de mayor peso (bit 7) nos indicará si está ocupado o no. Este bit se denomina **bit de busy**.

DESCRIPCIÓN DE LOS PINES DE CONTROL DEL LCD	
Terminal	**Descripción**
E (Enable o habilitador)	**Esta línea debe recibir un pulso que le permite capturar el dato colocado sobre el bus.**
RS (Select Register o selector de registro)	**Mediante esta línea le decimos al LCD si el dato que hemos puesto sobre el bus es un dato o un comando.**
R/W (Read/Write o lectura/escritura)	**Si estamos escribiendo un dato, este pin deberá ponerse en 0. Si queremos leer el estado interno del LCD para saber si está ocupado o no, lo ponemos en 1.**

> Tabla 2. Podemos ver aquí qué función cumple cada uno de los terminales destinados a controlar la comunicación con el LCD.



> **Figura 23. Esta es la manera de conectar el LCD en 4 bits al microcontrolador.**



Conexión del LCD en 4 bits

Otra forma de conectar el LCD a un microcontrolador es la de 4 bits, en cuyo caso ahorramos muchos puertos. Para hacer la conexión, se usan los 4 bits de mayor peso del bus de datos del LCD. Como en el caso anterior, podemos controlar las transferencias de datos usando el terminal **R/W**, que en este último ejemplo nos permitirá ahorrar otro pin más. Sin embargo, en la conexión de 4 bits es aconsejable leer el estado interno del LCD, ya que las transferencias son más lentas.

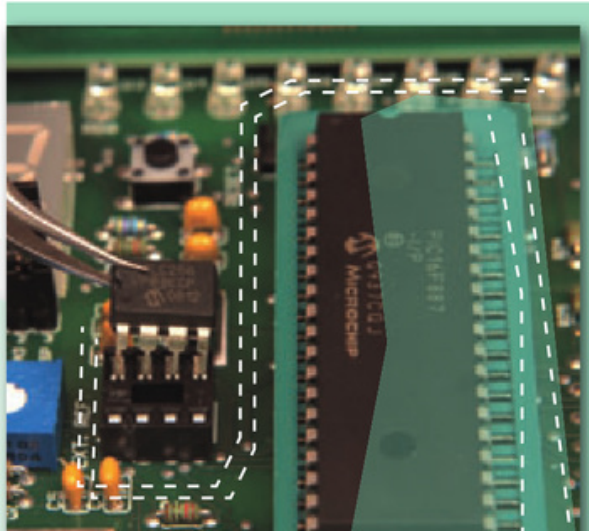
En la conexión de **4 bits** las transferencias se duplican, puesto que para enviar cada dato, hay que dividirlo en dos transferencias de 4 bits cada una. Esto aumenta el tiempo del proceso y hace que la comunicación resulte más lenta. Sin embargo, las librerías de los lenguajes de alto nivel están preparadas para la operación tanto en 8 como en 4 bits.

El único cuidado que debemos tener es que los puertos elegidos para conectar el microcontrolador con el LCD deben pertenecer al mismo **PORT**. En este caso, si usáramos el **PORTB**, podríamos conectar el LCD a los terminales **RB0** a **RB3** o **RB4** a **RB7**. De ningún modo podemos usar puertos de distintos PORTS en lo que respecta al bus de datos. Sin embargo, es posible usar cualquiera para las líneas de control. Por lo general, en la práctica es común ver, por ejemplo, todo el LCD conectado al mismo PORT (PORTB o PORTD).

También hay que tener cuidado en cuanto a la inicialización del LCD, porque en dicha rutina cambian los códigos que se le envían este. Si no se están usando las librerías que trae el lenguaje de alto nivel, habrá que crear una nueva rutina de inicialización.

Para enviar los caracteres y comandos, suele hacerse una rutina que divide cada byte por transferir en dos **nibbles** (**formación de 4 bits**) y realiza la transferencia correspondiente, comenzando por el nibble alto y, luego de haber verificado el estado del LCD, siguiendo con el bajo.

En la conexión de 4 bits las transferencias se duplican, puesto que para enviar cada dato, hay que dividirlo.



> VENTAJA DE LA CONEXIÓN EN 4 BITS

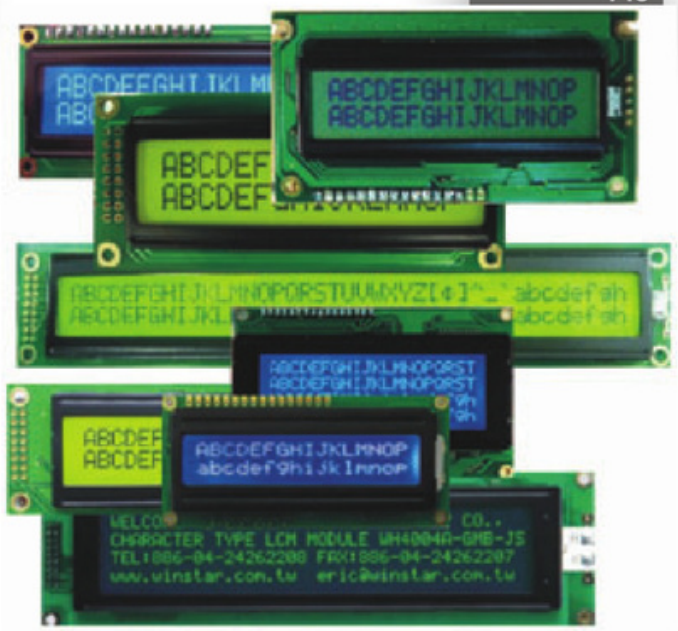
La conexión del display en 4 bits es muy usada en los microcontroladores que poseen poca cantidad de puertos. Esto nos permite tener una mayor libertad para aprovechar los escasos recursos de estos micros. Por ejemplo, el PIC16F84A solo cuenta con 13 puertos I/O, de modo que resulta indispensable ahorrar pines para sacarle el mayor provecho.

FUNCIONES DE LA LIBRERÍA LCD	
Función	**Descripción**
BusyXLCD	Interroga si el bus de datos está desocupado.
OpenXLCD	Inicializa el LCD.
putcXLCD	Escribe un byte en el LCD.
putsXLCD	Escribe un string de la memoria de datos en el LCD.
putrsXLCD	Escribe un string de la memoria de programa en el LCD.
ReadAddrXLCD	Lee la dirección de un byte del LCD.
ReadDataXLCD	Lee un byte del LCD.
SetCGRamAddr	Establece la dirección del generador de caracteres.
SetDDRamAddr	Establece la dirección de la visualización de caracteres.
WriteCmdXLCD	Escribe un comando en el LCD.
WriteDataXLCD	Escribe un byte en el LCD.

> Tabla 3. Listado de funciones de la librería LCD del MPLAB C18, con una breve descripción de cada una.



> Los displays LCD alfanuméricos poseen un controlador basado en el Hitachi HD44780. Es posible utilizar las librerías de los compiladores para controlarlos.



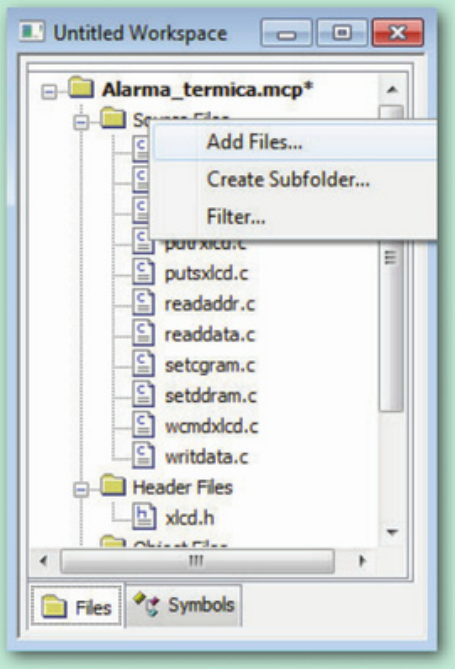
Librería LCD del MPLAB C18

El programa **MPLAB C18** posee su propia librería para manejar LCDs inteligentes, basada en la normalización que impuso el controlador Hitachi HD44780. Esta librería se llama **xlcd** (*External LCD Functions*, funciones para LCD externo) y nos permite modificar su código para definir, por ejemplo, cuál es el puerto del PIC que se conectará al bus de datos del LCD; es decir, si será de 8 o 4 bits y cuáles son los pines del PIC que actuarán para el control del LCD (conexión de los pines **RW**, **RS** y **E** del LCD al PIC). Todas estas configuraciones deben efectuarse desde el archivo de cabecera **xlcd.h** de la librería.
Más adelante veremos la manera de realizarlas.

Funciones de la librería xlcd

Esta librería posee un repertorio de funciones bastante amplio, que nos permite realizar diversas tareas con el LCD, desde enviar un comando, hasta escribir mensajes enteros en la pantalla. En la **Tabla 3** podemos observar las diferentes funciones que ofrece, y a continuación describimos las más utilizadas.

- **OpenXLCD**: inicializa el LCD, al cual debemos indicarle la cantidad de bits del bus de datos y el tipo de display que utilizaremos. También debemos señalar si es de una o de múltiples líneas, y el tamaño de la matriz para cada carácter (5 x 7 o 5 x 10).
- **WriteCmdXLCD**: con esta función enviamos comandos de control al LCD.
- **WriteDataXLCD**: envía caracteres al LCD, que deben ser de tipo **char de 8 bits**.
- **BusyXLCD**: interroga si el bus de datos se encuentra libre para poder enviar otro dato.
- **putsXLCD**: escribe una cadena de caracteres al LCD que se encuentra almacenada en la memoria de datos.
- **putrsXLCD**: escribe una cadena de caracteres al LCD que se encuentra almacenada en la memoria de programa.



> En la ventana Projects agregamos los códigos fuente de las funciones correspondientes a la librería del LCD.

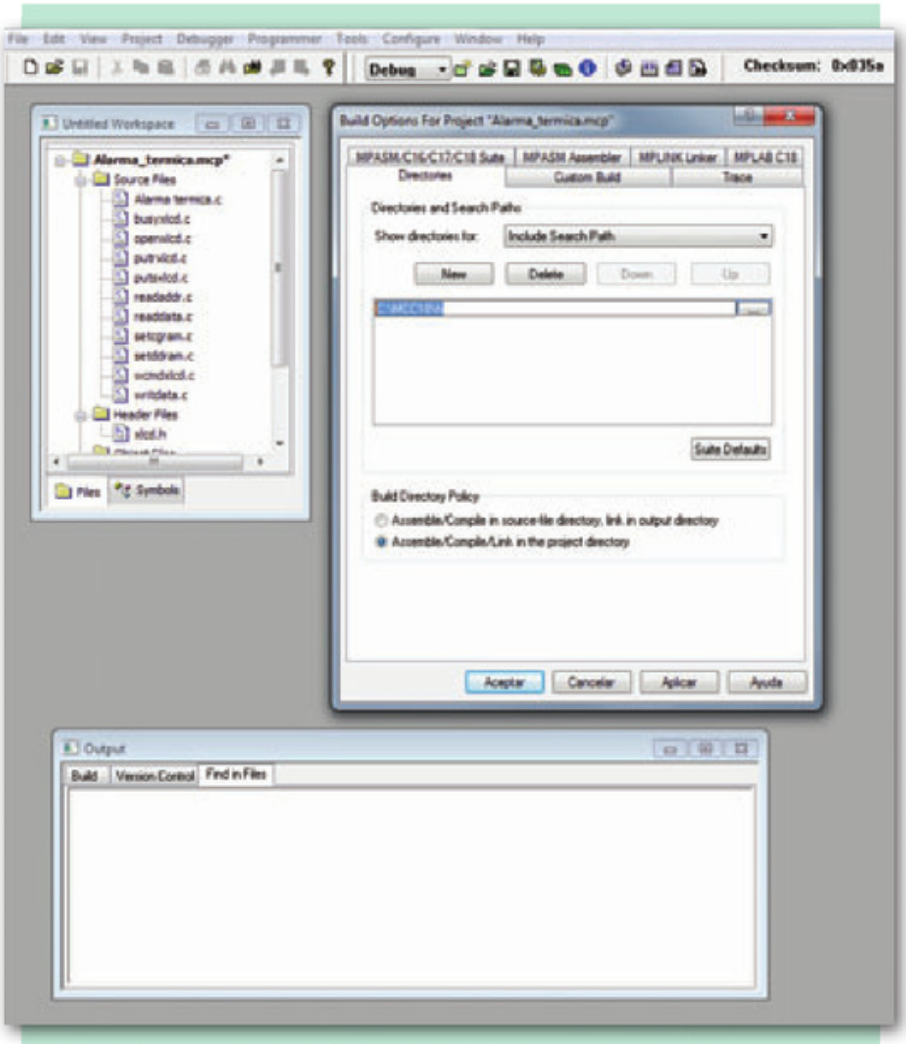
Agregar la librería xlcd a nuestro proyecto

Para utilizar las funciones de esta librería, primero tenemos que añadirla a nuestro proyecto, realizando los siguientes pasos. En el entorno del MPLAB, vamos a **Projects/Source Files/Add Files** y agregamos todos los archivos fuente de la carpeta **XLCD** a nuestro proyecto. Estos son los códigos fuente de las funciones de la librería. Nos dirigimos a la ventana **Projects**, hacemos clic con el botón derecho del mouse en la carpeta **Header Files**, seleccionamos **Add Files** y agregamos el archivo **xlcd.h** del proyecto. A continuación tenemos que configurar el compilador. Para hacerlo, le indicamos dónde buscar el archivo **xlcd.h**. Esto se hace desde la barra de menús **Project/Build Options/Projects**. En la solapa **Directories** elegimos **Include Search Path**, hacemos clic en **New** y colocamos la dirección "C:\MCC18\h".

Cuando trabajamos con la librería del LCD, debemos modificarla para adaptarla a la conexión de nuestra placa.

Modificar la librería del LCD

Cuando trabajamos con la librería del LCD, debemos modificarla para adaptarla a la conexión de nuestra placa. Por defecto, la librería define la línea de datos en el puerto **B** de los PICs, y los pines de control **E**, **RS** y **RW** en los pines **RB4**, **RB5** y **RB6**, respectivamente. Por ejemplo, si queremos colocar el LCD en el puerto D y las líneas de control en el E, escribimos lo siguiente en el archivo xlcd.h: ■



> Debemos configurar el compilador indicando la dirección donde se encuentra el archivo de cabecera xlcd.h.

```
Para la conexión del bus de datos:
#define DATA_PORT  PORTD
#define TRIS_DATA_PORT TRISD

Para la conexión de las líneas de control:
#define RW_PIN LATEbits.LATE0 /* PORT for RW */
#define TRIS_RW TRISEbits.TRISE0 /* TRIS for RW */
#define RS_PIN LATEbits.LATE1 /* PORT for RS */
#define TRIS_RS TRISEbits.TRISE1 /* TRIS for RS */
#define E_PIN LATEbits.LATE2 /* PORT for E */
#define TRIS_E TRISEbits.TRISE2 /* TRIS for E */
```

TECNOLOGÍAS DE DISPLAYS LCD

Existen diversos tipos de tecnologías constructivas de displays LCD, tanto alfanuméricos como gráficos. Veremos algunos detalles de ellos.

Los displays LCD (*Liquid Cristal Display*) son dispositivos que **no emiten luz**, sino que deben utilizar la luminosidad disponible en el entorno. Por este motivo, es posible tener displays de muy bajo consumo, como los de las calculadoras, relojes de pulsera, y otras aplicaciones, donde cada microampere cuenta.

El cristal líquido es una sustancia orgánica que presenta una estructura cristalina como los sólidos, pero es líquida. Las moléculas de este material tienen la propiedad de **orientarse** con la aplicación de un **campo eléctrico**, y es este el principio de funcionamiento del display.

Tecnologías constructivas

Un display LCD se construye sobre la base de dos placas de vidrio (o equivalente) transparente, con el cristal líquido en el medio. Se colocan dos **polarizadores** alineados, uno en cada placa transparente. Cuando no hay campo eléctrico aplicado, la luz pasa a través del display y vemos el fondo. Al aplicar un campo eléctrico, las moléculas del cristal líquido se orientan y **polarizan** la luz en un sentido. La luz así polarizada no atraviesa los polarizadores y, entonces, vemos un punto oscuro. La versión más simple de esta tecnología se denomina **TN** (*Twisted Nematics*); las otras son:

- **STN** (*Super-Twisted Nematics*): el material utilizado produce un ángulo de rotación (polarizado) mucho mayor, cercano a los **240º** (en **TN** se acerca a los **90º**). Entonces, el efecto de bloqueo es mucho más pronunciado, y podemos obtener mayor contraste. Un defecto conocido en esta tecnología se debe al fenómeno de **birrefringencia**, por el cual se produce un desplazamiento del color y, entonces, se ven el fondo amarillo-verdoso y los puntos azulados.

> Un cristal líquido es una estructura cristalina (como un sólido), pero en estado líquido.

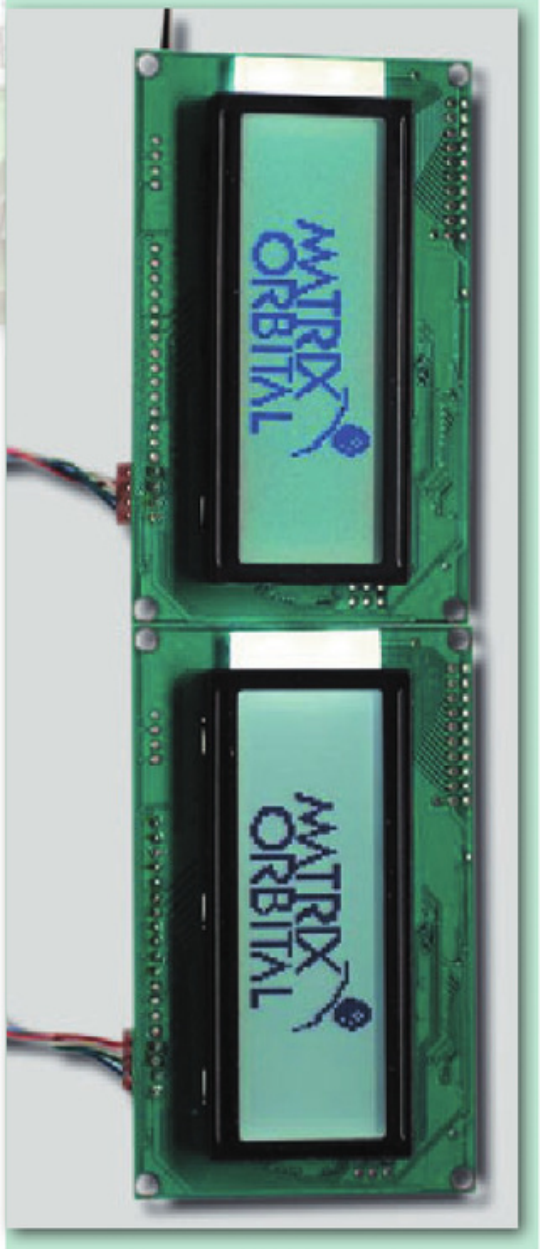




> TFT O MATRIZ ACTIVA

Las tecnologías detalladas en esta página se aplican a las denominadas "matrices pasivas". En displays color, se utilizan matrices activas TFT (*Thin-Film Transistor*) para controlar de forma más localizada la polarización en cada punto, y mejorar el contraste y el ángulo de visión.





> **Vemos aquí el tinte azulado que introduce el STN (izquierda) y la compensación de FSTN (derecha).**

- **HTN** (*High-performance TN*): es una tecnología intermedia entre **TN** y **STN**, con ángulos de rotación del orden de los **180º**. Presenta las ventajas de tener mayor contraste que **TN** y ser más económica que **STN**, sin poseer efecto de birrefringencia.
- **FSTN** (*Film-Compensated STN*): para compensar el defecto de **STN**, se agrega un film compensador que produce un giro opuesto al producido por el fenómeno de birrefringencia. Esto permite obtener displays con puntos de color negro y fondo de color blanco, con gran contraste. Sin embargo, el film introduce una pérdida de luminosidad, por lo que suele utilizarse con iluminación posterior (*backlight*).

Ángulo de visión

Es la dirección desde donde el display se ve mejor, y esto se debe al posicionamiento de los polarizadores. Se especifican como posiciones en un reloj: por ejemplo **6:00** indica un display que se ve mejor en dirección de abajo a arriba, como si estuviéramos parados viendo un reloj en línea a las **6**; es decir, con el equipo por encima de nuestra línea de visión.

Un display 12:00 está diseñado para ser visto con el equipo por debajo de la línea de visión.

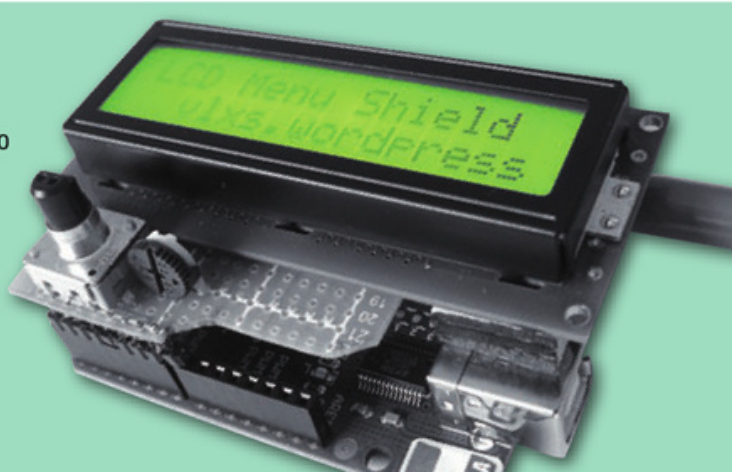
Tipos o modos de displays

Existen, fundamentalmente, tres tipos constructivos, según su parte posterior:

- **Reflectivos**: poseen un reflector en su parte trasera. Esto refleja la luz ambiente y, entonces, permiten ver solo durante el día.
- **Transmisivos**: no poseen reflector, sino un iluminador en su parte posterior. Esto hace que lo que se vea sea la luz posterior, y los vuelve ideales para ambientes oscuros. En general, se los utiliza con polarizadores invertidos, de modo que la imagen es un negativo. Es decir, el punto que se polariza deja pasar la luz y, por lo tanto, vemos un punto iluminado. El iluminador debe estar siempre encendido.
- **Transflectivos**: son una combinación de los dos anteriores. El display se ve de manera excelente a la luz del día, y utiliza el iluminador posterior para mejorar la luminosidad y usarse de noche. El elemento posterior se denomina **transflector**, ya que refleja la luz que incide por una cara y deja pasar la que incide por la opuesta. El iluminador puede encenderse ante determinadas condiciones, a voluntad.

> TN O STN NEGATIVO

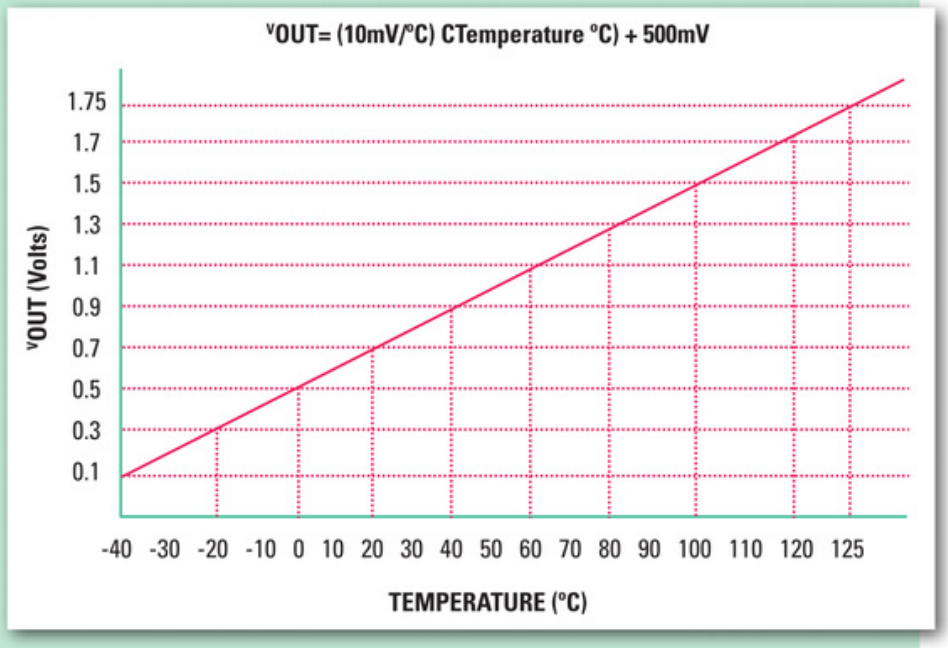
Por TN negativo (TN-) se conoce a un display de tipo transmisivo de tecnología TN, cuyos polarizadores están en contrafase. Con las moléculas del cristal líquido sin orientar, la luz es frenada por ambos polarizadores. Al orientarse las moléculas, la polarización compensa la del primer polarizador y, entonces, vemos el fondo, que posee un iluminador.



PROYECTO DE ALARMA TÉRMICA

Con los conocimientos obtenidos hasta el momento, estamos en condiciones de construir una alarma térmica para indicar la temperatura en el LCD.

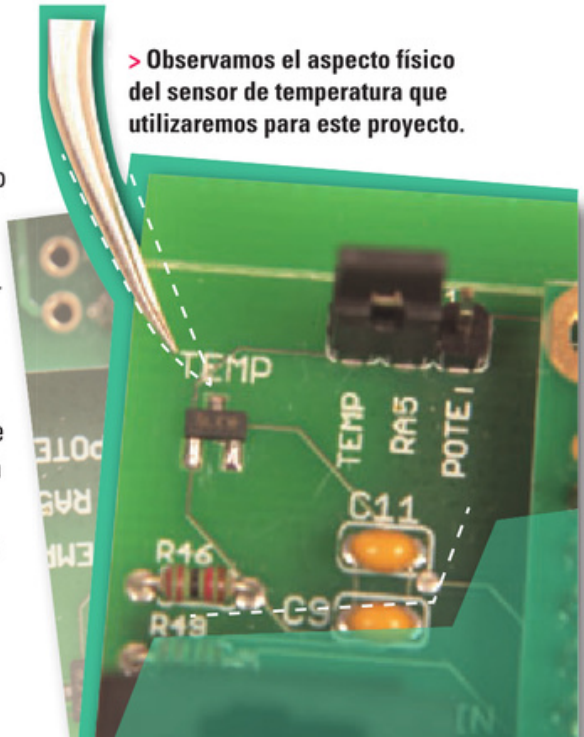
Esta alarma se ocupará de monitorear la temperatura de un recinto cerrado, que puede ser una habitación, el gabinete de una PC o un horno industrial. Este valor debe ser visualizado en todo momento a través de una pantalla LCD, donde tiene que figurar con un dígito decimal de precisión.
A su vez, la alarma debe tener, predefinida por el programador una temperatura crítica, de modo que si el recinto excede ese valor, se active un relay que controle un sistema de refrigeración, como un ventilador. También podemos definir que el programador se encuentre conectado a una bocina, para indicar que la temperatura está por encima del valor crítico. Además, si se acerca, por ejemplo, a 5° C (umbral de la temperatura crítica), la alarma tendrá que indicarnos sobre esta situación mediante un mensaje de atención en la pantalla LCD.



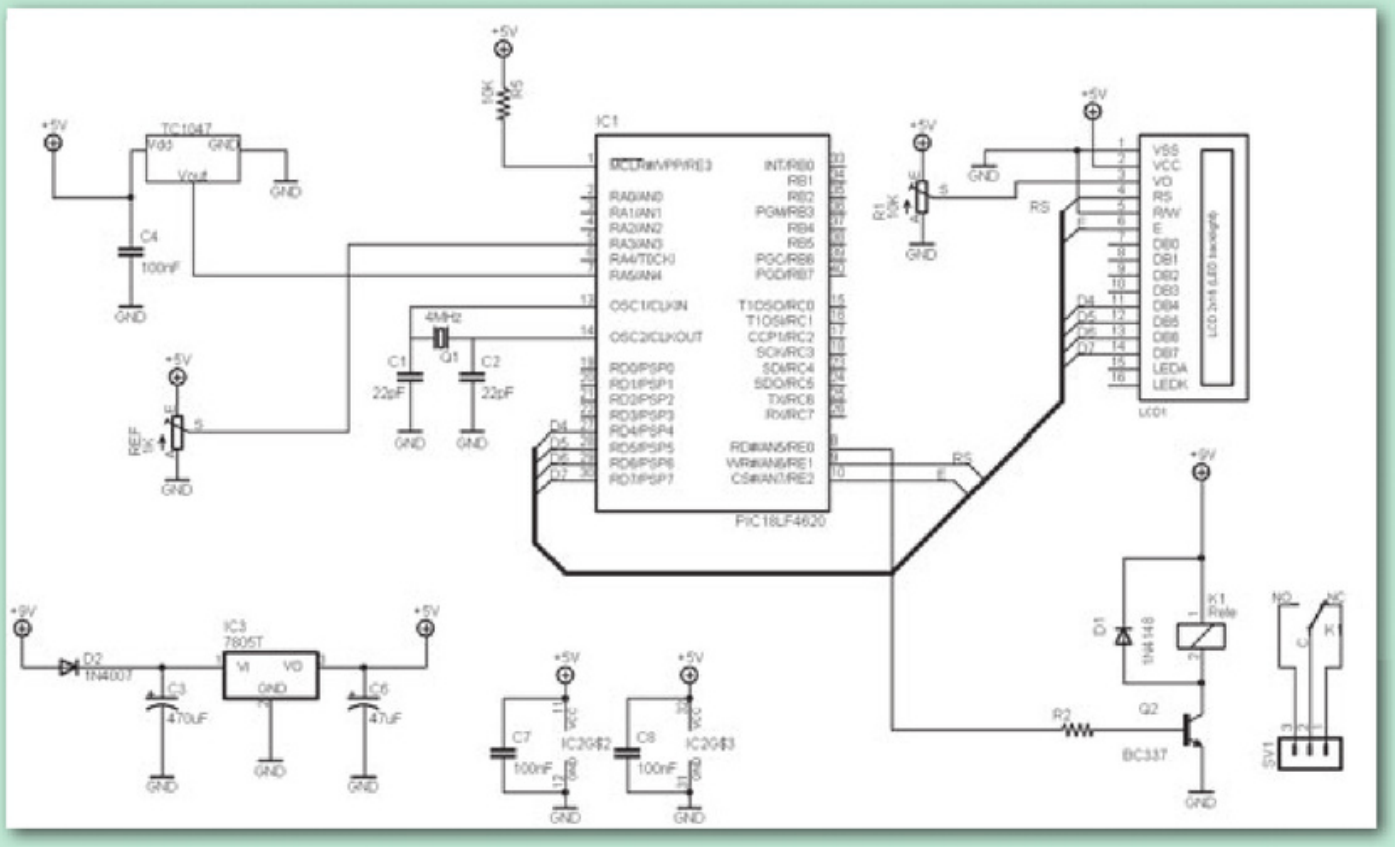
> **Figura 24. Esta es la gráfica que relaciona la temperatura leída por el sensor y la tensión de salida. Podemos notar que se comporta de manera lineal en el rango de temperatura especificado por el fabricante.**

Sensor de temperatura

El sensor de temperatura que utilizaremos para este proyecto es el **TC1047** de Microchip, que convierte la temperatura en un nivel de tensión bastante preciso.
Es de tres pines solamente y puede ser alimentado con **2,5 V** hasta **5,5 V**. Tiene un encapsulado **SMD**, lo cual lo hace ideal para medir temperaturas en recintos muy pequeños, como en teléfonos celulares. El rango de temperatura de medición es de **-40° C** a **125° C**, como se observa en la **Figura 24**, en donde la tensión de salida se comporta de manera lineal. La pendiente de la recta es de **10 mV/°C**, es decir que por cada grado que se modifique la temperatura, la tensión variará unos **10 mV**.
Podemos obtener el valor de la temperatura sensada por el dispositivo si sabemos cuál es la tensión de salida (Vout). Solo debemos despejar la temperatura de la ecuación de la recta que se muestra en la **Figura 24: Temperatura (°C) = (Vout (Volts) – 500 mV) / 10 mV/°C**; donde **Vout** es la tensión de salida que mediremos con el conversor, y **10 mV/°C**, la pendiente de la recta.



> **Observamos el aspecto físico del sensor de temperatura que utilizaremos para este proyecto.**



> **Figura 25. En la imagen vemos el esquemático del proyecto.**

Esquemático

En la **Figura 25** observamos el esquemático del proyecto con el que vamos a trabajar. La alimentación estará dada por un transformador de **9 Volts** a la entrada. Un circuito regulador con el **7805** se encargará de controlar y estabilizar la tensión a unos **5 V**. El centro de nuestro proyecto es el **PIC18LF4620**, al cual se le acoplarán los distintos periféricos elegidos para este desarrollo. A un costado vemos el sensor de temperatura, donde solo tenemos que conectar la alimentación y el pin con la tensión de salida al pin **RA5** (canal 4 del conversor) del PIC. En el pin **RA3** del PIC colocamos un potenciómetro para calibrar la tensión de referencia del conversor A/D y lo llamamos **potenciómetro de referencia**. La pantalla LCD está conectada con un bus de 4 bits al **puerto D** del PIC, y los pines de control E y RS del LCD, a los pines RE2 y RE1 del PIC, respectivamente. La línea R/W del LCD se conecta a masa. Solo vamos a escribir en el LCD, nunca leeremos un dato de él. Luego, la salida de nuestra alarma térmica es un relay controlado por el pin **RE0** del microcontrolador. Allí podemos conectar el control de un sistema de refrigeración o un simple ventilador para regular la temperatura del recinto, o agregar una bocina que nos indique el exceso de temperatura de manera sonora. Cada usuario puede expandir una placa experimental para este proyecto, aplicando todo lo aprendido hasta el momento en este libro. También existe la alternativa de utilizar una placa de entrenamiento comercial para simular el ejercicio.

El sensor de temperatura que utilizaremos es el TC1047, que resulta bastante preciso en la conversión.

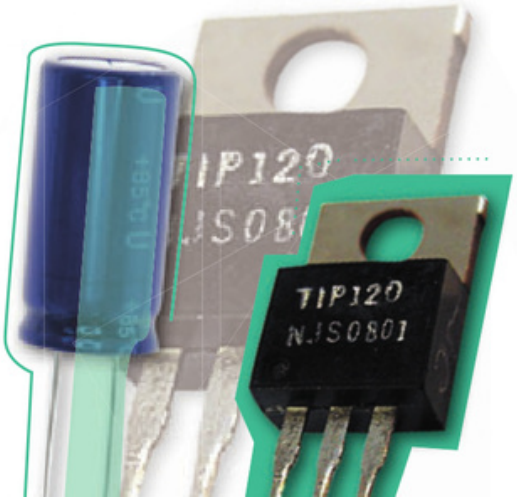
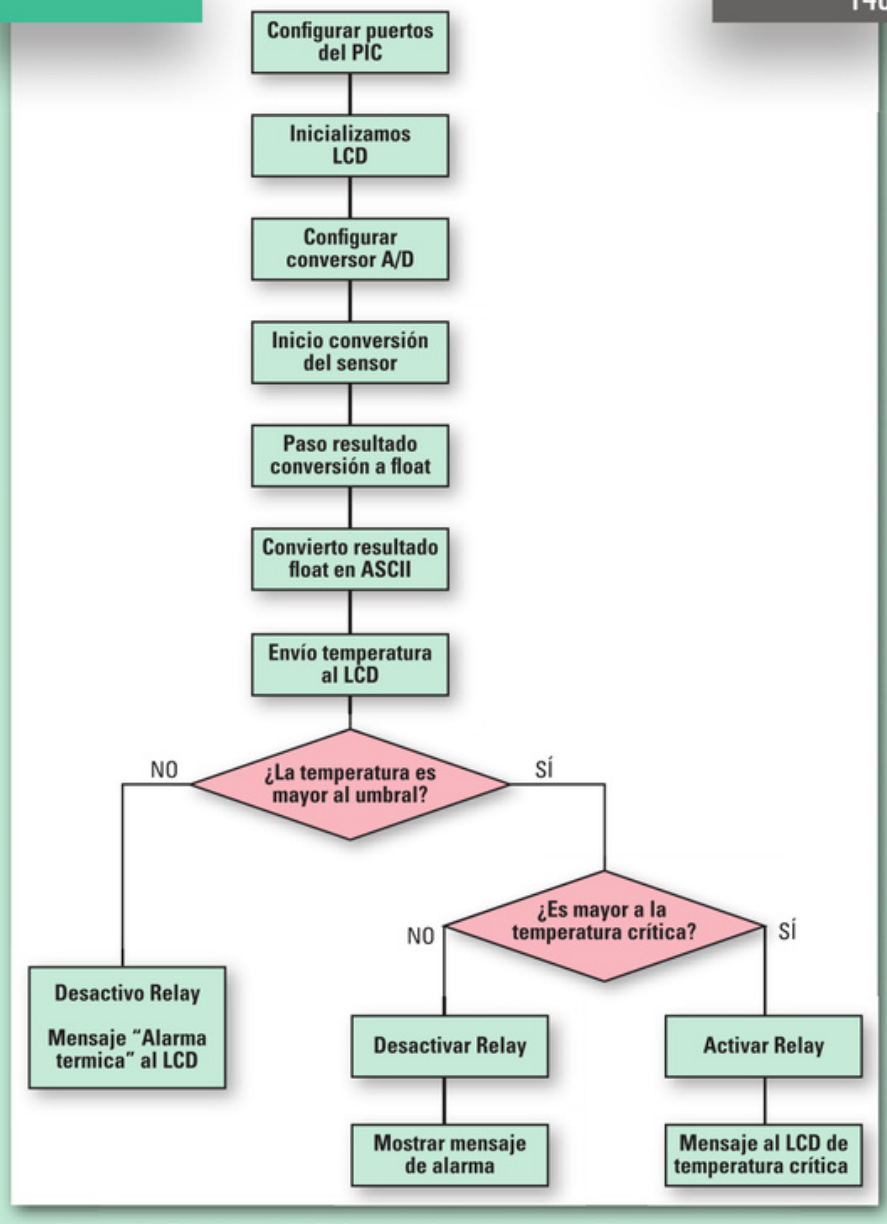
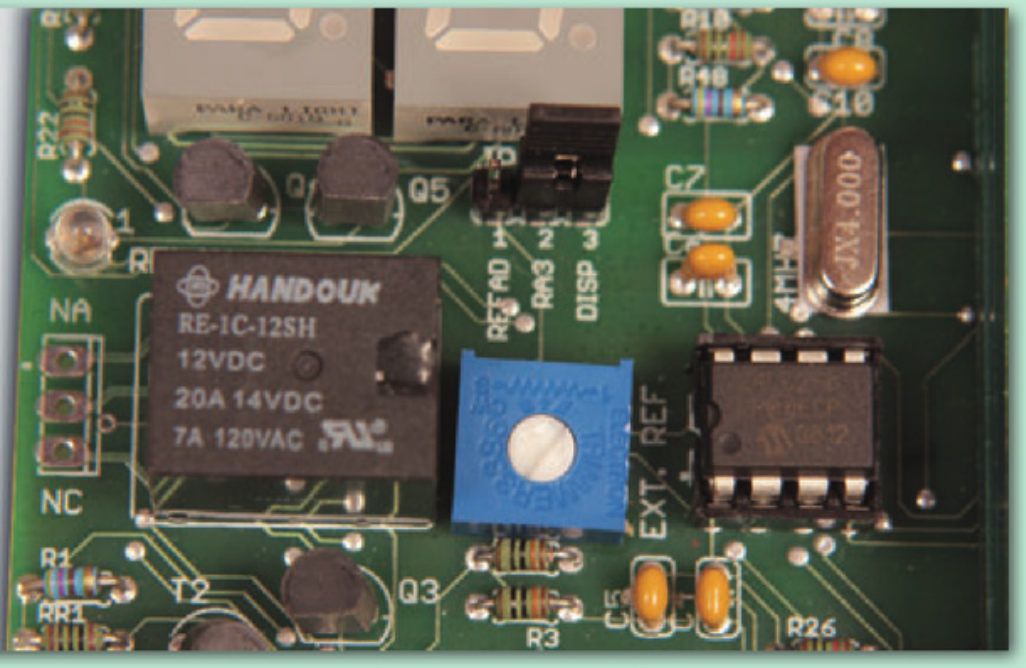


Diagrama de flujo

En la **Figura 26** se presenta el diagrama de flujo de este proyecto, que tomaremos como guía para programar el PIC. Para manejar el LCD, en esta práctica vamos a emplear las funciones de la librería de **MPAB C18**. Luego, continuamos con la configuración de los registros del conversor **A/D** del PIC para efectuar la conversión durante el programa.
Cuando terminamos de configurar todos los recursos del PIC, comenzamos el programa realizando la conversión analógica-digital de la entrada analógica, donde está conectado el sensor de temperatura.
Una vez que tenemos el valor digital de la conversión, calculamos el valor de la temperatura mediante la ecuación descripta en la página anterior, con lo cual obtendremos un valor que es almacenado en una variable **float**.
Si queremos visualizarlo en el LCD alfanumérico, debemos convertir ese valor en caracteres ASCII, para, luego sí, enviarlo a la pantalla.
Ahora, si la temperatura del recinto está por debajo de un cierto umbral crítico, no hacemos nada; pero si está dentro del rango del umbral, lo indicamos en la pantalla mediante un mensaje de atención. Cuando la temperatura del recinto supera el valor crítico, se activará el relay, lo cual se indica en el display LCD.



> **Figura 26. Diagrama de flujo completo correspondiente al proyecto de la alarma térmica.**



> **Figura 27. Este es el relay conectado al pin RE0 que utilizaremos como salida para controlar algún sistema de refrigeración o alguna bocina que funcione como alarma.**

Configuraciones del PIC

Los primeros bloques del diagrama de flujo se encargan de configurar los recursos del PIC. Para configurar los puertos, debemos identificar cuáles se utilizarán como entradas digitales, cuáles como salidas digitales y cuáles serán las entradas analógicas.
En el esquemático podemos ver que no contamos con entradas digitales, y que tenemos una salida digital en RE0, que controla el relay. No debemos preocuparnos por los puertos que maneja el LCD, porque la librería del MPLAB C18 se encarga de configurarlos.

Luego tenemos la configuración de la parte analógica del PIC, en la cual seteamos el **conversor A/D** y la entrada del canal donde se encuentra el sensor. Vamos a configurar el módulo A/D para tomar la señal analógica del canal 4 (RA5), realizar la conversión y hacer que la tensión de referencia sea el nivel de tensión en que se encuentre el pin **RA3/Vref**.
Es por eso que allí colocamos un potenciómetro que nos permita calibrar dicha referencia.
Para configurar este módulo, utilizamos tres registros: **ADCON0**, **ADCON1** y **ADCON2**. Con **ADCON0** seleccionamos el canal de entrada para la conversión –en este caso, el 4 (RA5)– y habilitamos el módulo para poder utilizarlo. Con **ADCON1** configuramos los primeros cinco canales del PIC (AN0, AN1, AN2, AN3 y AN4), con el fin de usarlos como entradas analógicas (solo utilizaremos el 4). El nivel de referencia positivo será la tensión en el pin **RA3/Vref**, en tanto que la otra referencia será **GND**. Con **ADCON2** elegimos una justificación a izquierda del resultado (para que los 8 bits más significativos de la conversión queden en el registro ADRESH) y el tiempo de adquisición más alto.

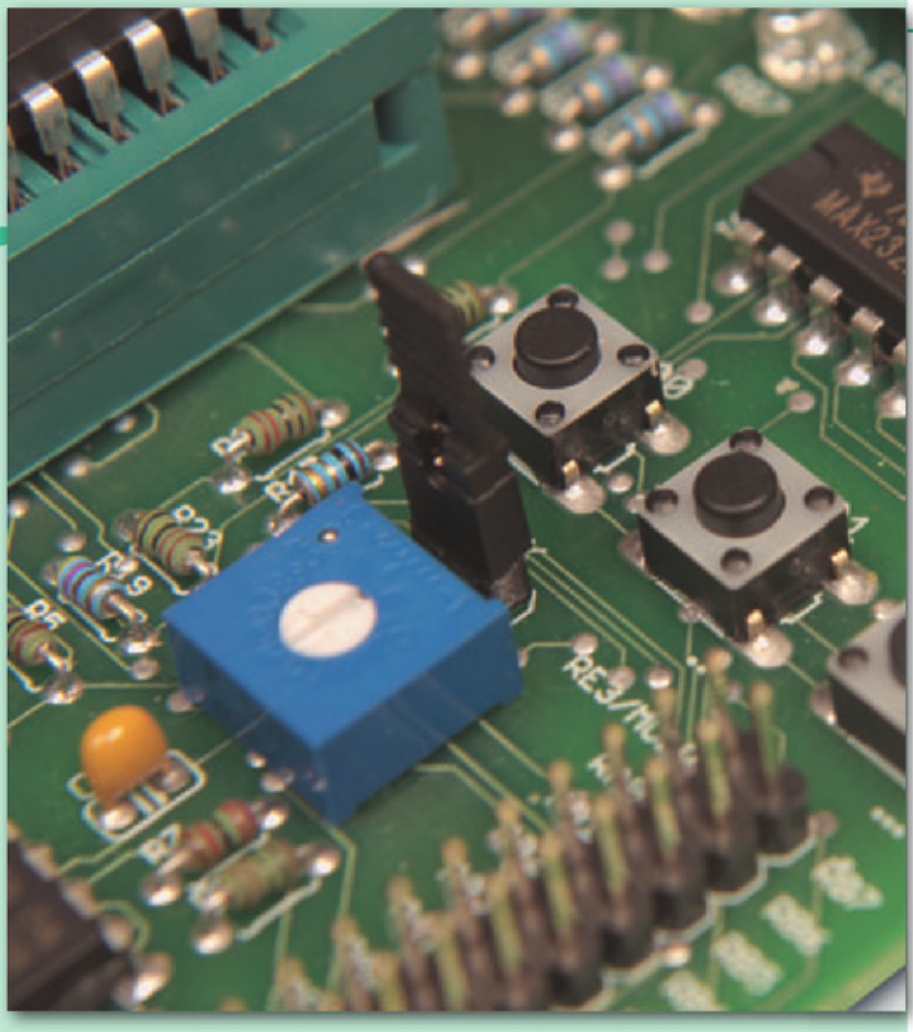
El nivel de referencia positivo del conversor será la tensión en el pin RA3/Vref, en tanto que la otra referencia será GND.

Configuración del LCD

Para usar el LCD modificamos la disposición de los pines que se conectarán a él, que están definidos en el archivo xlcd.h. Si utilizamos el esquemático de la **Figura 11**, los cambiamos tal como se mostró en páginas anteriores. Otro tema es la conexión del pin R/W: si lo colocamos a masa como en el esquemático, debemos modificar la función BusyXLCD para que solo genere un retardo de 1 mseg, en vez de realizar el chequeo del busy flag. Esto lo hacemos así porque, al no estar conectado el R/W, no podemos leer datos del LCD; siempre vamos a escribir, por lo cual no podemos leer el estado del pin D7 para saber si el bus de datos está desocupado. ■

> EL BUSY FLAG

Existen dos modos de operar con el LCD: con busy flag y sin él. En el primer caso, debemos leer el bit D7 del LCD antes de enviar un carácter o comando. Si es **1**, el LCD está ocupado, mientras que si es **0**, significa que está libre. Para no tener que realizar la lectura del busy flag, podemos aplicar un retardo antes de enviar un carácter lo suficientemente largo, de manera de asegurar que el bus esté libre. En ese caso, podemos conectar R/W del LCD a masa.



> Con el potenciómetro podemos emular una entrada analógica al conveyor A/D del PIC.

PROYECTO ALARMA TÉRMICA EN MPLAB C18

En este paso a paso convertiremos en código toda la teoría vista para armar este proyecto.

Estamos preparados para pasar todo lo que estuvimos estudiando sobre este proyecto a código fuente. Ya tenemos en claro cuál es la tarea que debe realizar la alarma térmica; tenemos el diagrama de flujo y el esquemático, y sabemos cuáles son los periféricos con los que el microcontrolador tendrá que interactuar. Ahora, volquemos toda esta teoría al código fuente. ■

01. En el encabezado del código incluimos todos los archivos que necesitaremos en el programa. Algunos contienen funciones útiles, como stdlib.h e itoa. Luego, definimos algunas constantes y variables globales para almacenar el valor de la temperatura, tanto en float como en ASCII. Colocamos los prototipos de las funciones que definimos en el código.

02. Entramos en el programa principal y realizamos el código según lo indica el diagrama de flujo, inicializando los puertos que utilizamos, el LCD y la función ADC_Init(); para configurar el conversor. Luego, dentro del bucle while, la función ADC_Convert(); realiza la conversión A/D, y el resultado se lo asigna a Conversion, para después calcular la temperatura.

03. Una vez que tenemos la temperatura, la función Temp_LCD(); la toma, la convierte en ASCII y la envía al LCD. Luego están los bloques condicionales, que actúan según el diagrama de flujo y, dependiendo del valor de la temperatura, envían un mensaje al LCD para indicar el estado y controlar la temperatura crítica definida en la cabecera.

01

```
/* Includes */
#include "p18f4620.h"
#include "delays.h"
#include "xlcd.h"
#include "stdlib.h" //Libreria que contiene la fun

/* Definiciones */
#define Vref            1.75
#define Decimales_precision     10
#define Temp_critica    35
#define Umbral          5

/* Variables */
#pragma udata
unsigned char Conversion;
float Temp;
char Temperatura[3]; // Cadena de String
float Vout;
int Entera;
int Decimas;

/* Prototipos */
void ADC_Init(void);
unsigned char ADC_Convert(void);
void Temp_LCD(float fTemp);

/* Programa Principal ***/

void main(void)
{
    Temp = 0;   // Inicializo

    TRISEbits.TRISE0 = 0;   // RE0 salida

    OpenXLCD( FOUR_BIT & LINES_5X7); //Inicializam
    // Configuramos LCD con 4 bits de datos de var
    // con matris de caracter de 5x7

    putrsXLCD(" Alarma Termica  "); // Enviamos me

    // Inicializamos conversor ADC
    ADC_Init();
```

02

```
/* Programa Principal ***/

void main(void)
{
    Temp = 0;   // Inicializo

    TRISEbits.TRISE0 = 0;   // RE0 salida

    OpenXLCD( FOUR_BIT & LINES_5X7); //Inicializamos LCD
    // Configuramos LCD con 4 bits de datos de varias lineas
    // con matris de caracter de 5x7

    putrsXLCD(" Alarma Termica  "); // Enviamos mensaje al LCD

    // Inicializamos conversor ADC
    ADC_Init();

    while(1)
    {
        // comenzamos convirtiendo el valor leido en el conversor
        // en grados dentro de una variable del tipo float
        Conversion = ADC_Convert();
        Vout = (float)(Conversion*Vref)/255;
        // obtengo temperatura en una variable float
        Temp=(float)(Vout - 0.5)/ 0.01;

        // La siguiente rutina convierte la variable float
```

03

```
        // La siguiente rutina convierte la variable float
        // en ASCII y lo envia al LCD
        Temp_LCD(Temp);

        // Temp es mayor al umbral???
        if (Temp >= (Temp_critica - Umbral))
        {   // si, entonces preguntamos
            // Temp es mayor a la critica???
            if (Temp >= Temp_critica)
            {   // Si
                LATEbits.LATE0 = 1; // Activo Rele
                // Comando para colocarnos en la primera
                // linea del LCD
                WriteCmdXLCD(LINE_0);
                putrsXLCD(" Rele activado  ");
            }else
            {   // Temp es menor a la Temp_critica
                LATEbits.LATE0 = 0; // Desactivo Rele
                // Comando para colocarnos en la primera
                // linea del LCD
                WriteCmdXLCD(LINE_0);
                putrsXLCD(" ;;Atencion!!   ");
            }
        }else
        {   // Temperatura es menor al umbral
            LATEbits.LATE0 = 0; // Desactivo Rele
```

04. Estos son los retardos que debemos definir para que la librería del LCD del MPLAB C18 pueda funcionar. Fueron realizados con las funciones de la librería Delays, que generan retardos de 100 ciclos de máquina o 1000 ciclos de máquina, como en este caso. Si el cristal externo es de 4 MHz, cada ciclo de máquina equivale a 1 useg.

```
// Retardos que debemos definir para la libreria LCD

void DelayFor18TCY( void )
{
  Delay100TCYx(1);  // Retardo de 100uSeg
}

void DelayPORXLCD(void)
{
    Delay1KTCYx(15);     // Delay de 15mS
                         // ciclos = (TimeDelay * Fosc)/4
                         // ciclos = (15mS * 4Mhz)/4
                         // ciclos = 15000
}

void DelayXLCD(void)
{
    Delay1KTCYx(5);      // Delay de 5mS
                         // ciclos = (TimeDelay * Fosc)/4
                         // ciclos = (5mS * 4Mhz)/4
                         // ciclos = 5000
}
```

04

05. La función ADC_Init(); configura el conversor como lo pedimos en páginas anteriores. Observemos que modifica los valores de los registros ADCON por configurar. Por su parte, ADC_Convert() realiza la conversión; primero activa el módulo poniendo a 1 el bit GO_DONE de ADCON0, y espera a que se termine. Luego, devuelve el resultado del registro ADRESH.

```
void ADC_Init(void)
{ // Inicializmos el conversor Analogico-Digital
    // Comenzamos configurando cuales van a ser los pines analogicos
    // y digitales.

    /*
    La siguiente configuracion coloca la referencia en VREF+ (AN3)
    para poder colocar la referencia con el pote a 1,79V
    y configura los 4 primeros canales como analogicos
    */
    ADCON1 = 0b00011010;

    /* Modificando las referencias */
    //Las tensiones de referencia seran VDD y VSS
    // y configuramos los 4 primeros canales como analogicos
    //ADCON1 = 0b00001010;

    // Justificamos a izquierda, elegimos un tiempo de adquisicion
    // de 20TAD y FOSC/2
    ADCON2 = 0b00111000;

    // Seleccionamos canal 4(RA5), allí encontramos el sensor
    ADCON0 = 0b00010001;
}

unsigned char ADC_Convert(void)
{ // funcion que comienza la conversion y devuelve el resultado
    ADCON0bits.GO_DONE = 1;          // Comienzo la conversion
    while (ADCON0bits.GO_DONE == 1);  // Espero a que se complete
    return ADRESH;                   // retorno la parte alta del resultado
}
```

05

06. Esta función toma el valor de la temperatura en float, lo convierte en ASCII y lo manda al LCD, del siguiente modo: primero, toma la parte entera de fTemp (float) y la convierte en ASCII con la función itoa, lo manda al LCD junto con el punto decimal y, después, despeja los decimales en una variable entera, lo convierte y lo envía al LCD.

07. Una vez finalizado el código fuente, creamos el proyecto "Alarma térmica" en el MPLAB, desde Project Wizard. En él tenemos que incluir los códigos fuente de la librería del LCD y el archivo "xlcd.h" que modificamos, porque si lo compilamos sin tener estos archivos, surgirán errores. El código ya está listo para simularlo y grabarlo en el PIC.

```
void Temp_LCD(float fTemp)
{   // Para poder mostrarlo en el LCD se debe pasarlo a ASCII
    // Lo que realizamos es una conversion de float a int para
    // quedarnos primero con la parte entera
    Entero = (int)fTemp;
    // Enviamos ese valor al LCD
    WriteCmdXLCD(NEXT_LINE + 4);// Nos posicionamos en el LCD
    // en la sugnda fila y en la posicion 4

    itoa(Entero,Temperatura);   // Realizo conversion en ASCII

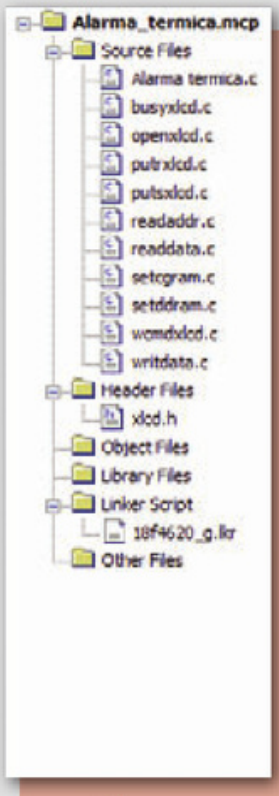
    putsXLCD(Temperatura);      // y lo enviamos al LCD
    // Luego escribimos el punto que separa el entero de las
    // decimas
    WriteDataXLCD('.');

    // y convertimos las decimas en ASCII
    fTemp = (fTemp - (int)fTemp) * Decimales_precision;
    Decimas=(int)fTemp;

    itoa(Decimas,Temperatura);
    // Y lo mandamos al LCD y a la USART
    putsXLCD(Temperatura);
    putsXLCD(" grados    ");
    WriteCmdXLCD(CURSOR_OFF & BLINK_OFF); // Apagamos cursor

}
```

06



07

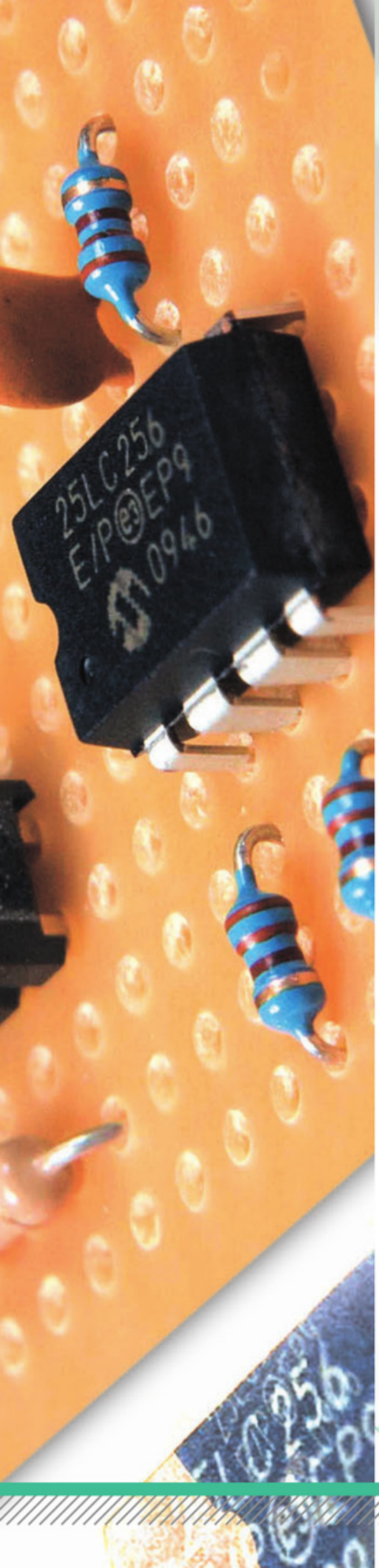
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué características posee la familia PIC18F?
2. ¿Para qué sirve el oscilador interno de los PICF16?
3. ¿Qué puertos de entrada y salida posee un PICF16F?
4. Mencione las características de un relay.
5. ¿Qué periféricos incorpora el PIC18LF4620?
6. Describa la función de un timer.
7. ¿Qué son los display LCD inteligentes?
8. Enumere algunos de los comandos para el LCD.
9. Nombre algunas de las funciones existentes en la librería LCD.
10. ¿Qué tecnologías pueden utilizarse en la construcción de LCDs?

EJERCICIOS PRÁCTICOS

1. Instale un compilador de C para PIC.
2. Genere una tabla que contenga los tipos de datos enteros del MPLAB C18.
3. Arme una placa experimental para PIC18LF4620.
4. Ejemplifique los periféricos que incorpora el PIC18LF4620 en un diagrama.
5. Construya una alarma térmica.

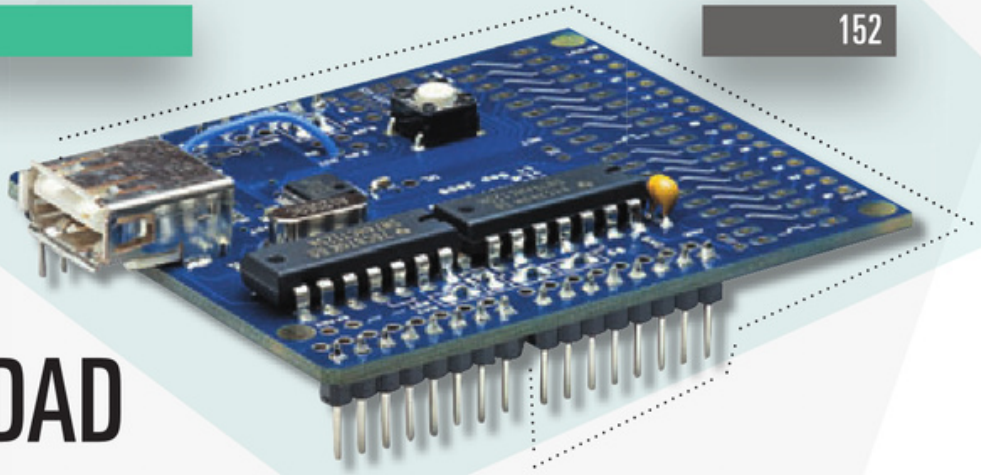


CAPÍTULO 5

CONECTIVIDAD NO INALÁMBRICA

Protocolos de conectividad
Serial Peripheral Interface
Conexión de una memoria SPI
Bus de comunicación I^2C
Conexión y manejo de un periférico I^2C
Docklight
Universal Serial Bus
Actividades

05



PROTOCOLOS DE CONECTIVIDAD

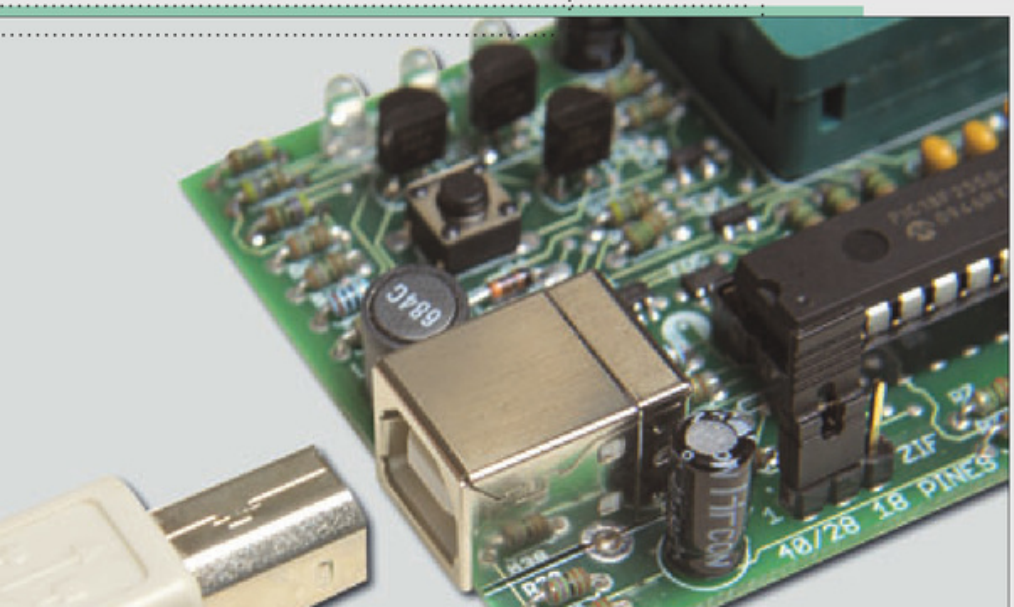
Analicemos qué opciones existen para comunicar un microcontrolador con periféricos de manera no inalámbrica y, por qué no, con una computadora de escritorio.

SPI, microwire, I²C y SMBus son protocolos de interconexión utilizados en comunicaciones de microcontroladores con periféricos, memorias EEPROM, displays LCD u otro tipo de circuitos integrados que requieran una intercomunicación digital. Estas interfaces tienen un formato de transmisión serie y se encuentran, principalmente, en sistemas embebidos.

La computadora personal o PC, omnipresente en nuestra vida, es actualmente la herramienta tecnológica más popular junto con el teléfono móvil. Nos permite realizar todo tipo de tareas y nos ofrece una capacidad de procesamiento de datos cada vez más poderosa. Además, el avance de Internet la ha transformado en un puente de conexión con el mundo. Por todo esto, sería deseable poder establecer una comunicación entre un dispositivo periférico (diseñado por nosotros) y la PC. La **Tabla 1** de la página 153 muestra algunas de las opciones disponibles al respecto. Además de las mencionadas, existen otras interfaces populares. El protocolo MIDI, por ejemplo, se emplea en la conexión de instrumentos musicales electrónicos con la PC.

Conectar un dispositivo a la PC nos permite aprovechar todas las ventajas que esta ofrece.

> **La conectividad USB nos abre un nuevo mundo de posibilidades, y nos permite ampliar las características y funciones de nuestros periféricos.**



Comunicación serie asíncrona y RS-232

Analizaremos la comunicación serie asíncrona, su aplicación en la conectividad de un sistema embebido con una PC y el estándar **RS-232**. Este tipo de comunicación define una manera de transmitir la información de a un bit por vez, en un formato determinado, a una velocidad acordada de antemano por las partes. En general, existen muchos protocolos de comunicación serie que, además, son asíncronos. En particular, aquí estudiaremos lo relacionado con la comunicación

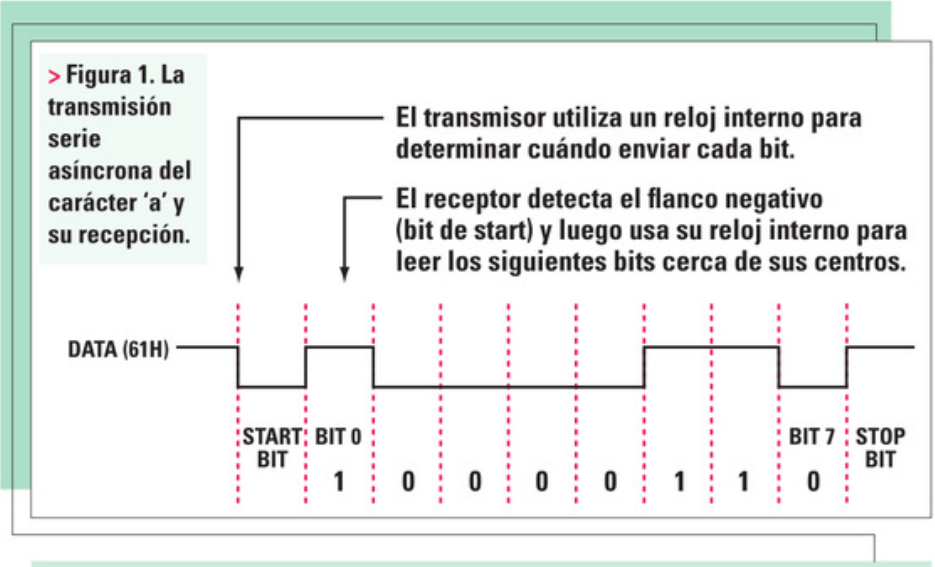
elemental sobre la base de un puerto serie en una **PC** y una **UART** (*Universal Asynchronous Receiver/Transmitter*) en un microcontrolador.

La comunicación

El concepto de comunicación es muy amplio; sin embargo, en este caso nos referiremos a la acción de conectar dos dispositivos electrónicos por un medio físico. En este sentido, la comunicación es bidireccional y, vista desde el microcontrolador, se realiza de la siguiente manera:

- En ausencia de información, la señal se mantiene en el estado lógico alto.
- Un carácter de información se inicia con un bit de **start**, cuya misión es indicar al receptor que la información sigue a continuación.
- El dato de información se envía con el bit menos significativo primero, y puede contener desde 5 hasta 8 bits. En algunos casos, se manda un noveno bit para aplicaciones particulares.
- De manera opcional, pero acordado de antemano, se envía un bit de paridad. Este se calcula emparejando la cantidad de unos en el carácter, para que sea par o impar.
- El carácter se termina con 1 o 2 bits de **stop**, cuya finalidad es volver la línea de transmisión al estado de reposo durante un tiempo suficiente como para que el receptor pueda identificar el próximo bit de **start**. La **Figura 1** muestra el envío del carácter 'a' (0x61) en 8 bits.

Es importante aclarar que **asíncrono** significa que se utiliza una señal de reloj para la transmisión y recepción de datos. Los relojes internos de transmisor y receptor deben estar lo suficientemente próximos y mantener una estabilidad tal, que permita la comunicación de un carácter. Por lo general, el receptor sincroniza su reloj con el flanco descendente del bit de start y observa los datos en el centro de cada bit de información.



> Figura 1. La transmisión serie asíncrona del carácter 'a' y su recepción.

OPCIONES DE CONECTIVIDAD NO INALÁMBRICA

INTERFAZ	FORMATO	NÚMERO MÁXIMO DE DISPOSITIVOS	DISTANCIA MÁXIMA	VELOCIDAD MÁXIMA (BITS/SEGUNDO)	USO TÍPICO
RS-232	Serie asíncrono	2	15 m	19200 a 15 m	Módem, mouse instrumentación
RS-485	Serie asíncrono (half-duplex)	32	1200 m a 100 Kbps	10 M a 12 m	Adquisición de datos y sistemas de control
USB	Serie asíncrono	127	5 m	1,5 M; 12 M; 480 M; 4,8 G (USB 3.0)	Mouse, teclado, memoria, audio, impresora, periférico personalizado
Ethernet	Serie	--	100 m par trenzado	10/100/1000 M	Comunicaciones de redes en general
Puerto paralelo	Paralelo	2	3-9 m	8 M	Impresora, escáner
IEEE-1394 (Firewire)	Serie	64	90 m	3,2 G	Video, audio, dispositivos de almacenamiento
IEEE-488 (GPIB)	Paralelo	15	18m	8 M	Instrumentación

Tabla 1. Existen múltiples opciones de conectividad no inalámbrica con la PC; en la tabla se presentan las más importantes.



> **El estándar RS-232 utiliza un conector tipo DB-25 (de 25 pines) o DB-9 (de 9 pines).**

Puerto serie

El puerto serie es la interfaz en una computadora u otro dispositivo que transmite datos de a un bit por vez. En el uso convencional, el término **puerto serie** se refiere a aquel que utiliza este protocolo **asíncrono** que acabamos de ver. Es bidireccional, ya que puede enviar y recibir información. En muchas ocasiones, podemos pensar que transmitir un bit a la vez es ineficiente para las comunicaciones actuales; sin embargo, este sistema tiene sus ventajas, como la posibilidad de utilizar cables de bajo costo y conectores pequeños.
En una PC, las aplicaciones de software acceden al puerto serie mediante el llamado **COM port**.

SEÑALES RS-232

NOMBRE	PIN (25)	PIN (9)	SENTIDO	FUNCIÓN
TD (Transmit Data)	2	3	DTE→DCE	Datos por transmitir
RD (Received Data)	3	2	DTE←DCE	Datos recibidos
RTS (Request To Send)	4	7	DTE→DCE	El terminal desea transmitir
CTS (Clear To Send)	5	8	DTE←DCE	El terminal puede transmitir
DTR (Data Terminal Ready)	20	4	DTE→DCE	El terminal está operacional
DSR (Data Set Ready)	6	6	DTE←DCE	El módem está operacional
DCD (Data Carrier Detect)	8	1	DTE←DCE	El módem recibe portadora del módem remoto
RI (Ring Indicator)	22	9	DTE←DCE	Se recibe un llamado por la línea telefónica
GND (Ground)	7	5	COMÚN	---

Tabla 2. Estas son solo algunas de las señales más utilizadas en una comunicación asíncrona del estándar RS232.

El estándar RS-232

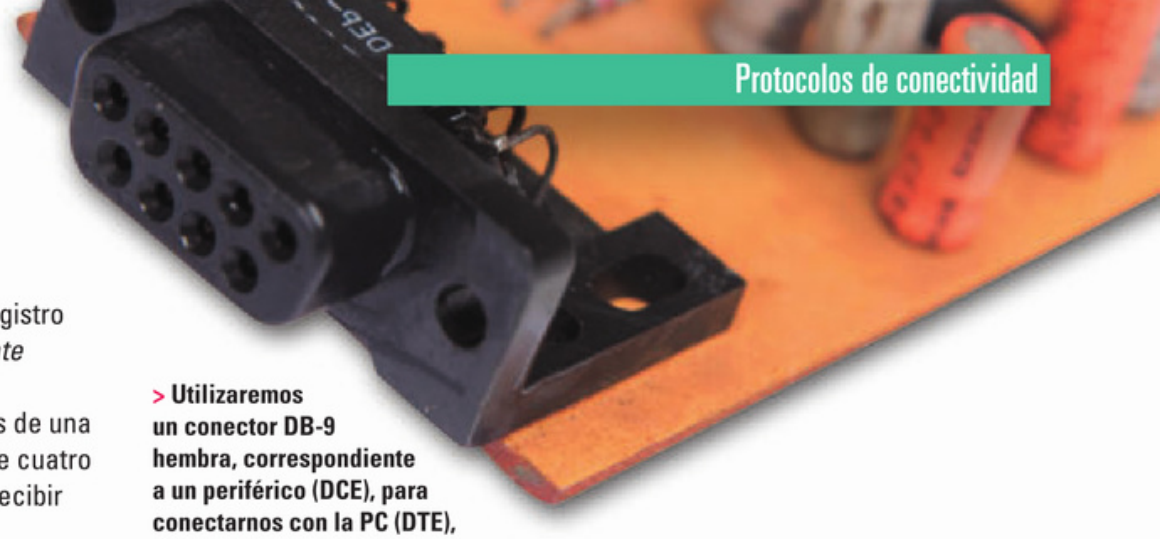
El estándar V.24 de la **ITU-T** (*International Telecommunications Union, Telecommunications Standardization Sector*) define 25 líneas de conexión entre un **DTE** (*Data Terminal Equipment*, una computadora) y un **DCE** (*Data Communications Equipment*, un módem o periférico). El estándar **RS-232** de la EIA/TIA (*Electronics/Telecommunications Industries Association*) determina los niveles de tensión para realizar la comunicación. Existen variantes que fijan el nivel máximo, pero la mayoría coincide en aceptar lo siguiente:

- Un 0 lógico se representa con un nivel de tensión mayor a 3 V.
- Un 1 lógico se representa con un nivel de tensión menor a -3 V.

La distancia máxima es de **15 m**, y la velocidad máxima a esa distancia es de **19200 bps**; velocidades de hasta **115200 bps** o más son posibles a distancias menores. La comunicación es por cable convencional, con referencia de masa (no-balanceado). Las señales más comunes de RS 232 pueden apreciarse en la **Tabla 2**. Existe un modo de conexión muy utilizado, denominado "de 3 cables" (3-wire), que emplea solo TD, RD y GND.

Uso de la UART del PIC18F4620

Este microcontrolador posee un módulo denominado **EUSART**, que es una UART con capacidad de funcionar, además, en modo sincrónico. El módulo se habilita mediante el bit **SPEN** en el registro **RCSTA**. La transmisión se efectúa a través del registro **TXREG**, y la recepción, mediante el **RCREG**. Existen dos registros de estado, **TXSTA** y **RCSTA**, que nos permiten controlar y conocer la actividad de la UART. La velocidad y forma de operación



> **Utilizaremos un conector DB-9 hembra, correspondiente a un periférico (DCE), para conectarnos con la PC (DTE), que tiene un conector macho.**

se controlan a través del registro **BAUDCON** y de un *Baud Rate Generator* (**BRG**) interno.
En MPLAB C18, disponemos de una función de incialización y de cuatro elementales para enviar y recibir información por la UART:

- OpenUSART(): permite la configuración de la UART del microcontrolador, incluyendo velocidad e interrupciones. Por ejemplo:

```
OpenUSART( USART_TX_INT_
OFF & USART_RX_INT_OFF &
USART_ASYNCH_MODE & USART_
EIGHT_BIT & USART_CONT_RX &
USART_BRGH_LOW, spbrg );
```

El puerto serie es la interfaz en una PC u otro dispositivo que transmite datos de a un bit por vez.

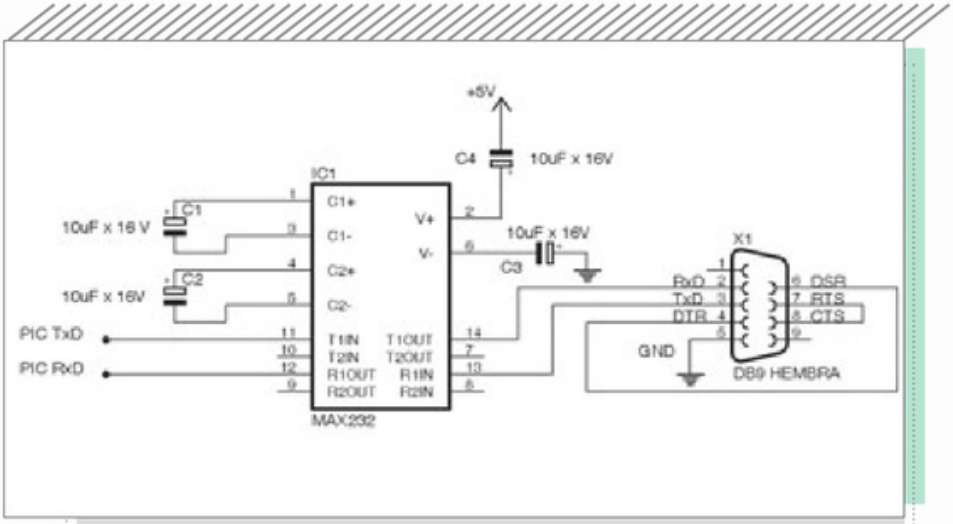
Configura un modo de trabajo sin interrupciones, en 8 bits por carácter, recepción continua a FOSC / (64 * (**spbrg** + 1)) bps.

- WriteUSART(): escribe un byte en el buffer de transmisión de la UART.
- BusyUSART(): permite conocer si la UART está transmitiendo.
- DataRdyUSART: permite conocer si se ha recibido un carácter.
- ReadUSART(): lee un byte del buffer de recepción de la UART.

La operatoria en transmisión es verificar si la UART ha terminado de transmitir y, luego, enviar un carácter. En recepción, verificamos si hay uno disponible y, a continuación, llamamos a la función que nos lo entrega.

Interfaz RS-232

En la **Figura 2** vemos el circuito esquemático de una interfaz RS-232 genérica para nuestros proyectos. Se basa en el **MAX232**, que incluye dos excitadores para convertir las entradas de niveles TTL a RS-232 y dos receptores que transforman entradas RS-232 en salidas a niveles TTL. ■

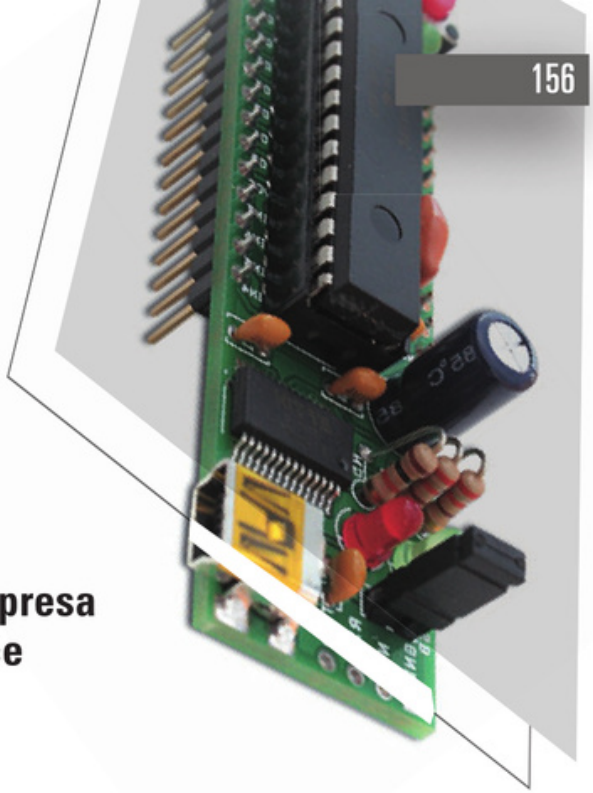


> **Figura 2. El circuito esquemático de la interfaz RS-232.**

> **La mayoría de las nuevas PCs no poseen puerto serie en su hardware. Sin embargo, es posible agregarlo mediante un convertidor de USB a serie.**



SERIAL PERIPHERAL INTERFACE



SPI fue introducido en el mercado por la empresa Motorola. Sin embargo, este protocolo ofrece distintas variantes. Aquí detallaremos las convenciones más aceptadas.

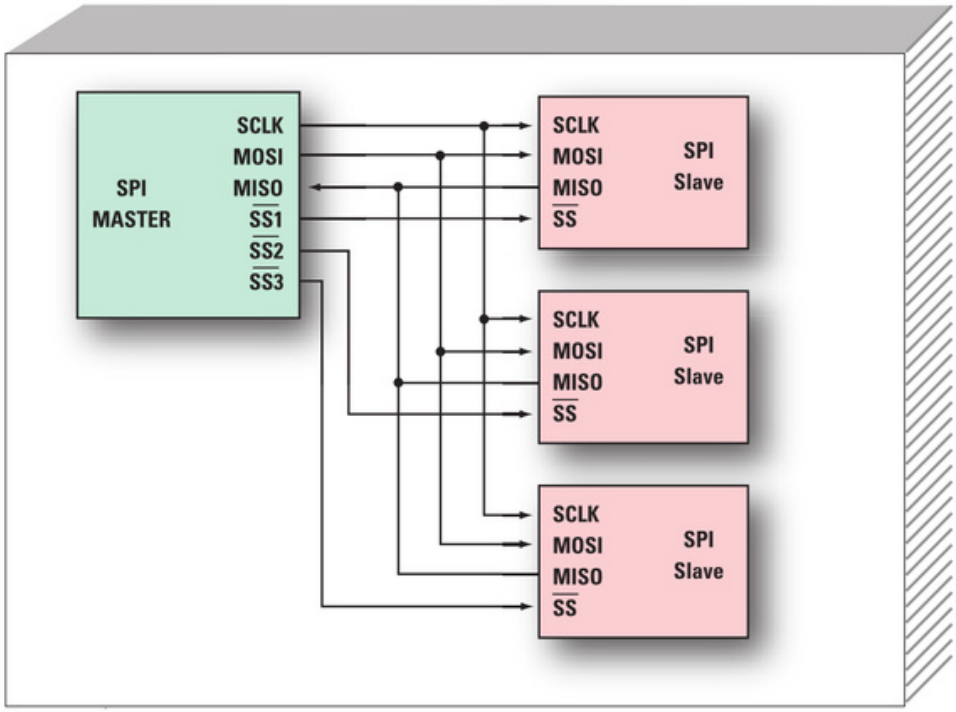
SPI es un protocolo **serial**, **sincrónico** y **full-duplex** muy utilizado en la comunicación entre microcontroladores y periféricos (memorias Flash EEPROM, conversores A/D y D/A, RTCs, etc.),
y de microcontroladores entre sí.
Sus características principales son:

- Utiliza muy pocas líneas de comunicación, y esto lo convierte en una alternativa excelente frente a los buses paralelos.
- Tiene velocidades del orden de 5 a 10 MHz, e incluso mayores.
- La comunicación es del tipo maestro/esclavo. Existe un **maestro** (host) encargado de establecer una comunicación sincrónica, y un **esclavo** (generalmente, un periférico) que responde a los requerimientos (comandos) de aquel.
- Es posible la convivencia de múltiples **esclavos** sobre el mismo bus.

El bus SPI

Está compuesto por dos líneas de **datos**, un **reloj** y una línea de **selección** por cada esclavo. Por ejemplo:

- **MOSI** (*Master Output - Slave Input*): datos de **maestro** a **esclavo**.
- **MISO** (*Master Input - Slave Output*): datos de **esclavo** a **maestro**.

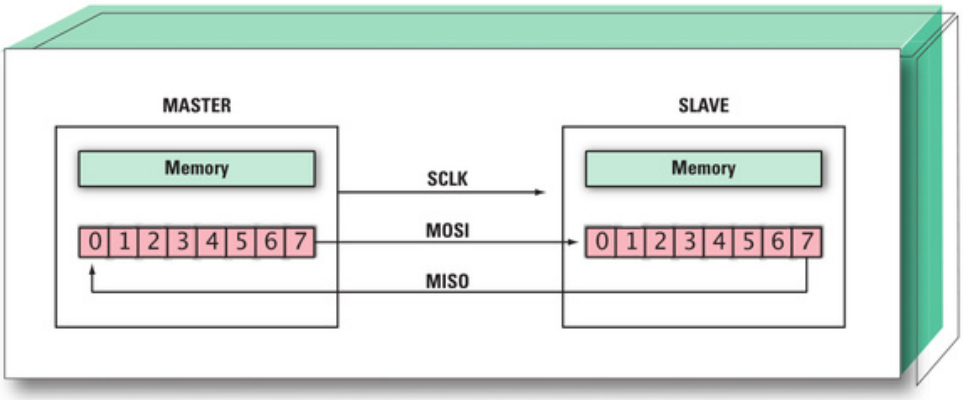


> **Figura 3. Conexión de un maestro con tres esclavos, y denominación de las líneas correspondientes.**

Es importante recordar que el modo multi-master permite tener más de un maestro SPI.

- **CLK o SCLK:** reloj de sincronización que entrega el **maestro** a todos los esclavos del circuito.
- **/CS o /SS** (*Chip Select* o *Slave Select*): líneas de habilitación de los **esclavos**, generalmente activas en nivel bajo. Si el esclavo no es seleccionado (/**CS** se encuentra en nivel alto), su línea de datos de salida (*SO, Slave Output*) queda en estado de alta impedancia, de manera de no interferir con el periférico habilitado en ese momento específico.

Queda claro, entonces, que cada **esclavo** recibe de su **maestro** tanto el reloj (**CLK** o **SCLK**) como la línea de selección/habilitación (**/CS** o **/SS**). El **maestro** tiene el control de la comunicación: establece el sincronismo llevando el reloj a todos los periféricos, y selecciona con quién se quiere comunicar en cada momento mediante **/SS** o **/CS**.



> **Figura 4. Configuración del hardware de una interfaz SPI, donde se establece un anillo circular entre los dos registros de desplazamiento.**

La comunicación SPI

Esta comunicación es, básicamente, el diálogo entre dos registros de desplazamientos de 1 byte (aunque el protocolo no está limitado a solo 8 bits de datos): uno, ubicado en el **maestro**; y el otro, perteneciente al esclavo. Es un anillo circular, donde la información, comandos de operación o datos se desplazan sincrónicamente a medida que los flancos del reloj del maestro disparan el traspaso bit a bit. La información obtenida por ambos queda luego disponible internamente en forma paralela.

La comunicación es full-duplex, al mismo tiempo que el **maestro** transmite información al **esclavo** por la línea **MOSI** (*Master Output-Slave Input*); este le responde mediante **MISO** (*Master Input - Slave Output*). La información viaja al mismo tiempo en ambos sentidos. Debido a la existencia de la línea de reloj para sincronismo, es posible encontrar variaciones sobre ella sin que esto se traduzca en pérdida o distorsión de los datos. Por todo esto, SPI se presenta como un protocolo apropiado para usar con microcontroladores regidos por relojes de relativa precisión, como osciladores **RC**.

Modos SPI

Antes del inicio de una comunicación, son necesarias dos tareas:

- Establecer y configurar una frecuencia de **CLK** apropiada para la comunicación que esté dentro del rango de operación de un esclavo o más.
- Establecer las reglas de interpretación de la señal de sincronismo. Esto significa definir tanto la **polaridad CPOL** (*Clock POLarity*) como la **fase** del reloj **CPHA** (*Clock PHAse*) en cuestión, con respecto a la transferencia de datos.

A estas reglas de interpretación se las conoce como **modos**.

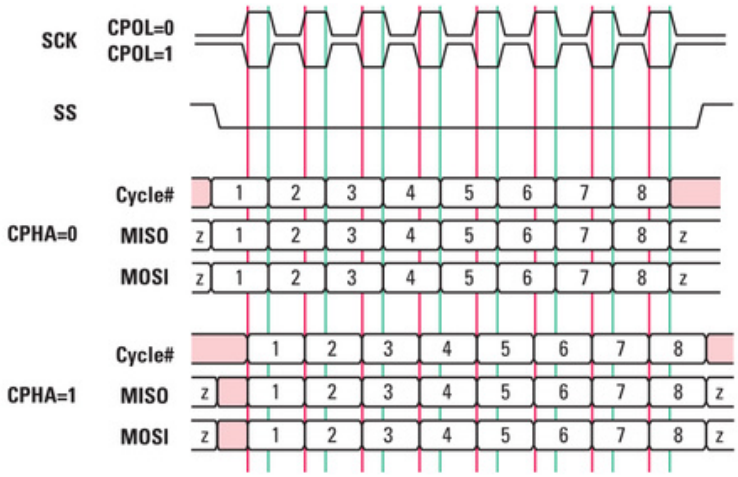
De acuerdo con la convención tomada de Motorola, existen **cuatro modos de operación**, cada uno de ellos determinado por un valor del vector (CPOL; CPHA) (**Tabla 3**).

LOS MODOS SPI

MODO	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Tabla 3. CPOL y CPHA marcan los cuatro modos SPI.

> **Figura 5. Los cuatro modos SPI y la interpretación de la información de reloj con respecto a la transferencia de datos.**





> CONCLUSIONES DE LA FIGURA 5

A) Con CPOL = 0, el valor de "reposo" del CLK es 0 (nivel bajo).

a1) Si CPHA = 0, **los datos se leerán en el primer flanco de CLK**; para este caso: el flanco de subida. A su vez, **los datos cambiarán en el segundo flanco de CLK**, es decir, en el de bajada.

a2) Si CPHA = 1, **los datos se leerán en el segundo flanco de CLK**; para este caso: el de bajada. A su vez, **los datos cambiarán en el primer flanco de CLK**, es decir, en el de subida.

B) Con CPOL = 1, ahora el valor de "reposo" del CLK es 1 (nivel alto).

b1) Si CPHA = 0, **los datos se leerán en el primer flanco de CLK**; para este caso: el flanco de bajada. A su vez, **los datos cambiarán en el segundo flanco de CLK**, es decir, en el de subida.

b2) Si CPHA = 1, **los datos se leerán en el segundo flanco de CLK**; para este caso: el flanco de subida. A su vez, **los datos cambiarán en el primer flanco de CLK**, es decir, en el de bajada.

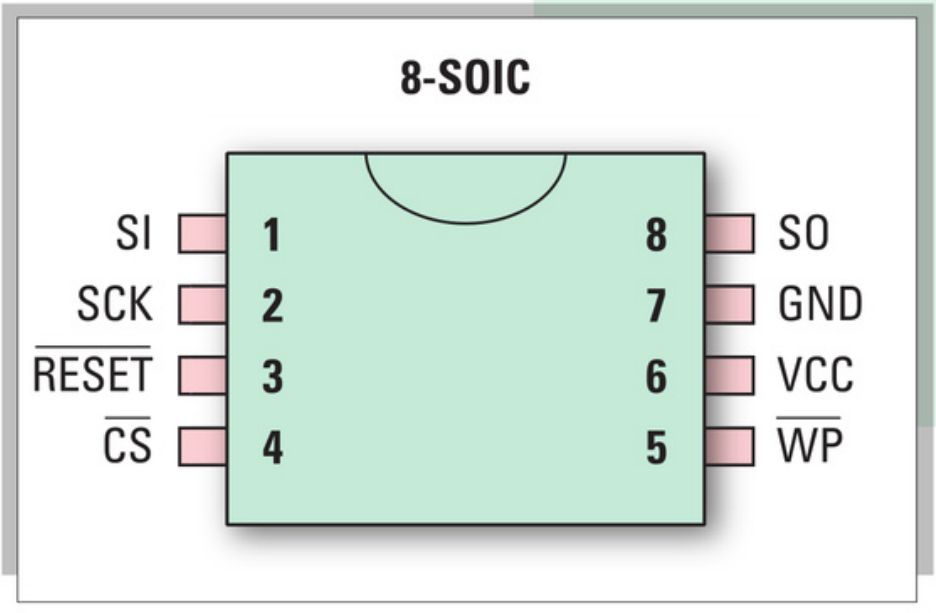
El módulo MSSP del PIC18F4620 puede ser configurado para operar en modo SPI.

Algunas conclusiones que se desprenden de la **Figura 5** son que, con **CPHA = 0**, el dato en la línea **MOSI** debe estar disponible al menos medio ciclo de reloj antes del primer flanco. También se requiere que la polaridad de **CLK** esté bien establecida antes de habilitar con /**SS** al dispositivo **esclavo**.

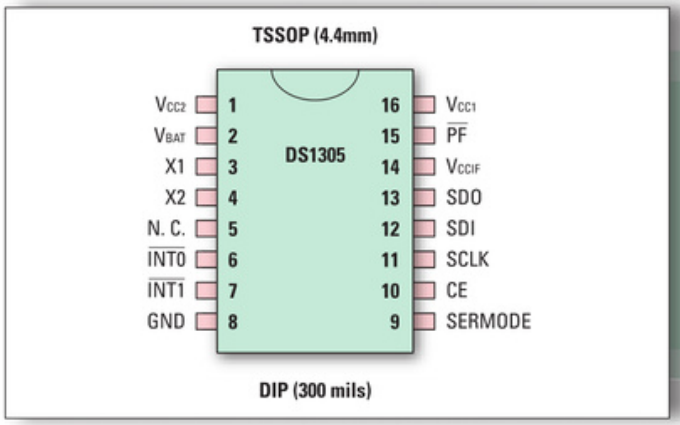
Tipos de periféricos

Existe una gran variedad de periféricos que incluyen el protocolo SPI como interfaz para intercambio de datos y configuración. Entre los más conocidos y habituales podemos destacar los siguientes:

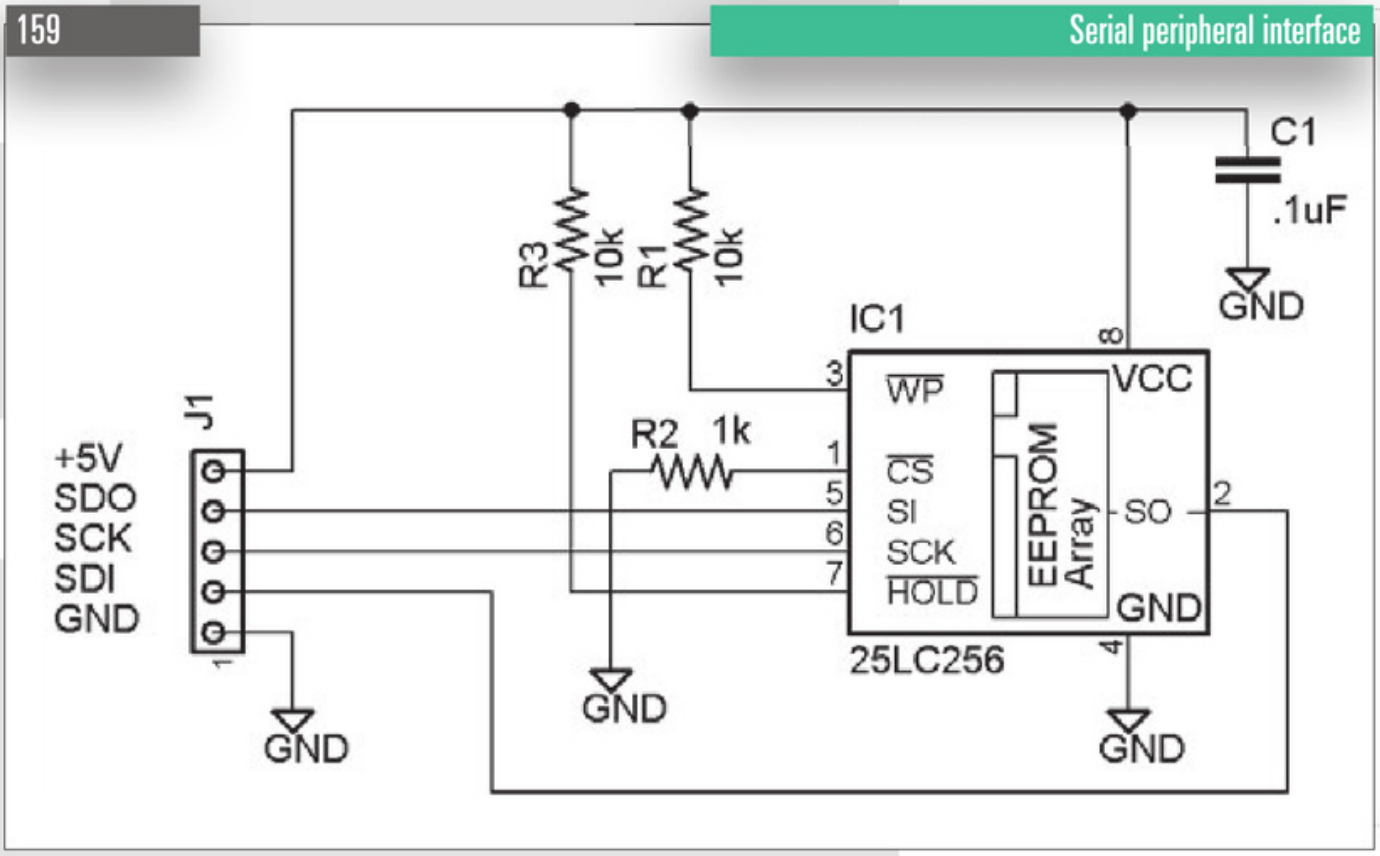
- Memorias **Flash, EEPROM**, **FRAM**: por ejemplo, **AT45DB041B** de Atmel (flash), **25LC256** (EEPROM) y **FM27L256B** (FRAM).
- **RTC** (*Real Time Clocks*) y accesorios: por ejemplo, el **DS1305** de Maxim y los Processor Companions como el **FM3316** de Ramtron.
- **Conversores A/D y D/A**: por ejemplo, el **MAX5222** de Maxim, doble D/A de 8 bits en encapsulado **SOT23-8**.



> Figura 6. Chip AT45DB041b de Atmel, distribución de pines y encapsulado.



>Figura 7. Chip DS1305 de Maxim, distribución de pines y encapsulado.



> **Figura 8. Circuito esquemático de una interfaz genérica para una memoria SPI, en este caso, una 25LC256.**

```
        SSPEN=1;                // Habilita las líneas MOSI, MISO y SCK
para la interfaz SPI.

        CKP=1;                  // "Clock Polarity": indica el estado
inactivo ó de reposo del SCK.
        STAT_CKE=0;     // "Clock Edge Select": para Mode 3 los datos son
transmitidos en
                                // el flanco de subida, por lo que CKE=0.
        STAT_SMP=1;     // "Sample Bit": Los datos de entrada deben ser
muestreados al
                                // final del tiempo de los datos de salida.
        SSPM3=0;                // Aquí se configura la frecuencia del CLK
a Tosc*4
        SSPM2=0;
        SSPM1=0;
        SSPM0=0;
```

Luego, para realizar una escritura seguida de una lectura, efectuamos lo siguiente:

```
        SSPBUF=txdata;  // Carga del registro SSPBUF con el dato a
transmitir
        while(!STAT_BF);        // Espera a que se setee el flag "Buffer
full".
        rxdata= SSPBUF; // Se efectúa la lectura del Esclavo
```

En la próxima página veremos cómo agregar un dispositivo SPI a una placa de desarrollo que hayamos armado con anticipación.

Uso de SPI en el PIC18F4620

El módulo **MSSP** (*Master Serial Synchronous Port*) de este micro puede ser configurado para operar en modo SPI, estableciendo los registros de control **SSPCON1** y **SSPSTAT**. Observemos las siguientes características para su uso como **maestro**:

- Los pines que son salidas (**SDO y SCK**) deberán configurarse de modo maestro.
- La entrada (**SDI**) es controlada por el módulo **MSSP**.
- En el registro **SSPCON1**, el bit **CKP** controla la polaridad del reloj.
- En el registro **SSPSTAT**, los bits **SMP** y **CKE** controlan en qué flanco se leen y escriben los datos, respectivamente.
- Los bits **SSPM0** a **SSPM3** eligen el modo maestro o esclavo y la frecuencia de clock.
- La escritura se inicia escribiendo el registro **SSPBUF**.
- La lectura se realiza al final del ciclo de escritura, lo cual es indicado por el micro seteando el flag **BF** en el registro **SSPSTAT**. Si no se requiere escribir nada, se ingresa cualquier valor, como **0xFF**.

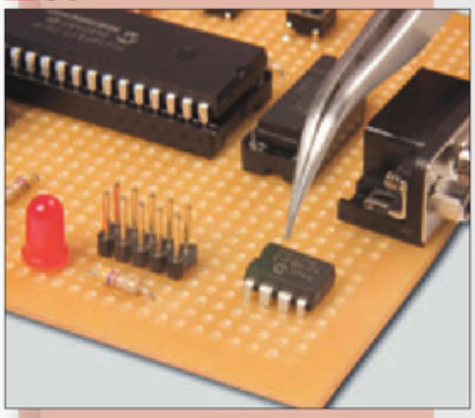
Por ejemplo, en C, para configurar el **MODO 3 (izq.)**:

PASO A PASO

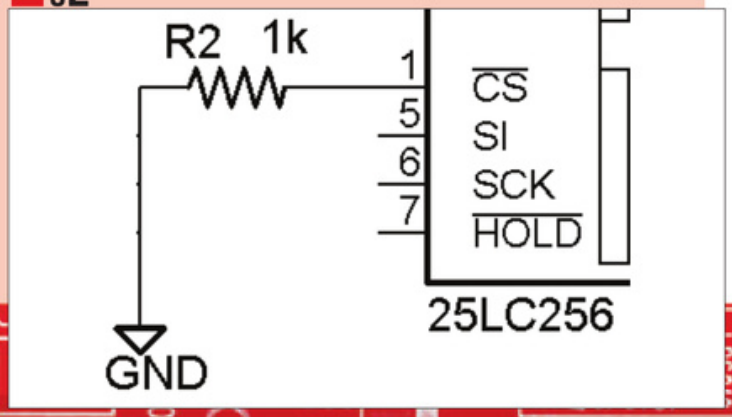
CONEXIÓN DE UNA MEMORIA SPI

Utilizaremos la placa de desarrollo armada con anticipación, en conjunto con una memoria EEPROM 25LC256 para realizar la práctica.

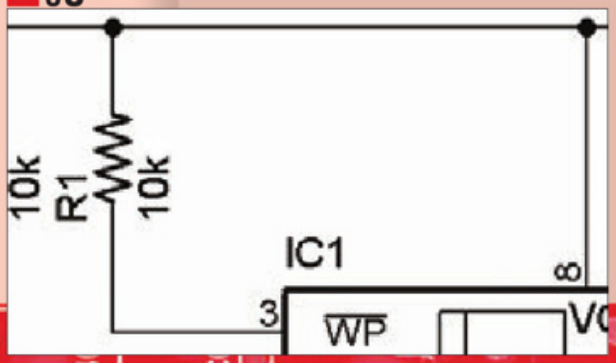
01



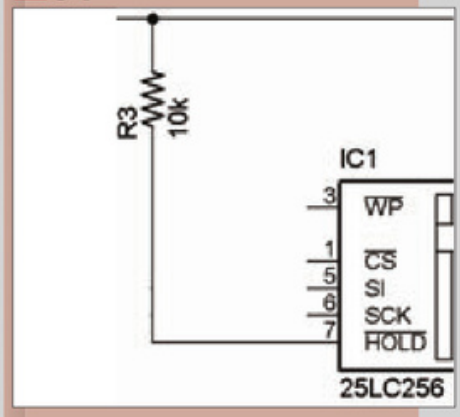
02



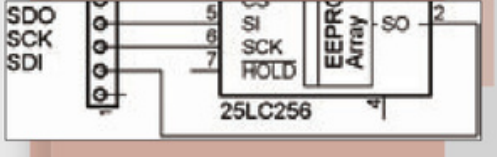
03



04



05



01. Montaremos la memoria EEPROM sobre la placa con cuidado de no doblar ni quebrar ninguno de sus pines. Utilizaremos cables de color para realizar las conexiones al bus de alimentación del protoboard: rojo para el positivo (pin número 8) y negro para la masa (pin número 4). Soldamos también el capacitor de desacople.

02. Es el turno del pin número 1: Chip Select (CS), que nos permite habilitar o no el dispositivo. Se lo utiliza con el fin de direccionar diferentes dispositivos que se encuentran conectados al mismo bus SPI. Como en nuestra práctica solo disponemos de una memoria, debemos habilitarlo conectándolo a través de una resistencia de 1 kΩ a masa.

03. Siguiendo con la memoria, debemos conectar, mediante una resistencia de 10 kΩ, el pin número 3: Write-Protect (WP), a positivo. Este pin se utiliza para proteger contra escritura el contenido de la memoria, y se activa mediante un nivel lógico bajo. Como no le daremos utilidad, lo dejamos desactivado para trabajar sin problemas.

04. El próximo pin por conectar es el 7: HOLD. Este funciona a modo de pausa, es decir que coloca la comunicación en un estado de suspensión ignorando todas las peticiones que lleguen del dispositivo maestro. Tampoco lo utilizaremos por el momento, de modo que lo conectamos al positivo mediante una resistencia de 10 kΩ.

05. Conectamos los pines de comunicación: el 5 (SI), el 2 (SO) y el 6 (SCK), mediante cables a los correspondientes pines del microcontrolador: 24 (SDO), 23 (SDI) y 18 (SCK). Observemos que salida y entrada se interconectan una con la otra.

BUS DE COMUNICACIÓN I²C

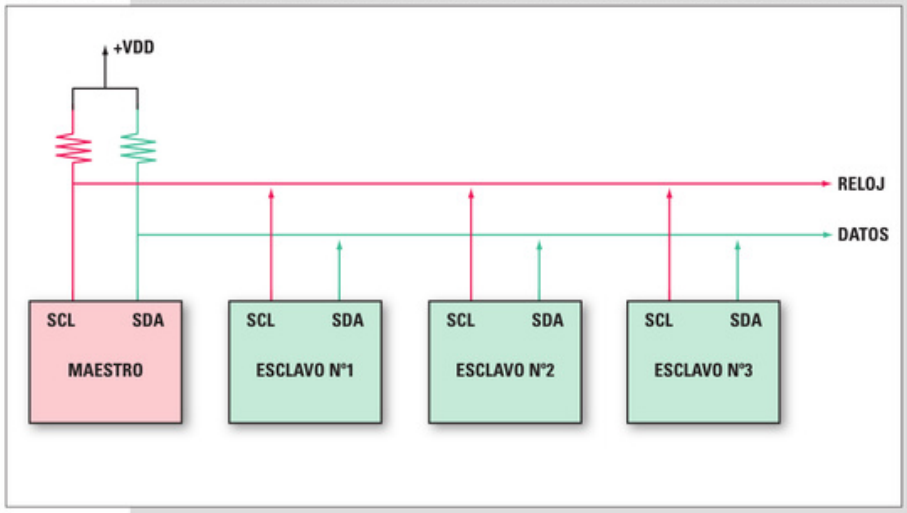
El bus I²C (Inter Integrated Circuits), diseñado por Philips Semiconductors en la década del 80, nació con el fin de realizar conexiones entre circuitos integrados.

El I²C es un bus **half-duplex** de comunicación de datos, de tipo serial sincrónico. Se desarrolló inicialmente para comunicar microcontroladores con distintos dispositivos electrónicos a través de dos hilos de forma bidireccional. En la actualidad, se lo usa en aparatos electrónicos de consumo, como televisores, equipos de audio, reproductores de DVD y monitores de computadora.

Permite conectar gran cantidad de circuitos integrados utilizando la configuración maestro/esclavo y con solo dos hilos: datos (SDA) y reloj (SCL). Dichas líneas son bidireccionales y están polarizadas a positivo mediante sendas resistencia de **pull-up**, por lo que en reposo se encuentran en nivel alto.

Los dispositivos **maestro** se encargan de generar la señal de reloj, de iniciar el protocolo de comunicación y de recibir las peticiones de los **esclavos**. La transmisión es en formato serie, y es posible alcanzar tasas del orden de **100 kbits** por segundo en la modalidad estándar, de **400 kbits** por segundo en el modo rápido y de hasta **3,4 Mbits** por segundo en el modo de alta velocidad.

Cada chip conectado al bus posee una dirección única para su identificación, seleccionable por software. Es posible que en la misma red convivan varios dispositivos maestro, debido a que el protocolo cuenta con un detector de colisiones. El I²C trabaja con palabras de datos de 8 bits, tanto para las direcciones como para la información que se transmite.

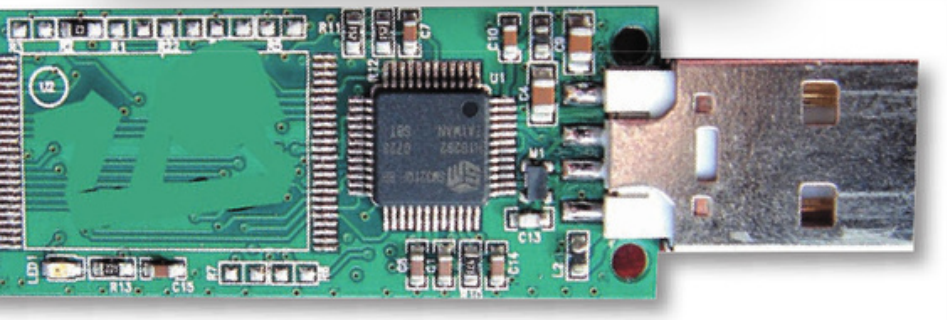


> **Figura 10. Conexión de múltiples dispositivos mediante el bus. Es necesario que ambas líneas se encuentren conectadas a positivo mediante resistencias pull-up.**

El I²C es un bus half-duplex de comunicación de datos, de tipo serial sincrónico.

Protocolo de comunicación

Ya sabemos que para realizar una comunicación entre dos o más dispositivos mediante el bus necesitamos solo dos líneas: datos y reloj. Cada dispositivo puede actuar como maestro o esclavo indistintamente, y posee una dirección que lo identifica. Por este motivo, es necesario cumplir una serie de condiciones para que la comunicación sea exitosa; en otras palabras, debemos seguir el protocolo de comunicación.



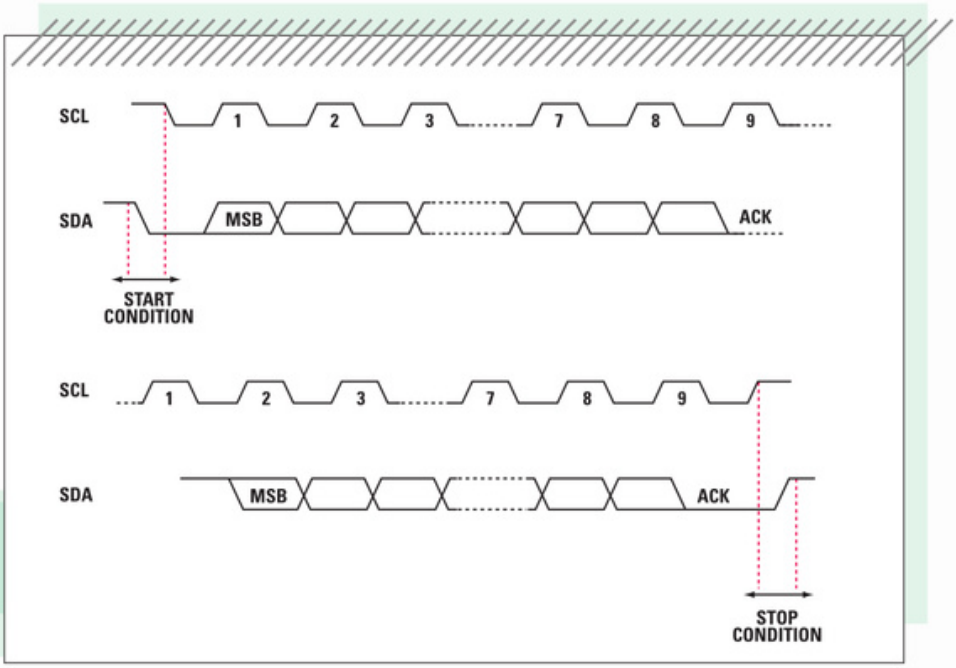
Condiciones de START y STOP

El primer paso antes de enviar palabras de datos entre los distintos dispositivos conectados es avisar que se iniciará una transmisión, es decir, ingresar en la condición de **START**. El encargado de hacerlo es el **maestro**. Para esto, debe enviar la línea de datos **SDA** a nivel bajo, mientras mantiene la de **reloj SCL** en nivel alto. A continuación, los dispositivos podrán enviar y recibir datos.
Una vez que concluye el proceso de transmisión, el maestro debe informarlo; es decir, ingresaremos en la condición de **STOP**. Para esto se debe llevar la línea SDA a nivel alto, mientras que la SCL permanece en nivel alto.

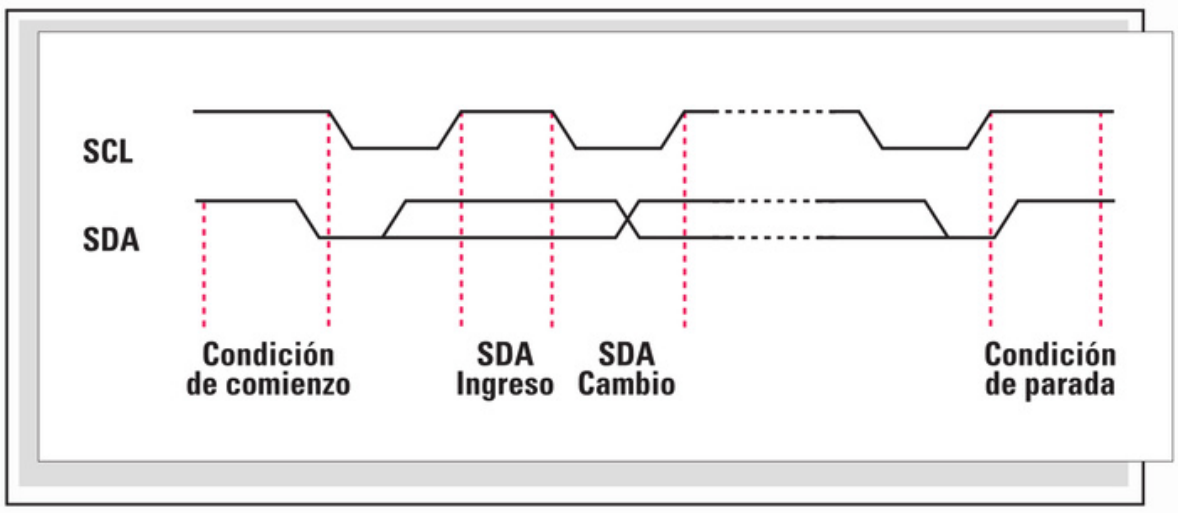
Transferencia de datos

Una vez que el dispositivo maestro generó la condición de inicio, comienza la transferencia. Como ya sabemos, las palabras deben tener una longitud de 8 bits, y la primera en enviarse representa la dirección del dispositivo esclavo seleccionado para establecer la comunicación. Dicha dirección está compuesta por 7 bits, lo que da un máximo de 128 dispositivos posibles (0 a 127).
El bit restante es el **R/W**, que permite seleccionar entre el modo de escritura y el de lectura.
El próximo paso que debe seguir el maestro es leer el estado de la línea SDA, impuesto por el dispositivo **esclavo**. En caso de que sea un nivel bajo, la transferencia continúa normalmente; en cambio, si la línea adopta un nivel alto, quiere decir que el esclavo no reconoce la transferencia o no está listo para la

Cuando finaliza la transmisión, el maestro debe generar la condición de STOP para liberar al bus.



> **Figura 10. Diagramas de transferencia de datos, junto a las condiciones de START, STOP y validación (acknowledge).**



> **Figura 11. Diagrama de tiempos en el que se muestran las condiciones de START y STOP, y los niveles que adoptarán las líneas SDA y SCL.**

comunicación. En el paso siguiente se genera desde el Maestro la condición de **STOP** para dejar el bus libre. Este proceso de validación se denomina **ACK** (*acknowledge*). Siempre que finaliza la transmisión, el maestro debe generar la condición de STOP para liberar al bus y dejar el sistema a la espera de nuevas comunicaciones.

Comparación entre I²C y SPI

Si bien los buses de comunicación **I²C** y **SPI** transmiten la información en formato serie y permiten conectar dispositivos similares, presentan grandes diferencias en su funcionamiento. El bus SPI tiene las siguientes características:

- Permite realizar una comunicación **full-duplex**, a diferencia de la **half-duplex** que maneja el bus **I²C**.
- Logra mayores velocidades de transmisión de datos.
- No limita la palabra de datos que se va a transmitir a 8 bits.
- Utiliza más líneas por dispositivo.
- Direcciona los dispositivos mediante líneas específicas, mientras que **I²C** utiliza una palabra de 7 bits para realizarlo.
- No posee bit de reconocimiento (ACK). Esto puede derivar en un gran problema, debido a que es posible enviar datos desde el maestro sin que ningún esclavo responda.
- No permite utilizar, por lo menos fácilmente, varios dispositivos maestro conectados en el bus.

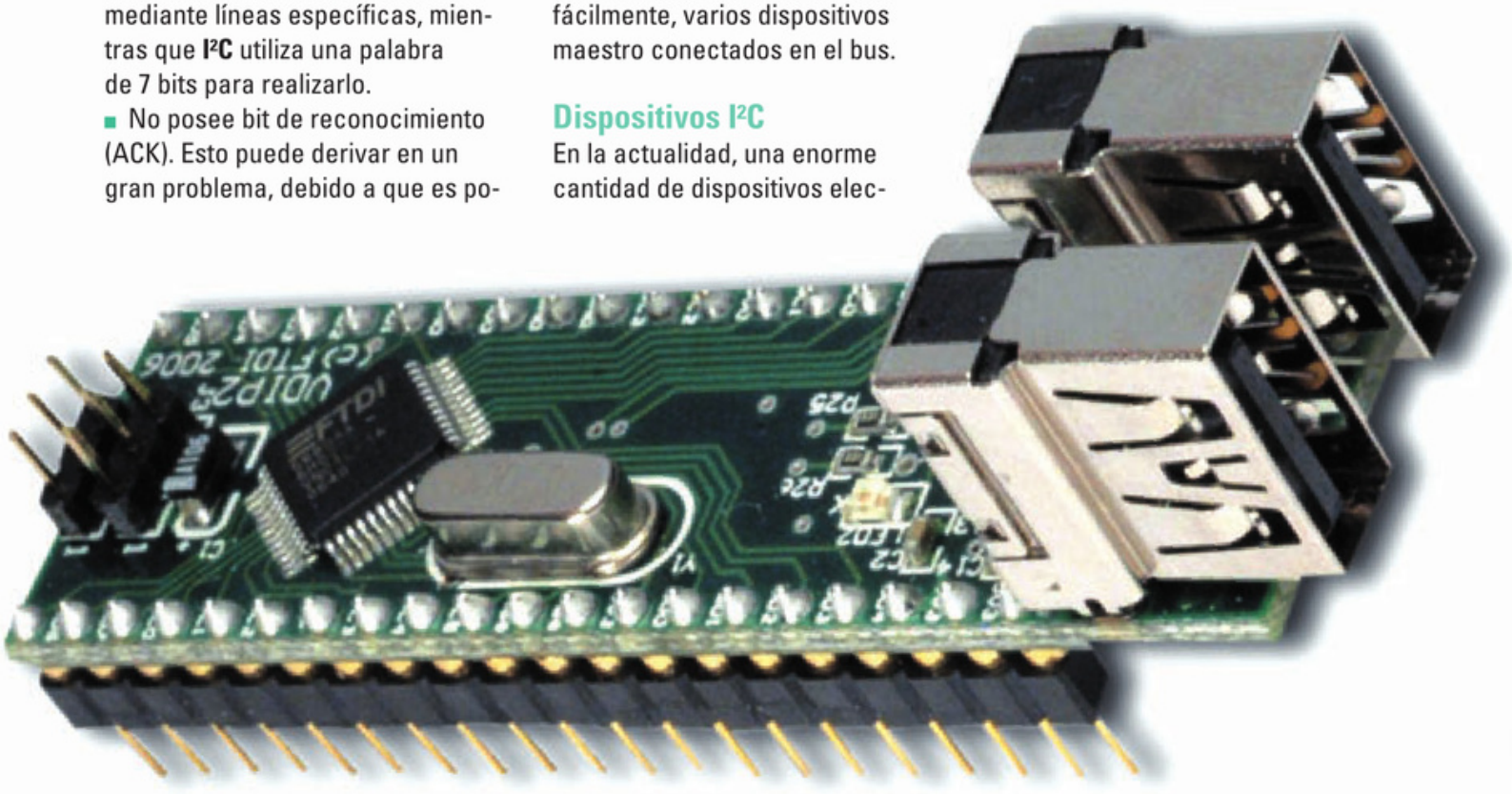
> 1-WIRE, IBUTTON

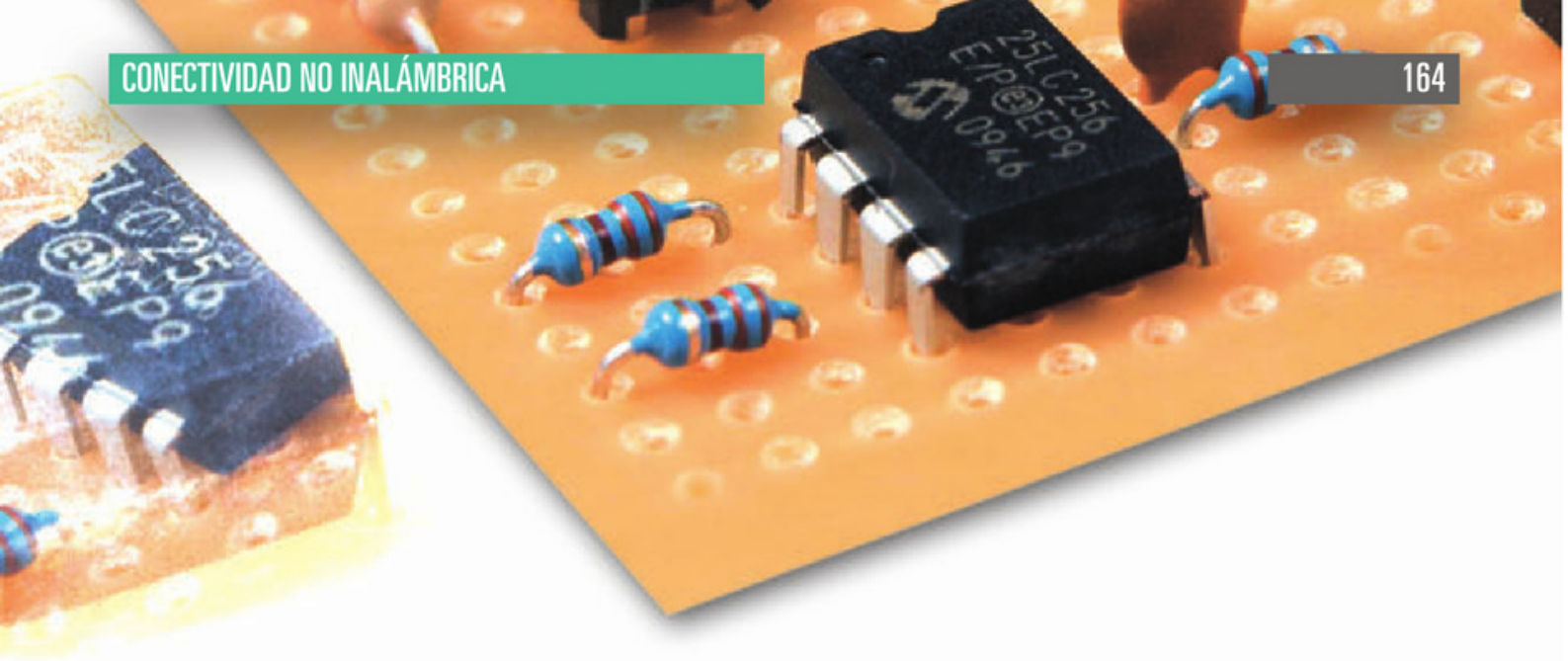
El protocolo 1-wire fue popularizado por Dallas Semiconductor, hoy perteneciente a Maxim. La familia iButton incluye periféricos que utilizan este protocolo y están contenidos en un recipiente metálico estanco (bien protegidos). El protocolo utiliza distintos anchos de pulso para comunicarse en una wired-OR, de la cual también se alimentan los dispositivos. Esta alternativa es muy popular entre los aficionados, ya que se dispone de sensores de temperatura, memorias, números de serie, relojes de tiempo real, etc.

Dispositivos I²C

En la actualidad, una enorme cantidad de dispositivos electrónicos integran el protocolo de comunicación **I²C**, lo que facilita enormemente la tarea de realizar buses de comunicación en sistemas electrónicos complejos. El bus ha tomado tanta popularidad en el diseño electrónico, que es posible encontrarlo integrado en equipos domésticos, y también en en sistemas a nivel industrial. Por ejemplo, es común disponer de series de diversos circuitos integrados con el bus incluido:

- Memorias seriales de tipo EEPROM.
- Microcontroladores de diferentes gamas.
- Displays LCD.
- Convertidores analógico-digitales y digital-analógicos.
- Procesadores de sonido.
- Sensores de distintos tipos.
- Relojes de tiempo real (RTC).
- Llaves digitales.





Uso de I²C en el PIC18F4620

El módulo **MSSP** (*Master Serial Synchronous Port*) de este micro puede ser configurado para operar en modo I²C, operando sobre los registros de control **SSPCON1**, **SSPCON2** y SSPSTAT. Observemos las siguientes características para el uso como **maestro**:

- Los pines (**SDA** y **SCL**) deben configurarse como entradas en el registro **TRISC**.
- Los bits **SSPM0** a **SSPM3** de SSPCON1 se configuran al valor **1000b** para modo **maestro**.
- La velocidad de operación se controla mediante el registro **SPADD**.
- La interfaz se configura para transmitir o recibir mediante el bit **RCEN** en el registro **SSPCON2**.
- El módulo facilita la operación por interrupciones; luego de cada operación elemental se configura el flag **SSPIF** en el registro **PIR1**, que debe resetearse por software.

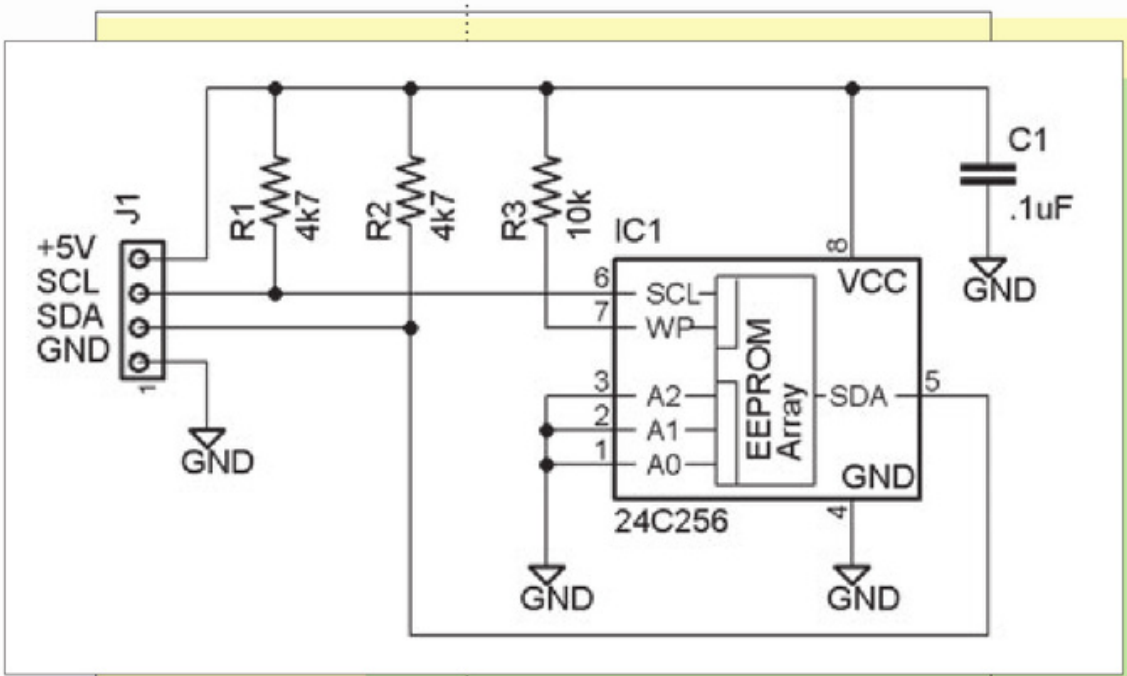
Una secuencia típica de direccionamiento de un dispositivo es la siguiente:

- El modo se configura en transmisión.
- La condición de START se inicia seteando el bit **SEN** en el registro **SSPCON2**.
- La transmisión de la dirección se inicia escribiendo el registro **SSPBUF**, el modo es el bit **0**.
- El **ACK** del dispositivo se observa en el bit **ACKSTAT** en el registro **SSPCON2**.

Según se trate de lectura o escritura, se cambia o no el modo de la interfaz (bit **RCEN**):

- Para una transmisión, los bits se ingresan en el registro **SSPBUF**.
- Para una recepción, el envío (o no) de **ACK** al dispositivo se configura en el bit **ACKDT** en el registro **SSPCON2**. La lectura de cada byte se efectúa en el registro **SSPBUF**.
- La operación se termina con una condición de **STOP**, que se inicia seteando el bit **PEN** en el registro **SSPCON2**.

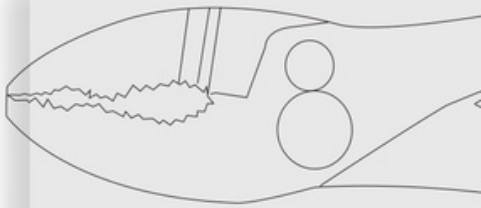
En la página siguiente veremos cómo agregar un dispositivo I²C a una plaza de desarrollo. ■



> **Figura 12. Circuito esquemático de una interfaz genérica para una memoria I²C, en este caso, una 24C256.**

PASO A PASO

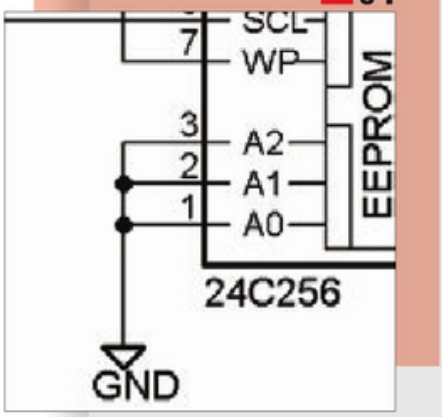
CONEXIÓN Y MANEJO DE UN PERIFÉRICO I²C

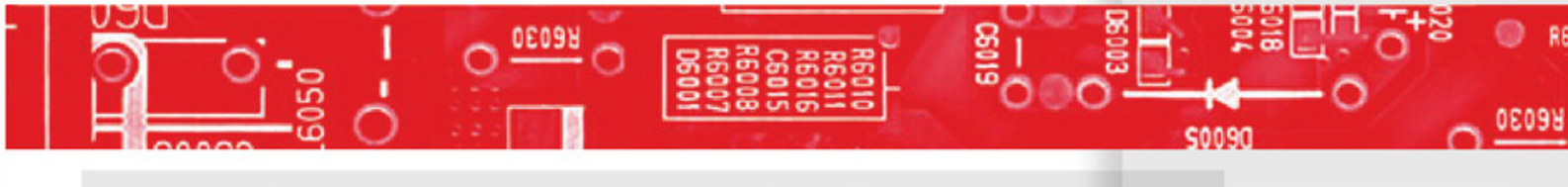


Utilizaremos la placa de desarrollo con una memoria EEPROM 24C256. El PIC18F4620 incluye un módulo I²C, lo que nos simplifica la tarea de escribir el código.

Montamos la memoria sobre la placa tomando los recaudos necesarios para no doblar o quebrar ninguno de los pines de manera accidental. Utilizando cables de color rojo (positivo) y de color negro (negativo), realizamos las conexiones correspondientes de la alimentación de la memoria y el capacitor de desacople. ■

01





01. La memoria tiene tres pines que permiten direccionarla mediante hardware. Como solo tenemos una memoria, conectamos a masa los tres pines de dirección: A0 (1), A1 (2) y A2 (3).

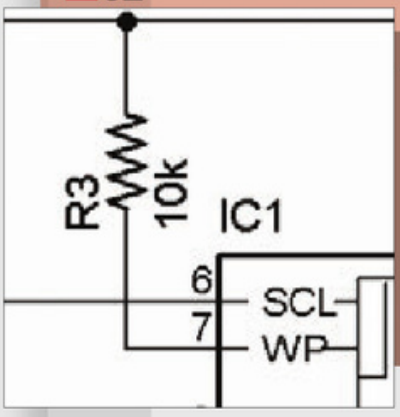
02. Conectamos, mediante una resistencia de 10 kΩ, el pin WP (7) Write-Protect a positivo. Este se activa mediante un nivel lógico bajo.

03. Conectamos las dos resistencias de pull-up del bus I²C. Utilizamos dos de 4k7 conectadas a positivo.

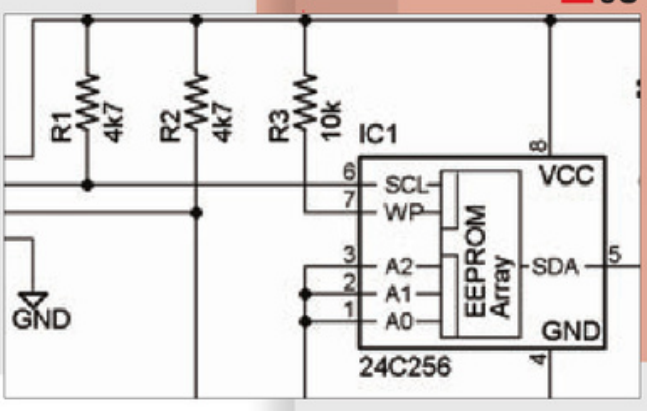
El extremo libre de una de las resistencias lo conectaremos al pin de datos SDA de la memoria (5), y el extremo libre de la otra resistencia va conectado al pin de reloj SCL de la memoria (6).

04. Vamos a unir con cables los pines del bus de la memoria con los del microcontrolador. El pin de datos SDA del microcontrolador (23) se conecta con el SDA de la memoria (5); y el pin de reloj SCL del microcontrolador (18), con el correspondiente en la memoria (6).

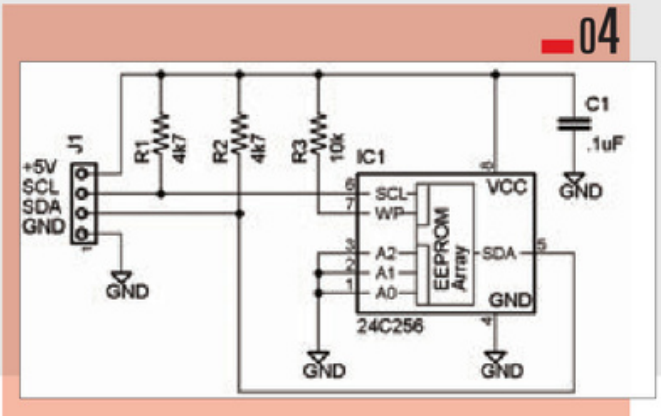
02



03

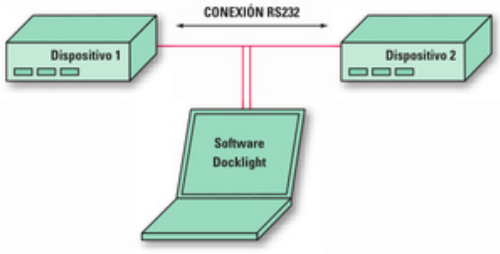


04



DOCKLIGHT

Docklight es un software para verificación, análisis y simulación de protocolos de comunicación serie (RS-232, RS-485 y otros). Veamos cómo utilizarlo.

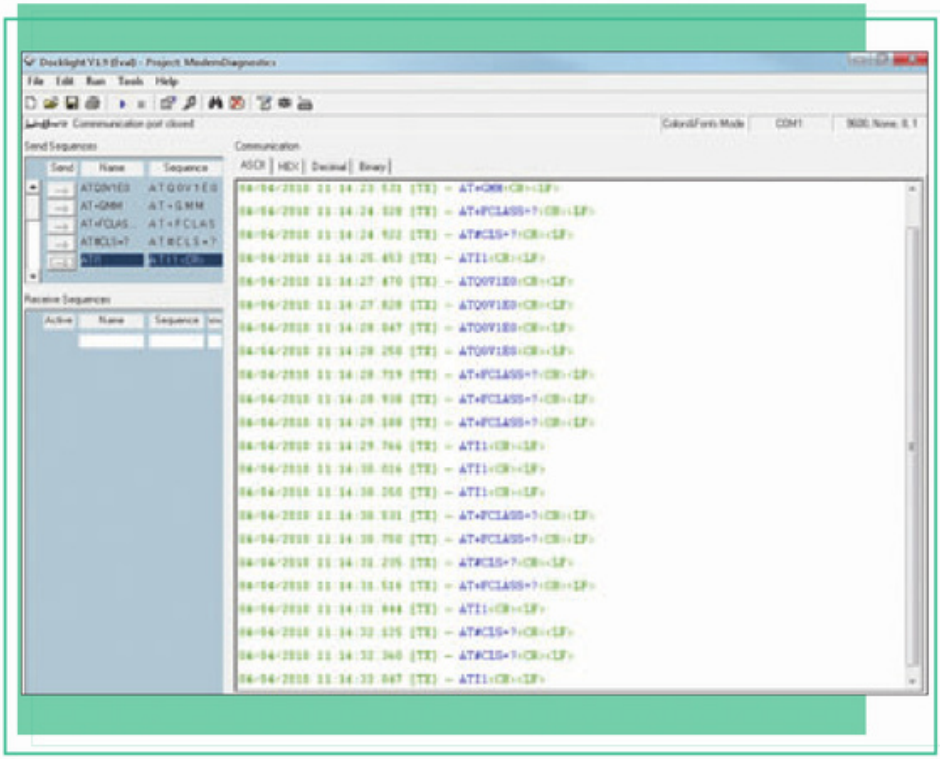


> Es posible insertar Docklight entre la comunicación de dos dispositivos serie para monitoreo. Debemos poseer dos puertos COM en la PC.

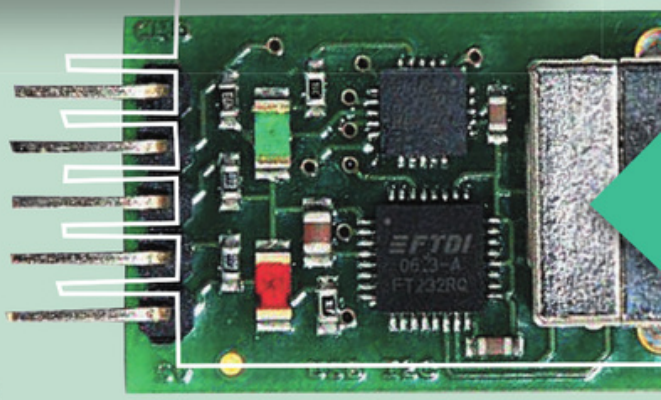
Este programa permite realizar el monitoreo de la comunicación serie entre dos dispositivos o simular el de uno de ellos para verificar el comportamiento del otro.
Las funciones de este software son las siguientes:

- **Simular un protocolo serie.** Docklight puede enviar secuencias de datos definidas por el usuario según el protocolo utilizado; además, es capaz de actuar según las secuencias que reciba. Esto hace posible simular el comportamiento de un dispositivo serie, algo particularmente útil para generar condiciones de verificación que son difíciles de reproducir con el dispositivo original.
- **Registrar datos de RS-232.** Cualquier comunicación serie puede ser registrada utilizando dos formatos de archivos diferentes. Es posible emplear texto normal para registrar una gran cantidad de datos rápidamente, o utilizar un formato HTML con texto que permita distinguir fácilmente entre los datos entrantes y salientes.
- **Detectar secuencias de datos específicas.** En muchos casos, necesitamos verificar la ocurrencia de una secuencia específica dentro de la información en RS-232 que indica una condición de error. Docklight maneja una lista de estas secuencias, y es capaz de realizar acciones definidas por el usuario en caso de detectarlas.
- **Responder a datos entrantes.** Docklight da la posibilidad de especificar respuestas definidas por el usuario a diferentes secuencias recibidas. De esta manera, es posible construir un simulador básico para nuestro dispositivo serie, y rastrear errores enviando un comando de diagnóstico luego de recibir un mensaje de error. ■

Docklight es un software de verificación, análisis y simulación de protocolos.



> Con Docklight podemos verificar la implementación del protocolo serie en un dispositivo. Es posible definir una secuencia de control reconocible por el dispositivo y analizar las respuestas que este genera.



UNIVERSAL SERIAL BUS

El puerto USB es una tecnología nueva, si lo comparamos con el tradicional serie. Veamos cuáles son las ventajas que podemos aprovechar para nuestros proyectos.

El puerto **USB** (*Universal Serial Bus*) es una interfaz de relativa complejidad. Una transferencia USB implica una comunicación serial, bidireccional y de índole "diferencial" sobre un cable compuesto por cuatro conductores: **GND** (común), **VCC** (alimentación) y un par diferencial balanceado para transporte de datos.

Existen cuatro tipos de transferencias definidas para una comunicación USB:

- **Isocrónicas:** garantizan una velocidad de transferencia determinada, pero con posibles pérdidas de datos.
- **Del tipo interrupción:** se utilizan cuando un dispositivo requiere garantizar transferencias rápidas.
- **Bulk:** realiza transferencias largas y esporádicas, que utilizan todo el ancho de banda remanente disponible, pero que no precisan de una entrega en un tiempo determinado. El **USB Host** le asigna la prioridad más baja a este tipo de transferencias.
- **De control:** usada para el envío de comandos cortos o información de estado al dispositivo.

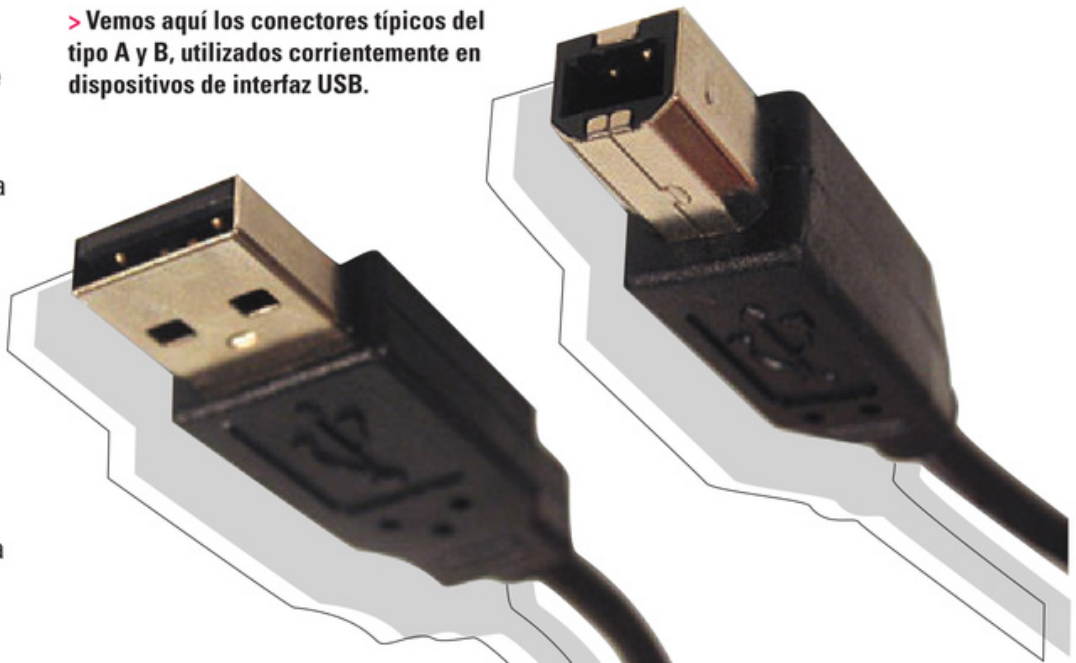
Cada vez que un dispositivo USB se conecta a un host, se inicia un proceso de enumeración, que comienza mediante el envío de una señal de reset al dispositivo. Durante esta etapa, se determina la tasa de transferencia del periférico USB. El USB Host lee la información y le asigna una dirección única de 7 bits. Una comunicación USB está basada en el establecimiento de pipes o canales lógicos de conexión. Se denomina *endpoint* al bloque de memoria de datos o registro del chip controlador sobre el cual se establece la comunicación física. Es posible trabajar a distintas tasas de transferencia:

- Show Speed: 10 - 100 Kbps
- Full Speed: 500 Kbps - 10 Mbps
- High Speed (USB 2.0): 25 – 480 Mbps (Mega bits por segundo).

Las diferentes tasas de transferencia involucran distintos niveles de tensión utilizados para la comunicación en el par diferencial.

> Cada vez que un dispositivo USB se conecta a un host, se inicia un proceso de enumeración.

> Vemos aquí los conectores típicos del tipo A y B, utilizados corrientemente en dispositivos de interfaz USB.



Cada intercambio entre dispositivos involucra tres paquetes:

- Paquete tipo **token**: contiene la dirección del dispositivo (número de endpoint) para el direccionamiento.
- Paquete de datos: contiene los datos propiamente dichos de la transacción entre dispositivos.
- Paquete de terminación: indica el fin de la transacción, de modo que pueda procesarse.

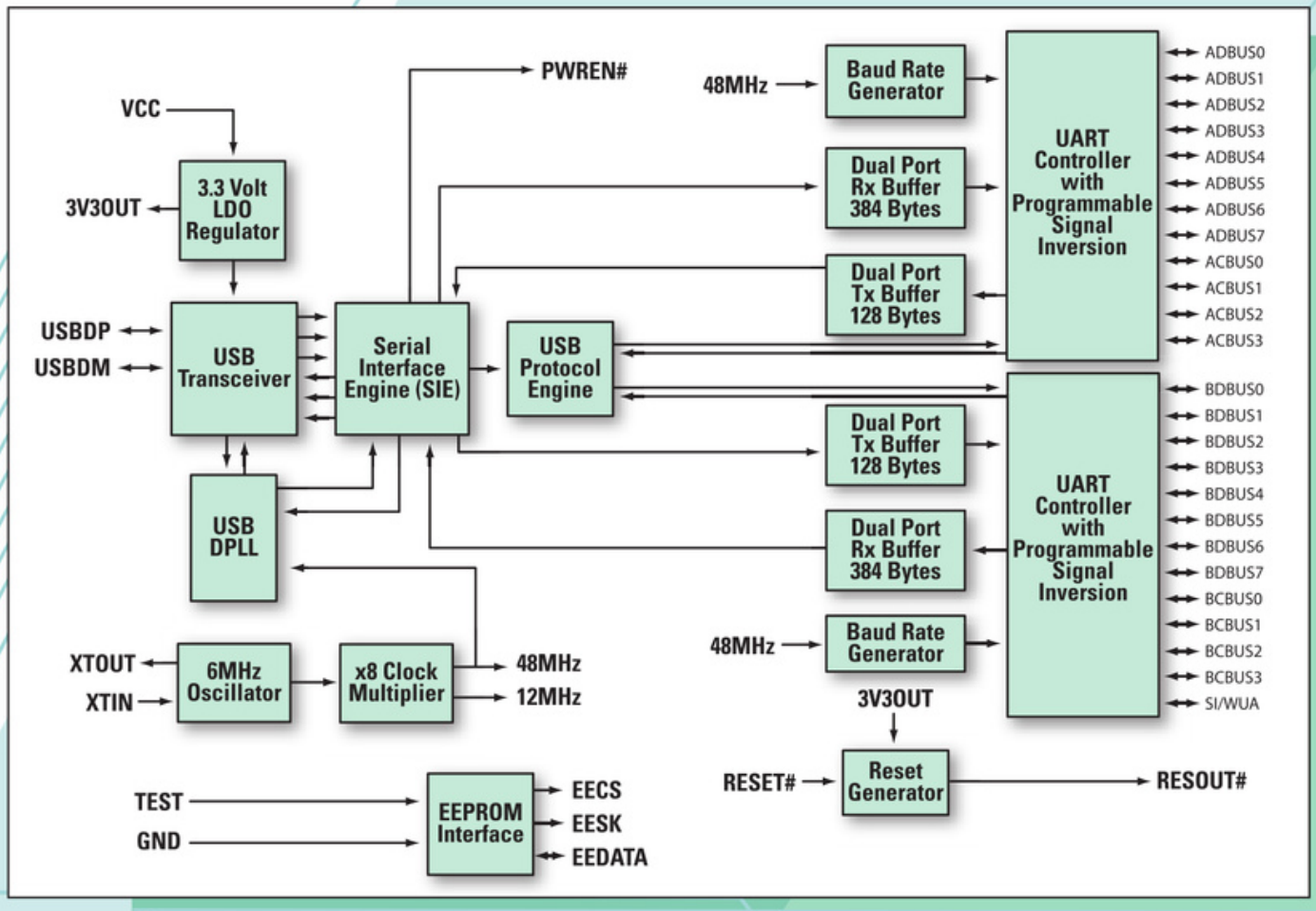
USB fácil: FT2232D chip

El núcleo del **FT2232D**, fabricado por FTDI, está conformado por dos controladores que manejan el proceso de enumeración y otras comunicaciones sobre el bus USB. Cada una ofrece distintos modos de configuración, que implementan la operatividad de otros dos chips de la familia: el **FT232B** y el **FT245B**, como ser:

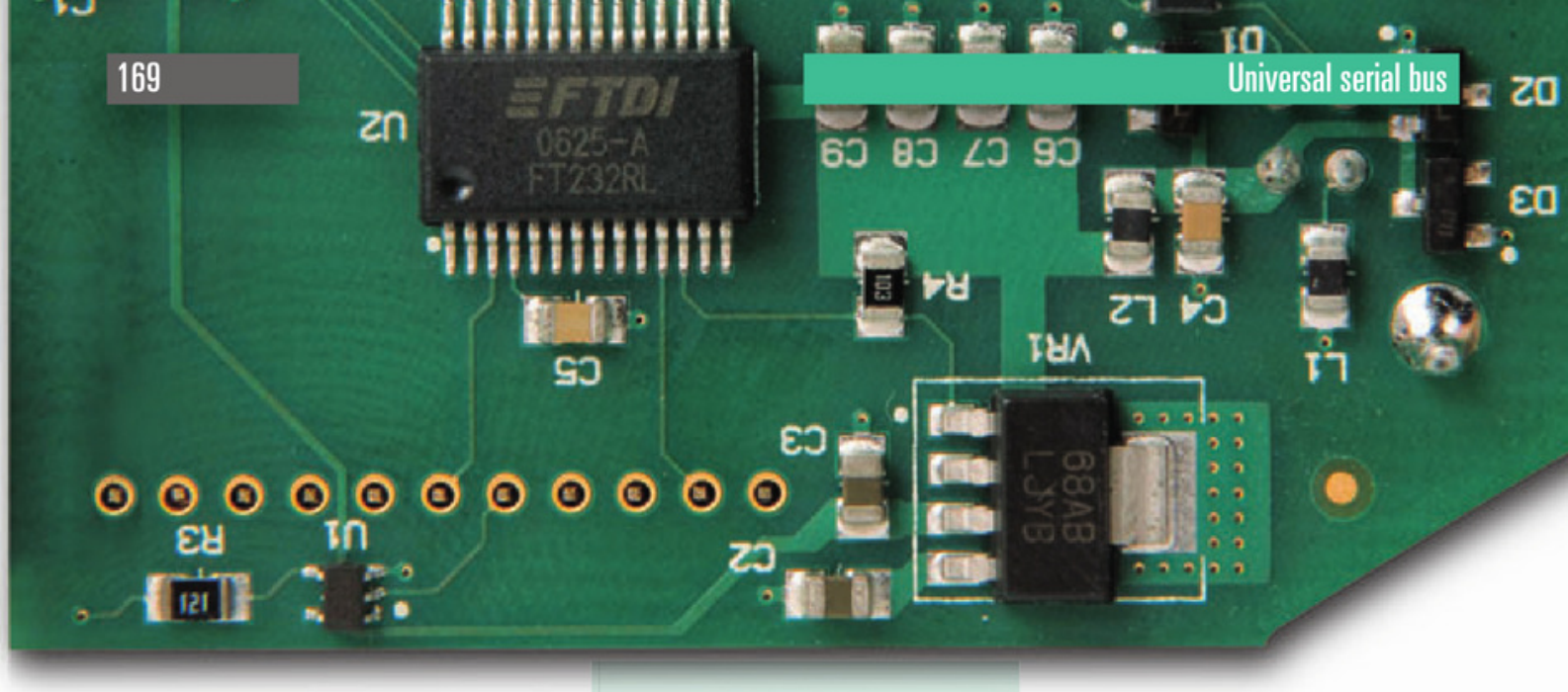
- Una interfaz full-speed USB-UART (asíncrona serial) a través del módulo hardware FT232BM embebido.
- Una interfaz full-speed paralela USB-FIFO a través del módulo hardware FT245BM embebido.
- Una interfaz serial sincrónica (USB-JTAG, USB-SPI o USB-I²C) a través del módulo hardware configurable: MPSSE (*Multi-Protocol Serial Engine Interface*).
- Una interfaz paralela compatible con un bus microcontrolador 8051 (MCU *Bus Host Emulation*).
- Modos bit-bang sincrónicos y

> TECNOLOGÍA HID

Human Interface Device es una clase de dispositivo USB especialmente diseñada para interactuar de manera directa con personas. Esta tecnología permitió innovar en dispositivos de entrada a computadoras y simplificar el procedimiento de instalación de estos. Fue propuesta por Van Flandern de Microsoft a la comitiva de USB.



> **Diagrama en bloques del chip FT2232. Observamos la interfaz con el bus USB y con la memoria EEPROM de configuración, y los dos bloques principales que implementan sus puertos virtuales.**



> El FT232B en circuito. Esta placa permite conectar un módulo ZigBee a una PC mediante USB.

asíncronos, de operación básica, donde no se necesita la conexión a una CPU externa. Se maneja directamente un bus E/S de 8 bits con el fin de controlar relays, LEDs o similares.

Drivers

Estos chips emplean drivers provistos por el fabricante y se dividen en dos opciones:

- **Virtual COM Port driver:** el dispositivo se ve como conectado a un puerto COM (RS-232). Trabaja bajo entorno Windows, y es utilizable por cualquier aplicación que emplee comunicaciones con un puerto COM.
- **D2XX Direct Driver:** para aplicaciones paralelo o en las que prevalezca la necesidad de velocidad, con funciones específicas para las aplicaciones que se comuniquen con el chip.

Algunas aplicaciones típicas son aquellas que necesitan manejar, a lo sumo, un puerto del tipo **bulk** o **isocrónico** para cada lado.
El chip provee pines de comunicación con una memoria **microwire**, en la cual es posible grabar información como **Vendor ID** (identificador del fabricante), **Product ID** (identificador del producto), si es un circuito que se alimenta del bus o tiene alimentación externa (self-powered), etc.

El FT232 provee un puerto serie vía USB. Lo conectado a este chip se ve como conectado a una UART.

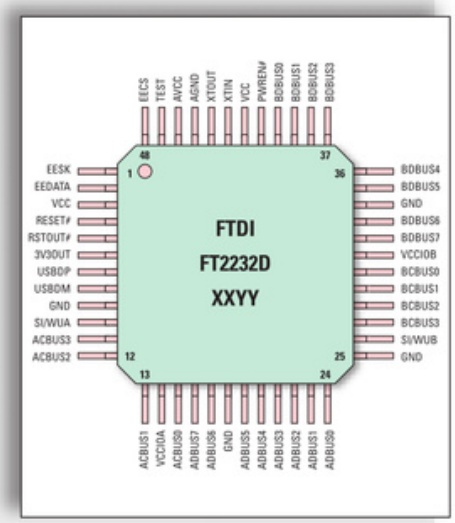


Figura 13. Encapsulado LQFP de 48 pines del FTDI2232D.

Modo FT245BM

El buffer de transmisión de esta interfaz es de **128 bytes**, en tanto que el de recepción es de 384 bytes de capacidad. La **Figura 13** muestra las funciones de los pines de la interfaz configurada en modo **FT245BM**. Esta interfaz paralela está conformada por **ocho líneas** de datos y **cuatro de control**:

- **RXF:** indica a la CPU conectada que tiene datos del host para leer.
- **RD:** es activada por la CPU para leer los datos.
- **TXE:** indica a la CPU que puede enviar datos al host si se lo requiere.
- **WR:** es activada por la CPU para escribir los datos.

Modo FT232BM

Este módulo convierte una transferencia USB en una del tipo serial asíncrona. Los tamaños de los buffers son exactamente los mismos que para el caso anterior. La velocidad de transferencia máxima que se puede alcanzar es de **300 Kbps** con un bit de **start** y uno de **stop**.
En la **Figura 13** se muestra la distribución y la función de los pines para una interfaz configurada en este modo. Las líneas implementadas son las correspondientes a una terminal DTE (*Data Terminal Equipment*), definidas en el estándar EIA/TIA-232, en tensiones de uso en lógica. ■

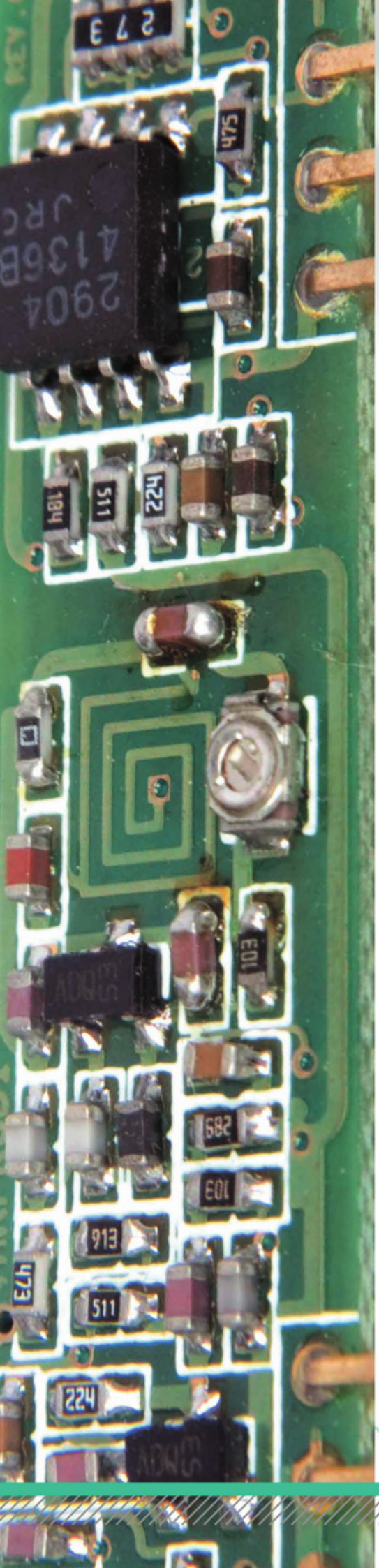
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Mencione algunos protocolos de conectividad.
2. ¿Qué es el estándar RS-232?
3. Caracterice al puerto serie.
4. ¿Qué es SPI?
5. Enumere los modos SPI.
6. ¿Para qué sirve el bus de comunicación I2C?
7. ¿Qué es Docklight y para qué sirve?
8. Enumere las características de Universal Serial Bus.
9. Mencione los tipos de transferencias definidas para una comunicación USB.
10. ¿Qué es la tecnología HID?

EJERCICIOS PRÁCTICOS

1. Genere una tabla que contenga los protocolos de conectividad no inalámbrica.
2. Realice la conexión de una memoria SPI.
3. Conecte un periférico I2C.
4. Instale y ejecute la aplicación Docklight.



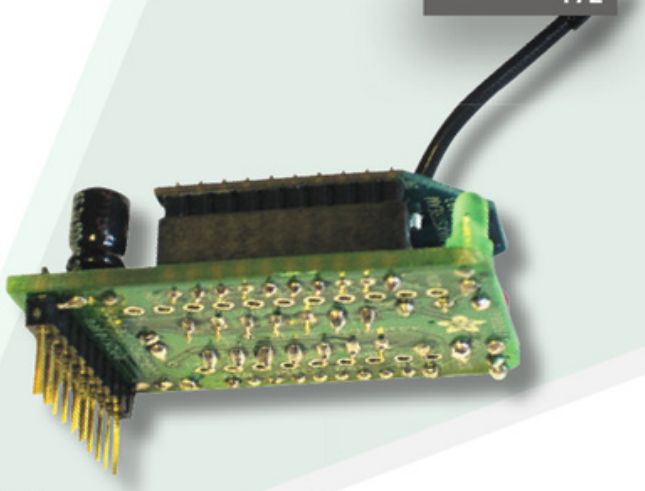
CAPÍTULO 6

CONECTIVIDAD INALÁMBRICA

Alternativas de conectividad
ZigBee y 802.15.4
Módulos prearmados (XBee y XBee ZB)
Conexión con la PC y configuración
Red de sensores ZigBee
Control por RF
Control por infrarrojos
Módulos prearmados (Kcwirefree)
Actividades

06

ALTERNATIVAS DE CONECTIVIDAD



Existen numerosas opciones de conectividad inalámbrica, pero la mayoría responde a un propósito diferente. Veamos cuáles son.

Hemos mencionado algunas opciones a lo largo de este libro y, seguramente, ya conocemos otras. Las razones de esta proliferación están en la búsqueda de la satisfacción de distintas necesidades por caminos diferentes. Empecemos por diferenciar entre simple **conectividad** y **networking**.

Por **conectividad** entendemos la posibilidad de **establecer una conexión** o comunicación entre dos dispositivos, de modo que puedan **dialogar entre sí**. Con **networking**, nos referimos a la interconexión de **varios dispositivos en red** soportando un stack de protocolos robusto que permita comunicar varias y diversas aplicaciones. Hagamos, entonces, el análisis desde este ángulo.

Tecnología Wi-Fi

La necesidad que da origen a **Wi-Fi** es mantener el esquema y la velocidad de una red, pero sin tener que recurrir a cables. Como tal, se la puede considerar una especie de reemplazo de los sistemas **Ethernet**. Los dispositivos ya implementan un stack de protocolo, generalmente, **TCP/IP**; lo que hace Wi-Fi es reemplazar la parte física, es decir, la de conexión al medio de comunicaciones.

Existen muchas opciones de conectividad, y todas apuntan a satisfacer distintas necesidades.

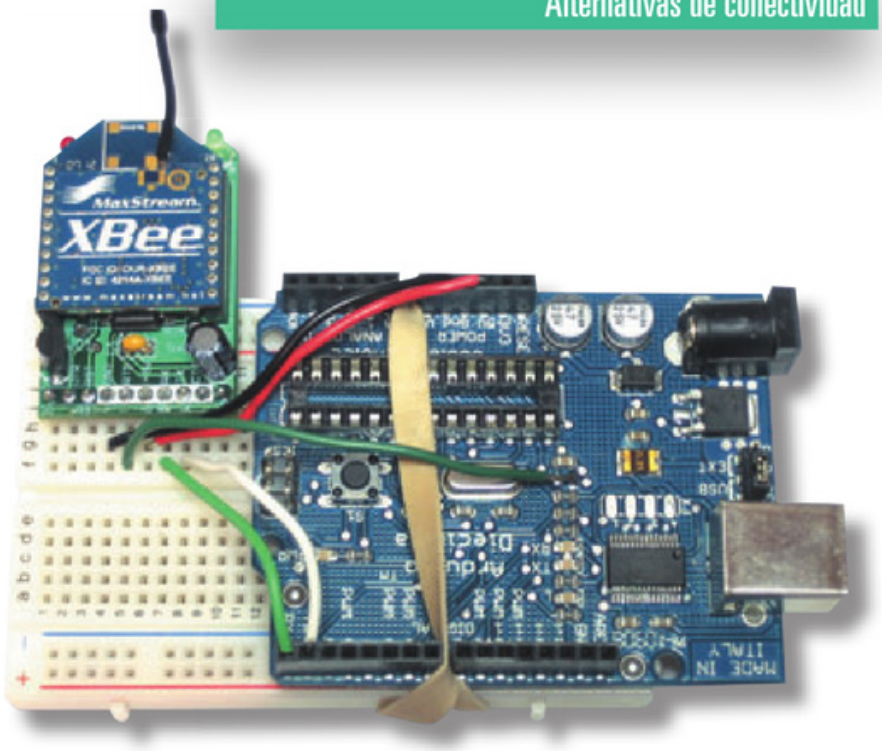
En este entorno, no nos sorprende que las velocidades de transferencia sean altas (del orden de los 50 MBps), el consumo resulte elevado, y el stack, complejo. En particular, lo que se refiere a encriptación requiere un hardware con una potencia de procesamiento elevada.

Bluetooth

Este stack de protocolo nace para mantener la conectividad que requiere un equipo respecto de sus periféricos,



> Podemos observar un robot que incorpora la tecnología Wi-Fi 802.11b/g. También se pueden ver estos dispositivos con Bluetooth o GSM.



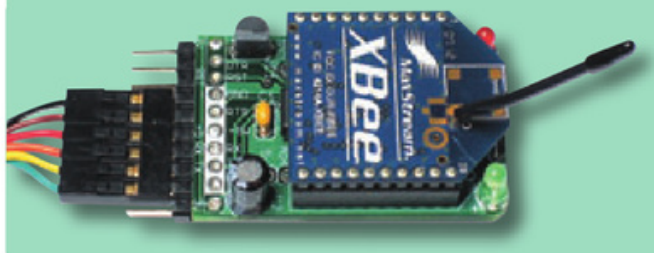
sin depender de los cables. Como tal, podemos pensar en él como un reemplazo de **USB**. En particular, el equipo central suele ser un celular o una **PDA** (*Personal Digital Assistant*, o asistente personal digital), de modo que el requerimiento es mantener la libertad de operación del individuo brindándole servicios de control, conexión y audio.
El stack resultante tiene una complejidad menor a la de **Wi-Fi**, pero suficiente como para requerir un procesador importante. No se pretende que haya conexión de periféricos entre sí, pero sí autodetección y una velocidad como para que varios periféricos puedan hacer su trabajo. El consumo es menor, pues los dispositivos involucrados funcionan con baterías, que el usuario debe recargar periódicamente.

ZigBee y 802.15.4

En este caso, el escenario es diferente. El origen de **802.15.4** puede asociarse a la necesidad de conectar sensores remotos a un sitio central que recolecta la información. Como tal, podemos considerarlo un reemplazo de **RS-485**, utilizado en conexiones multipunto cableadas. El factor que tiene más peso en las decisiones, en este caso, es el consumo, pues la idea es que los sensores operen con baterías y no es factible recolectarlas para recargarlas.

ZigBee RF4CE es una implementación particular del stack para formar redes de control remoto.

De modo similar, se trata de entes que envían un mensaje de forma esporádica, por lo que no se pretende que haya gran velocidad de comunicación. Sí se espera que haya muchos de estos entes en la red, por lo que se prioriza el hecho de evitar las colisiones en el acceso al medio de comunicaciones. Los sensores suelen ser dispositivos simples, por lo que el stack de protocolo también tiene que serlo. Sobre esta base se apoya ZigBee, proveyendo, además, la extensión de la conectividad mediante la repetición de los mensajes por dispositivos llamados routers (routing), y un conjunto de especificaciones para soportar aplicaciones distribuidas con interoperabilidad entre distintos fabricantes. El objetivo es proveer a estas aplicaciones de un marco de operación. Por ejemplo, ZigBee brinda los códigos necesarios para implementar dispositivos dedicados que realicen tareas de automatización del hogar. Estos marcos de operación se denominan perfiles. ■

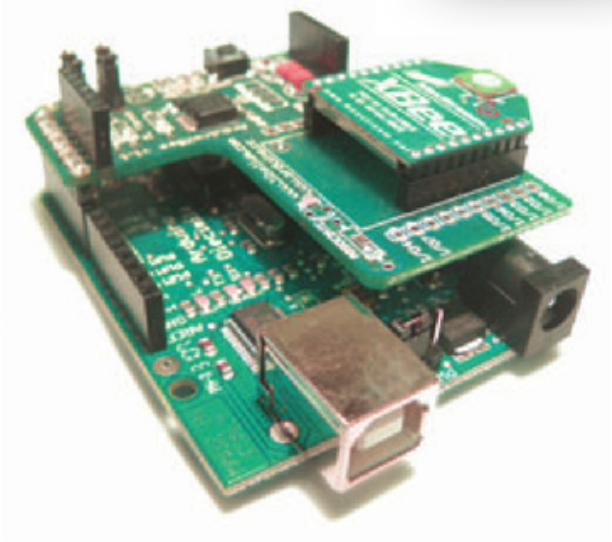


> ALTERNATIVAS A ZIGBEE

Utilizando 802.15.4 como base, o no, existen muchas alternativas a ZigBee, en particular, en aquellos casos en que solo interesa el routing. Por ejemplo, Microchip ofrece **Mi-Wi**, Texas Instruments tiene **SimpliciTI**, Digi brinda **DigiMesh**, y en especial, existe un protocolo llamado **ANT**, diseñado para mantener el mínimo consumo posible.

ZIGBEE Y 802.15.4

De gran difusión y amplias prestaciones, estos estándares abren camino a la comunicación entre dispositivos. Veamos qué son y cómo utilizarlos.



Describiremos a continuación los estándares de comunicaciones inalámbricas personales **802.15.4** y **ZigBee**. Gracias al dictado de estas normas, la industria pudo fabricar dispositivos inalámbricos capaces de dialogar entre sí.

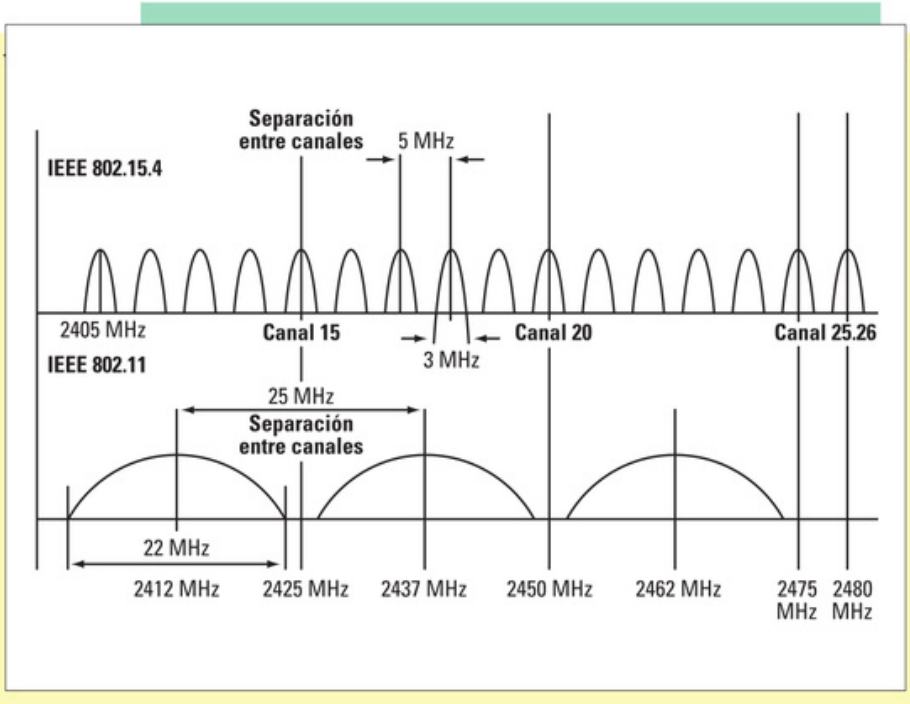
802.15.4

Este estándar está destinado a regular el acceso al medio de comunicaciones para redes personales inalámbricas. Como veremos en el apartado correspondiente, **ZigBee** se basa en él y le agrega una mayor versatilidad para la mayoría de las aplicaciones.
El estándar **802.15.4** define una banda de frecuencias sobre la cual operar: **868 MHz**, **915 MHz** y **2,4 GHz**. Tomando como ejemplo la de 2,4 GHz, el estándar especifica **16 canales** (denominados 11 a 26), que están separados cada **5 MHz**, desde 2405 hasta 2480 MHz (frecuencia central). Por ejemplo, el canal 11 estará ubicado en 2405 MHz, y el 12, en 2410 MHz.
Contiene todos los pormenores en relación a la sincronización entre dispositivos: control de errores y selección automática del canal menos ruidoso (con menos interferencias). También es posible prefijar el canal deseado. Cada dispositivo tiene una dirección única en la red, por lo que pueden direccionarse con un número, tal como lo hacemos cuando efectuamos una llamada telefónica.

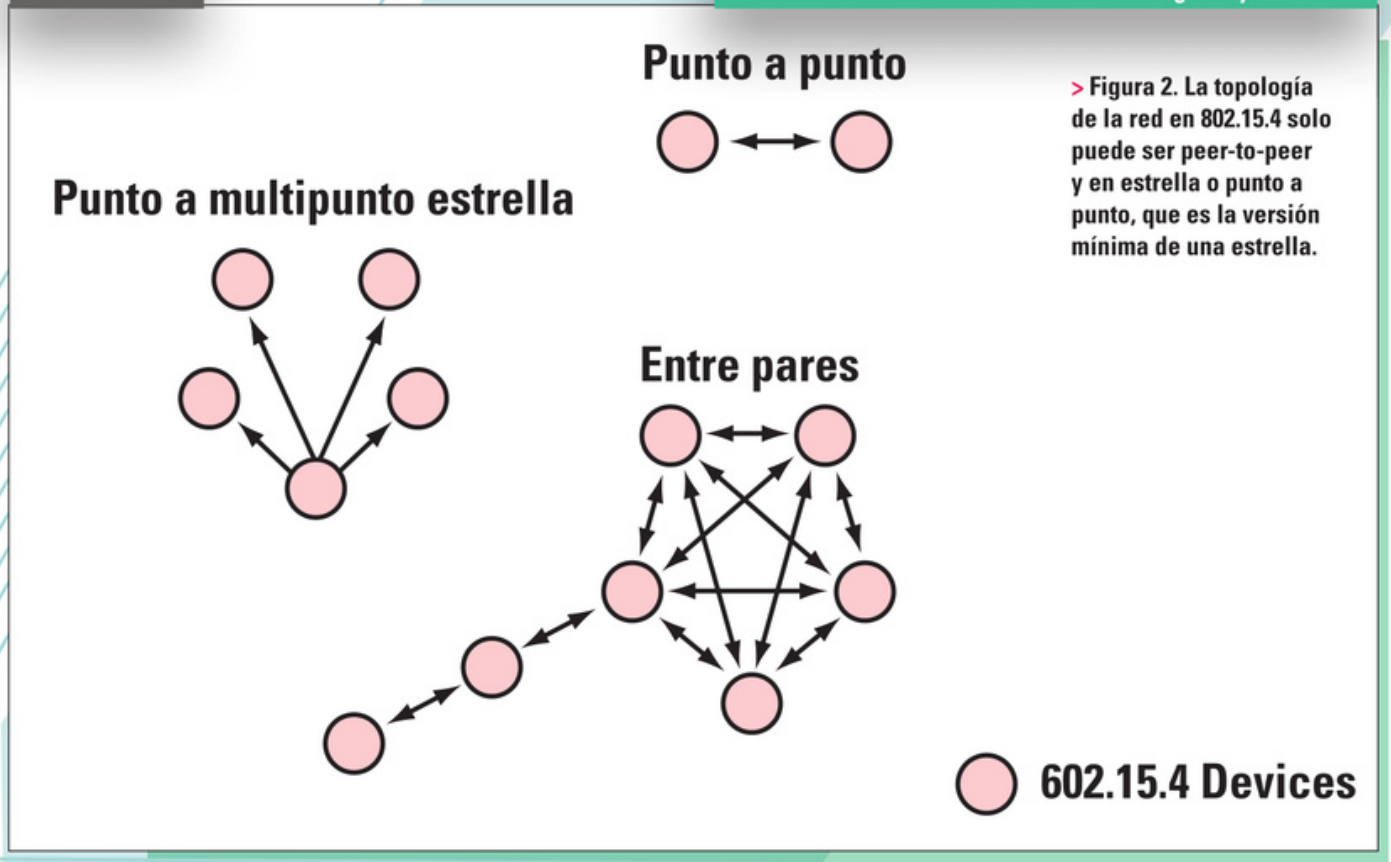
Dentro del estándar, se prevén todos los mecanismos necesarios para conocer si el receptor de un determinado mensaje lo ha recibido, ya que este responde con un código **ACK**, que es un acuse de recibo.
Un punto muy importante para considerar es que la comunicación es **half-duplex**, es decir que solo es posible emitir y recibir, pero no al mismo tiempo, como ocurre con las comunicaciones **full-duplex**. La tasa de transferencia es de **100 Kbps** en los dispositivos que operen en la banda de **868/915 MHz**, y de hasta **250 Kbps** en los que lo hacen en **2,4 GHz**.

ZigBee

Mencionamos al principio de este apartado que el estándar **ZigBee** se alza por sobre el **802.15.4**. Para comprender las diferencias entre ellos, es preciso conocer las particularidades de ambos.
El **throughput** y la latencia miden la performance del sistema inalámbrico. El primero, también llamado velocidad efectiva,



> **Figura 1. Distribución de los canales de 802.15.4 y ZigBee. Notemos cómo se comparte la misma banda con el estándar 802.11g (Wi-Fi).**



> Figura 2. La topología de la red en 802.15.4 solo puede ser peer-to-peer y en estrella o punto a punto, que es la versión mínima de una estrella.

es la cantidad de información que podemos enviar por unidad de tiempo a través del canal; mientras que el segundo, o demora, mide el tiempo en que esos datos llegan y son procesados por el dispositivo remoto, para producir una respuesta. Como podemos observar, a menor latencia, mejor es el throughput.

Las redes basadas en **802.15.4** permiten transmitir de forma constante a **115.200 bps** durante cortos períodos de tiempo. Dado que ZigBee es una arquitectura que extiende el estándar **802.15.4**, el throughput resultante será menor, ya que la latencia introducida será más alta debido al procesamiento adicional que se hace sobre los datos, como veremos seguidamente.

Una red ZigBee permite conectar dispositivos sin comunicación directa.

En las redes 802.15.4 solo es posible la comunicación directa entre los dispositivos. Es decir, no se puede dialogar entre dos dispositivos que no sean visibles entre sí, aunque exista uno intermedio que vea a ambos. En resumidas cuentas, una red 802.15.4 no permite hacer routing (encaminamiento) entre dispositivos que no sean visibles. ZigBee nace como consecuencia de esta limitación, como de otras también, y da la posibilidad de comunicar dispositivos que no se ven entre sí, ni están al alcance uno del otro, empleando otros intermedios que cumplirán el papel de routers.



> ROUTING EN ZIGBEE

ZigBee incorpora el concepto de ruteo: los mensajes viajan de nodo en nodo hasta llegar a destino. Cada nodo conoce a sus vecinos, y si recibe un mensaje que no es para él, les pregunta a ellos cómo llegar al destinatario. El proceso se repite hasta alcanzar al receptor del mensaje, y la información acerca de cómo llegar viaja por el camino inverso. A partir de aquí, ya se pueden enviar mensajes.

Debido al routing, una red ZigBee permite tener una red múltiplemente conexa. Es decir, pueden comunicarse dispositivos directos y distantes sin que haya comunicación radial entre sí, efectuando saltos intermedios en otros componentes ZigBee hasta llegar a destino.
En un dispositivo ZigBee, el stack o conjunto de aplicaciones de protocolos que corren dentro del procesador del dispositivo para gestionar y controlar la comunicación es más complejo que el de un dispositivo 802.15.4. Por lo tanto, uno ZigBee requiere mucha más memoria, y microprocesadores o microcontroladores más caros.

> COMUNICACIÓN DE APLICACIONES

Basándose en el esquema de routing, las aplicaciones ZigBee se mandan mensajes entre sí, formando una gran aplicación distribuida que se encarga de resolver tareas específicas, de acuerdo con el perfil de cada una; por ejemplo, la automatización del hogar.
En nuestro caso, vamos a utilizar ZigBee para comunicar datos entre equipos remotos o con uno central.

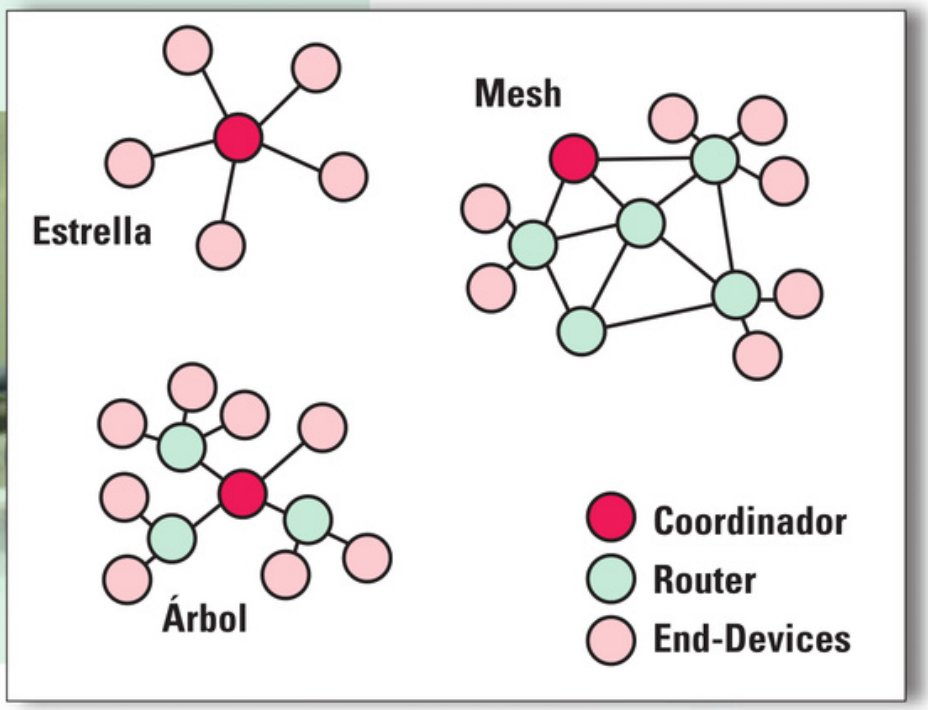
ZigBee en concreto

Una red ZigBee se basa en el estándar 802.15.4-2003 para la comunicación entre dispositivos conectados directamente entre sí. Además, define un nivel de routing mediante el cual aquellos dispositivos que actúan como router pueden transportar mensajes para otro destinatario, lo que extiende el alcance de la red. Existen tres tipos de nodos: coordinador, router y end-devices. El primero es el encargado de dar inicio a una red ZigBee, escogiendo el canal más silencioso y emitiendo una trama de datos llamada **Beacon Request**, para armar una lista de los dispositivos ZigBee encontrados. Los routers tienen la capacidad de encaminar los mensajes entre distintos dispositivos de la red y, además, de almacenar temporalmente aquellos destinados a los **end-devices**, que están durmiendo en bajo consumo hasta que despierten. Los end-devices son dispositivos con funcionalidad reducida, ya que son los encargados de recibir o enviar los mensajes propiamente dichos a los sensores, actuadores o microcontroladores que deseamos manejar en forma inalámbrica. ■

El coordinador es el nodo que inicia y controla una red ZigBee.



> **Figura 3. Dada la presencia de routers, es posible establecer una topología en árbol o mesh, donde un end-device no es directamente visible para otro que quiera enviarle un mensaje.**



MÓDULOS PREARMADOS (XBEE Y XBEE ZB)

Existen módulos que implementan internamente los stacks 802.15.4 y ZigBee, con lo cual reducen la complejidad a simples operaciones a través de un puerto serie.



> **Módulo XBee-PRO con antena externa. Es más potente y cuenta con un receptor más sensible, con lo que logra distancias superiores.**

Si bien podemos diseñar y construir nuestros propios módulos 802.15.4 o ZigBee ajustándonos a las especificaciones del estándar, lo cierto es que este es un trabajo muy complejo, ya que involucra diversos elementos y alta especialización, en particular, en RF (radiofrecuencia). Por este motivo, existen módulos ya armados y listos para funcionar.

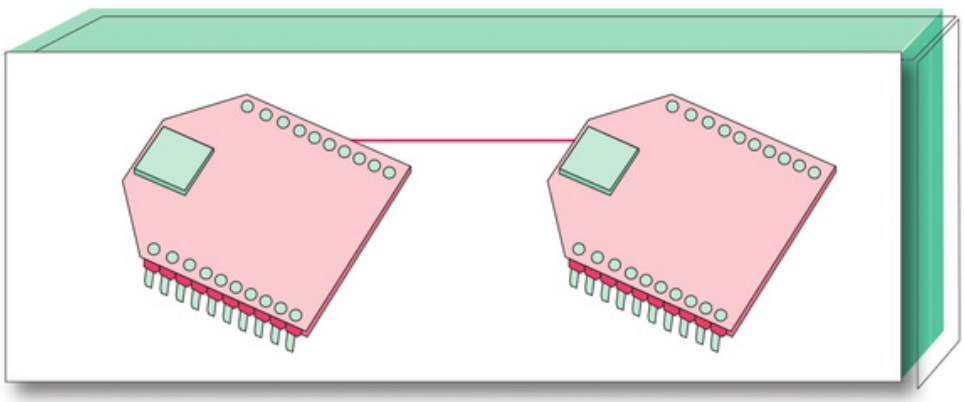
XBee 802.15.4

Por medio de este módulo, podemos crear una red 802.15.4. El XBee 802.15.4 tiene una potencia de transmisión de **1 mW** y una sensibilidad del receptor de -92 dBm, por lo que podremos esperar un alcance de hasta 100 metros en espacios abiertos y de 30 metros en zonas urbanas. Existe otra versión de mayor potencia denominada **XBee –PRO 802.15.4**, de **60 mW** y una sensibilidad del receptor de **-100 dBm**, que alcanzan una distancia de hasta 1000 metros y 300 metros, respectivamente.

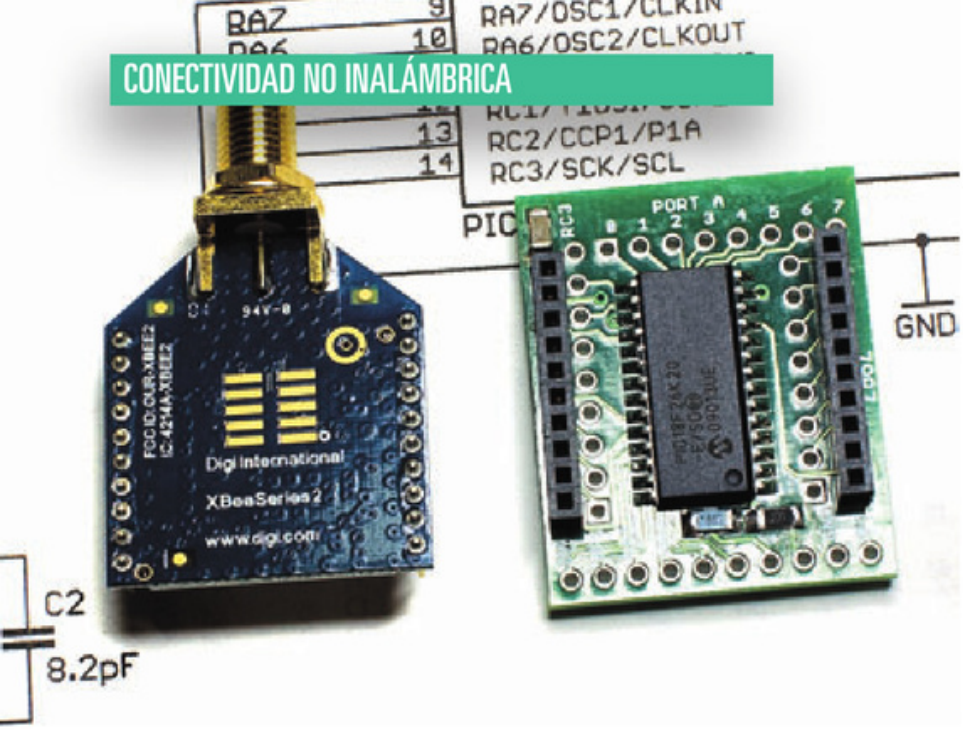
Conexión punto a punto

Realizar una conexión punto a punto es una tarea muy simple, porque los módulos ya vienen con la configuración predeterminada. Lo único que tenemos que hacer es conectar ambos, uno en el extremo remoto y otro al control central, que debe ser capaz de conectarse al módulo por medio de una **interfaz UART**. De esta manera, los módulos operan en forma transparente; es decir, los datos que se reciben por el puerto serie de un módulo son transmitidos al módulo remoto a la salida de su puerto serie y, recíprocamente, el remoto, al central. Es conveniente especificar las direcciones concretas de los módulos para evitar conflictos con otros que agreguemos. Para esto, configuramos los parámetros **MY=<dirección propia>** y **DL=<dirección remoto>** antes de dar inicio a la red. Por ejemplo:

Módulo 1
MY=1234
DL=5678
Módulo 2
MY=5678
DL=1234



> **Figura 4. Esta es la manera de interconectar dos módulos punto a punto. Con los valores de fábrica, es posible alimentarlos y ponerlos a funcionar de inmediato.**



Punto a multipunto con coordinador

Un coordinador es un dispositivo que opera como central de toda la red y que tiene la capacidad de almacenar hasta dos mensajes en espera de que los módulos correspondientes despierten para poder entregárselos. Esto soluciona el problema de consumo en las redes punto a punto, en las que ningún módulo puede entrar en el modo de bajo consumo. El número máximo de dispositivos dependerá del tráfico de la red.

Para crear una red multipunto con coordinador, debemos configurar el módulo que será coordinador con el parámetro **CE = 1**, mientras que en todos los remotos deberá estar configurado **CE = 0**.

Al iniciar la red, cada remoto manda mensajes para descubrir a un coordinador y asociarse a él. El coordinador le asigna una dirección al remoto. Además, en los remotos debe configurarse el **bit 2** en el parámetro **A1**. Para que el coordinador asocie remotos que se lo solicitan, hay que configurar el **bit 2** del parámetro **A2** en el coordinador.

La unidad central o computadora o microcontrolador que esté conectada al coordinador necesitará saber de qué remoto provienen los mensajes. Para esto, hay que agregar al mensaje de los remotos algún dato que los identifique, o utilizar un modo especial y documentado en las especificaciones del módulo, llamado modo **API**.

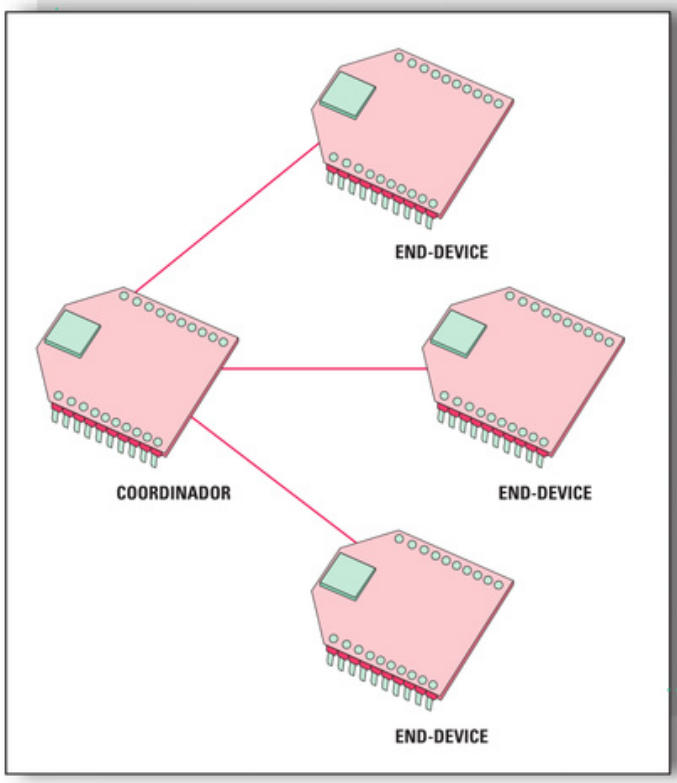
Para enviar un mensaje desde el coordinador hasta un remoto, tenemos que cambiar el parámetro **DH-DL** del coordinador por la dirección del remoto en cuestión. Los remotos siempre transmiten al coordinador.

El problema de consumo en las redes P2P se soluciona con la existencia de un coordinador.

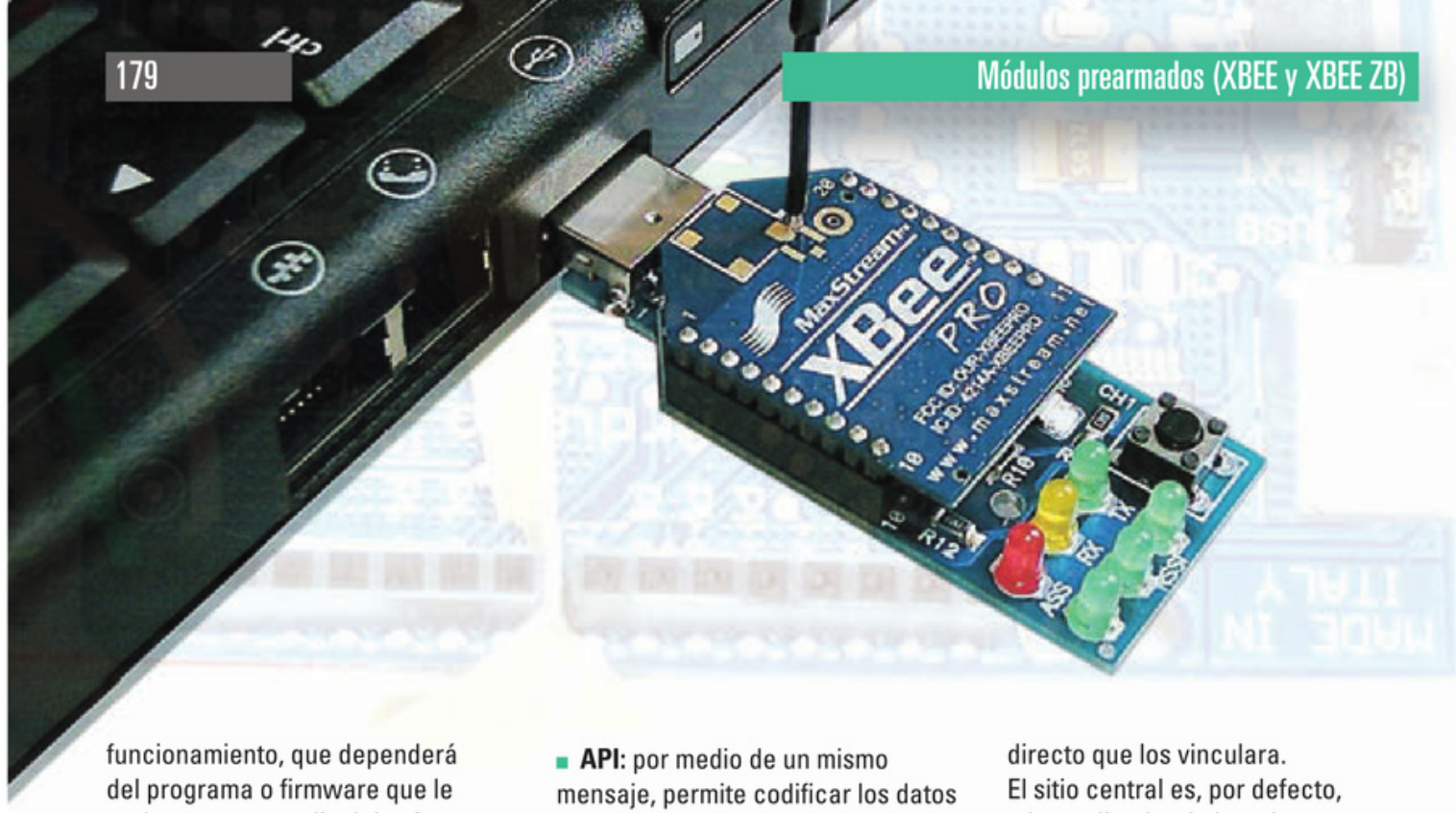
Módulos XBee ZB

Los módulos **XBee ZB** permiten crear una red ZigBee. Están disponibles en distintas potencias, y cuentan con entradas y salidas digitales, y con entradas analógicas, para efectuar tomas de mediciones reales en la entrada del conversor A/D de que disponen. También tienen una salida **PWM** que permite leer la intensidad de la señal de recepción de acuerdo con su ciclo de trabajo (al igual que el XBee 802.15.4), entre otras utilidades, según la configuración del módulo.

El módulo puede configurarse para un determinado modo de



> Figura 5. Conexión punto a multipunto con coordinador. En esta configuración, todos los end-devices se dirigen al coordinador, el que conocerá las direcciones de todos los módulos y almacenará hasta dos mensajes en espera de que el módulo direccionado despierte.



funcionamiento, que dependerá del programa o firmware que le grabemos, por medio del software gratuito **X-CTU.** A través de él, también vamos a ajustar los parámetros que sean necesarios, de una forma sencilla. Los modos de funcionamiento son dos:

- **Transparente + AT:** el módulo envía y recibe los datos de los remotos por medio de su interfaz serie, y es controlado por los comandos AT "escapados".
- **API:** por medio de un mismo mensaje, permite codificar los datos y comandos de una forma sencilla, en una única interfaz serie.

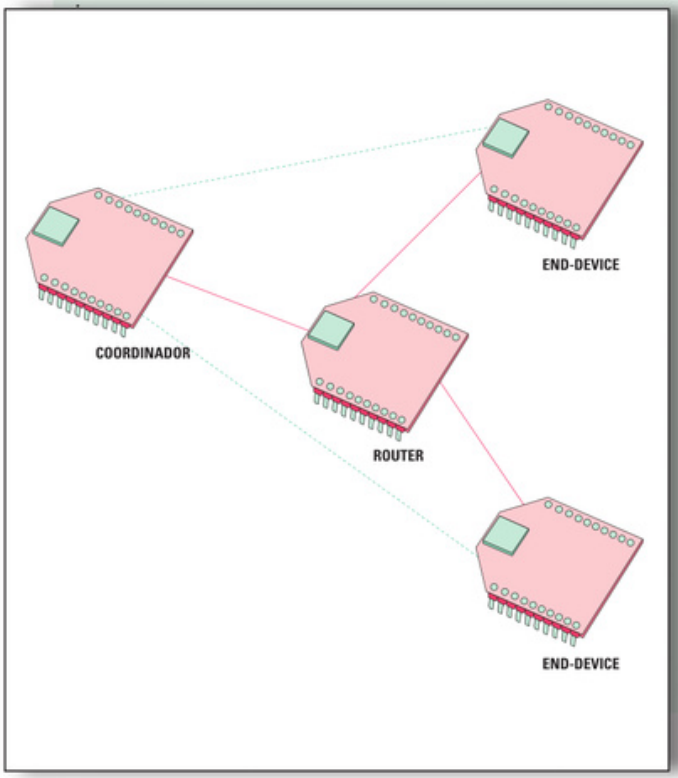
Comunicación a un sitio central

Este es el modo predeterminado al cargar el firmware AT y que opera de forma transparente. Es decir, une de manera inalámbrica los puertos serie de los módulos remotos con el de un sitio central, y obtiene los datos como si se tratase de un cable directo que los vinculara. El sitio central es, por defecto, el coordinador de la red.

Conectarse con los módulos XBee

Las opciones de conexión con los módulos XBee, teniendo en cuenta que el módulo es de 3 V, son las siguientes:

- Conexión directa al micro en los pines de una UART.
- Conexión a una PC o a otro dispositivo por RS-232.
- Conexión a una PC por un puerto serie USB.



> Figura 6. Una red XBee ZB es full-mesh. Los routers y el coordinador descubren la dirección hacia el destinatario. Si los end-devices están asociados a un router, los mensajes desde el coordinador que está "detrás" del router no siguen la línea punteada, sino la sólida.

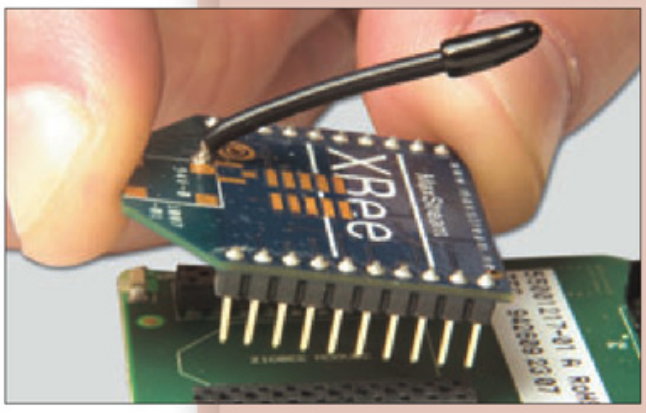




CONEXIÓN CON LA PC Y CONFIGURACIÓN

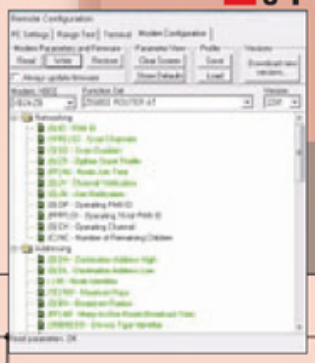
La interfaz de conexión con la PC nos permitirá configurar los módulos XBee, actualizar el firmware y utilizarlos para recibir la información enviada por los módulos remotos.

01



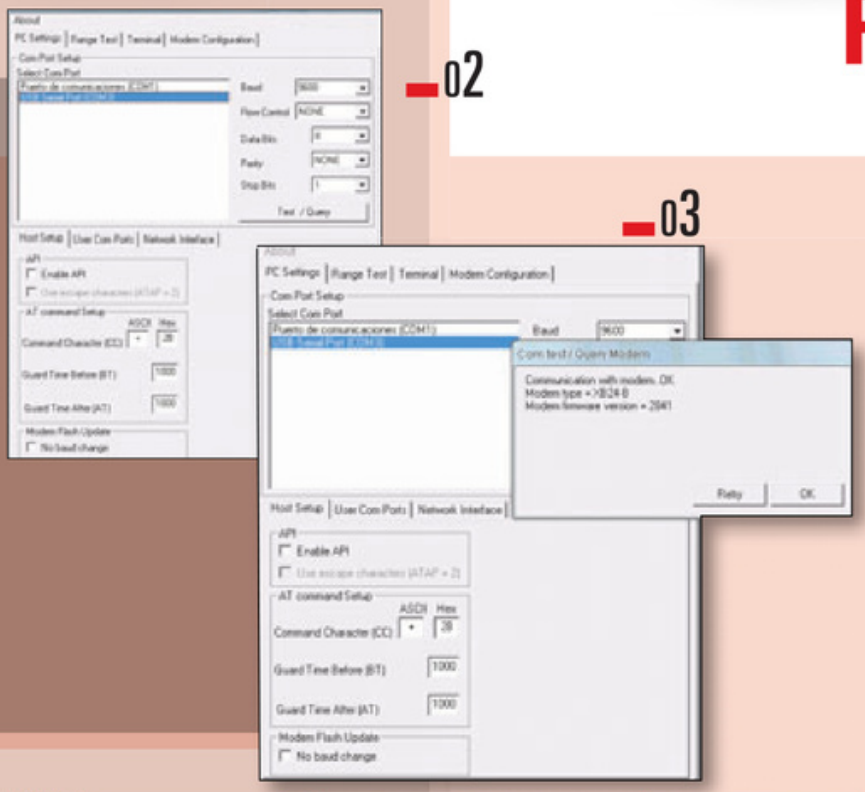


04



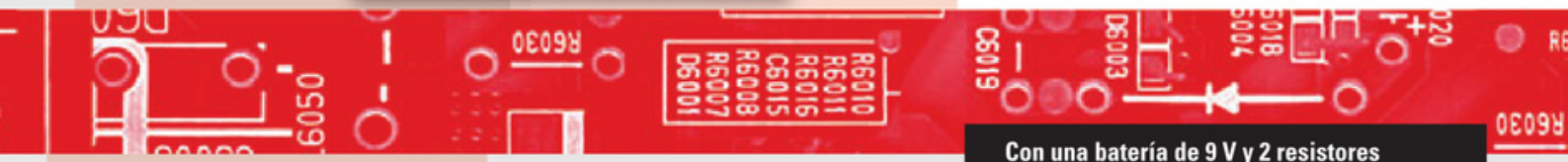
Este es el esquemático para conectar los módulos XBee a la PC mediante un puerto serie USB.

PASO A PASO

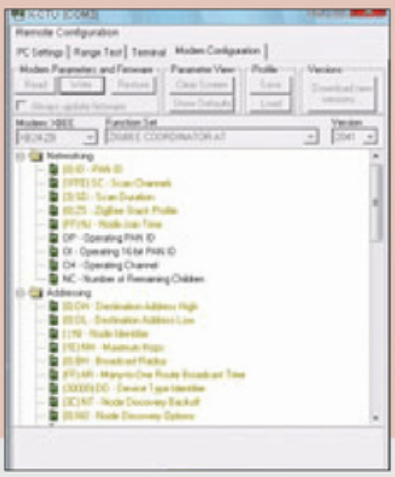


01. El FT232 viene en SMD, y los módulos tienen pines de paso de 2 mm, que requieren un zócalo. La hembra de pines de 2 mm viene en formato doble, ya sea SMD o de inserción. Como siempre, debemos ubicar los capacitores de desacople lo más cerca posible del chip correspondiente: regulador, FT232, XBee. Insertamos el módulo XBee en el zócalo, teniendo cuidado de hacer coincidir los pines correctamente.





Con una batería de 9 V y 2 resistores de 3.3 KO, podremos armar este circuito.

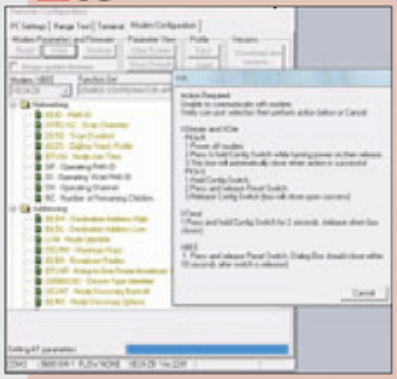


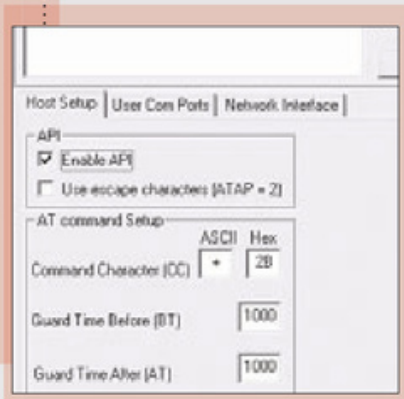
02. Instalamos el programa X-CTU, desde el CD que acompaña a la colección o yendo a la página web del fabricante: www.digi.com. Conectamos el puerto USB al hardware de interfaz y corremos el programa. Seleccionamos el puerto serie USB.

03. Presionamos el botón **[Test/Query]**; se abrirá una ventana que nos indicará la revisión de firmware del módulo. De no ser así, debemos revisar las conexiones con el módulo.

04. En la solapa **Modem Configuration**, podemos observar la configuración del módulo. Para obtenerla, presionamos el botón [Read]. Realizamos cambios operando sobre la ventana, y lo que cambiemos se verá en azul. Modificamos el valor de JV (*Channel verification*) a 1 (*Enable*). Guardamos lo hecho presionando el botón **Write**.

05. Uno de los módulos deberá ser coordinador. Desconectamos la interfaz, cambiamos el módulo y volvemos a leerlo. En **Function Set**, seleccionamos la opción **ZIGBEE COORDINATOR API** y, en **Version**, **[2141]**. Grabamos presionando el botón **Write**.





06. La comunicación con el firmware API requiere que se habilite este modo de operación en X-CTU. Habilitamos el modo API en X-CTU, marcando el casillero **Enable API**, dentro de la solapa **Modem configuration**.

RED DE SENSORES ZIGBEE

Hasta ahora hemos visto la teoría para conectar los módulos. Ahora vamos a conocer los detalles acerca de cómo armar una red de sensores.

Una **red ZigBee** debe tener un coordinador, que se encarga de iniciarla y darle un identificador. Este, además de un router, puede derivar mensajes a otros dispositivos y ser padre de **end-devices**, dispositivos que pueden estar en bajo consumo, con el receptor apagado. Puede haber otros **routers** y **end-devices**; estos últimos los pondremos en sitios alejados, para tomar mediciones. En esta red utilizamos, en lo posible, la configuración por defecto.

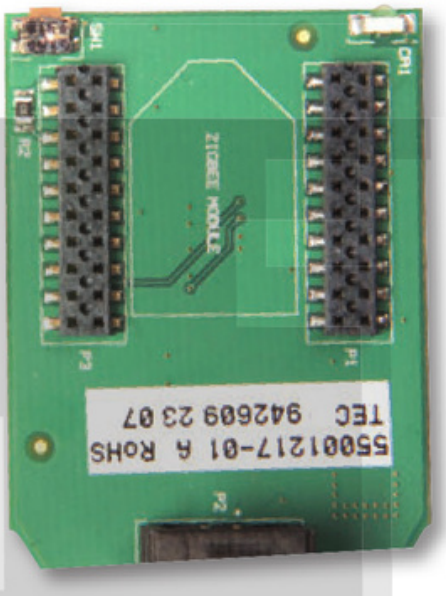
El coordinador

El coordinador de la red será, en nuestro caso, el sitio central donde recibiremos los reportes. Tendrá **firmware API**, para reportar los mensajes que reciba, indicando la dirección de quien los manda y los detalles de la medición. Su dirección es por defecto 0, que coincide con el destino prefijado para transmitir en los otros dispositivos. Si no utilizamos **routers**, no debemos configurar nada más. En caso contrario, aplicamos los cambios que hagamos en los routers, también al coordinador. Recordemos que, cuando configuramos el coordinador, tenemos que habilitar el modo API marcando el casillero **Enable API,** en la solapa **Modem Configuration**, y revertir esta acción cuando pasamos a configurar otro dispositivo.

> MODO API

El fabricante denomina **modo API** a una forma de comunicación con el módulo. En ella, en vez de comandos AT y datos transparentes, se utiliza una trama de información con encabezado, checksum e identificadores. Esto facilita la operación de envío de mensajes, pues la dirección del destinatario se entrega junto con el mensaje. Lo mismo ocurre en la recepción.





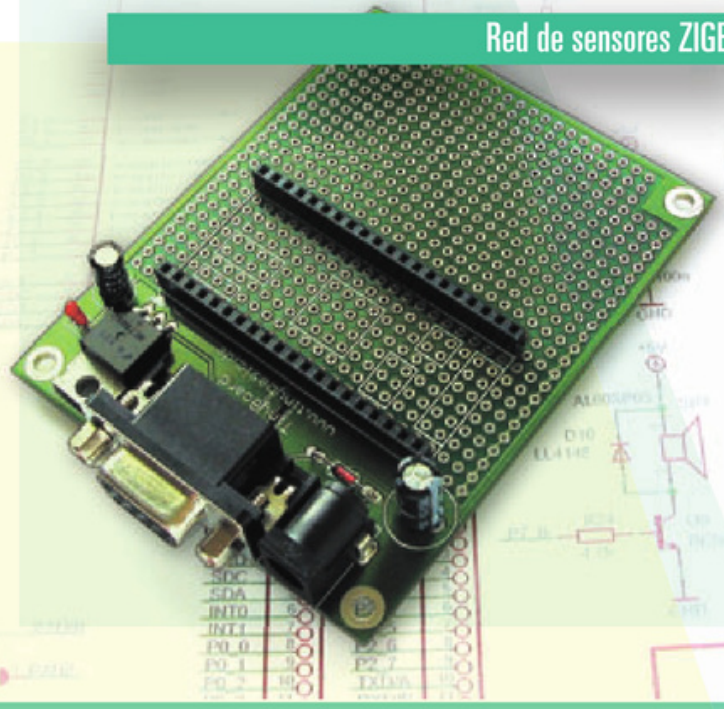
> El kit Rabbit RCM4500W incorpora un módulo XBee con interfaz USB para oficiar de coordinador de una red ZigBee.

Los routers

Los routers están siempre atentos y atienden los mensajes para los end-devices, ya que estos pueden permanecer en bajo consumo durante largos períodos de tiempo, con su receptor apagado. También recibirán sus reportes al despertar y los retransmitirán a destino. En los routers configuramos el parámetro **JV = 1** para que busquen un coordinador al arrancar. Dejamos el parámetro **ID = 0**, para que se asocien a cualquier red (de todos modos, tenemos solo una). También configuramos los parámetros **SP** y **ST** igual que lo hacemos con los **end-devices**.

Los end-devices

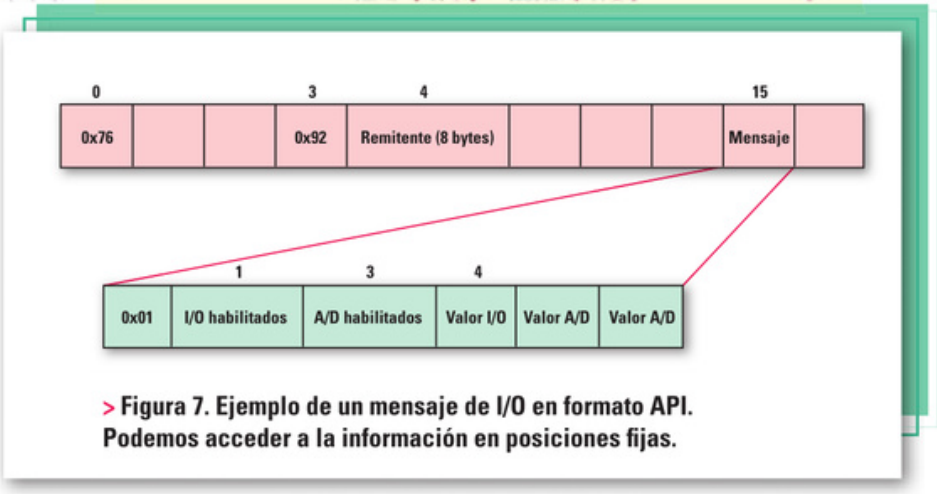
Para armar end-devices, grabamos el firmware **ZIGBEE END DEVICE AT 2841** en un módulo. Configuramos el parámetro **SP** para el tiempo en que queremos que el módulo duerma



(tiempo entre reportes).
Aquí podemos configurar los pines para que sean asignados al **conversor A/D** interno, entradas o salidas digitales, mediante los parámetros **D1** a **D4**, según lo que deseemos conectar. El parámetro **IR**, que se encuentra en la carpeta **[I/O Sampling]**, nos permite configurar el tiempo entre reportes cuando el módulo está despierto. Es decir, transcurrido **SP**, reportará cada **IR** hasta que transcurra **ST** y, luego, volverá a dormir.
Si queremos conectar un micro al puerto serie, este recibirá los mensajes que se envíen al módulo, y deberá contestar mientras se encuentre despierto.

Lectura de los reportes

El formato **API** tiene sus vericuetos, ya que está pensado para ser parte de un sistema de comunicaciones. Sin embargo, podemos leer los mensajes de manera muy simple, como si fuera una tabla. Por ejemplo:



> **Figura 7. Ejemplo de un mensaje de I/O en formato API. Podemos acceder a la información en posiciones fijas.**

- El byte en la posición 3 es el identificador de mensaje. El valor **0x90** corresponde a datos por puerto serie, en tanto que el valor **0x92** es una muestra de **I/O**.
- Los 8 bytes a continuación son la dirección del remitente.
- El mensaje se encuentra a partir de la posición 15.

En un mensaje de I/O, las entradas y salidas digitales (si hay alguna habilitada) se leen en los bytes que están en las posiciones **19** y **20.** Los canales analógicos habilitados están a continuación.
La información completa del formato del mensaje puede obtenerse de la hoja de datos del módulo, en la página web del fabricante. Vemos un ejemplo en la **Figura 7.** ■

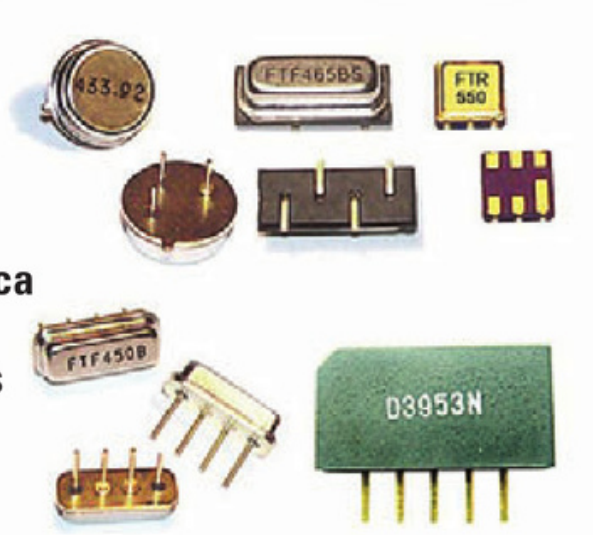
En modo API, el coordinador entrega la dirección de quien envía el mensaje.



> CONOCIENDO MÁS SOBRE API

Implementar el modo API no es difícil, pero requiere algo de experiencia en comunicaciones. En la hoja de datos del módulo hay simples ejemplos, pero podemos encontrar más información sobre 802.15.4, ZigBee y los módulos XBee en la página web www.ldir.com.ar/libros/equisbi.

CONTROL POR RF



Exploraremos ahora una opción muy económica para telecontrol. Haremos referencia a los módulos transmisores y receptores integrados de RF para comunicaciones de datos.

Si bien la radiofrecuencia (**RF**) requiere de un particular cuidado en el diseño y la construcción, existe una serie de módulos prearmados que facilita notablemente su aplicación. De esta manera, conectándonos directamente a un pin del microcontrolador, podemos dotar rápida y fácilmente a nuestro proyecto de la capacidad de controlar o ser controlado de forma remota. En el **Figura 8** vemos un esquema simplificado.

El transmisor

Los módulos transmisores de **RF** en su mayoría utilizan un esquema de modulación denominado **ASK** (*Amplitude Shift Keying*). En él, las señales de datos ocasionan que la amplitud de la portadora varíe entre dos niveles. En general, se utiliza **OOSK** u **OOK** (*On-Off Shift Keying* u *On-Off Keying*), que consiste en transmitir la portadora ante un estado lógico y anularla en el otro.

El esquema circuital de un transmisor es muy simple; por lo general, es solo un transmisor y, a lo sumo, un elemento estabilizador, como un resonador **SAW** (*Surface Acoustic Wave*, onda acústica de propagación superficial). Posee un pin para recibir los datos, y dos pines de alimentación. En algunos casos, existe un tercer pin para habilitar la transmisión.

El receptor

Los módulos receptores suelen ser circuitos **regenerativos** o **superregenerativos**, con un detector de umbral basado en un comparador a la salida. Estos receptores son circuitos con prestaciones muy por debajo de las que ofrece el más simple de los receptores **superheterodinos** (la clásica radio AM portátil).

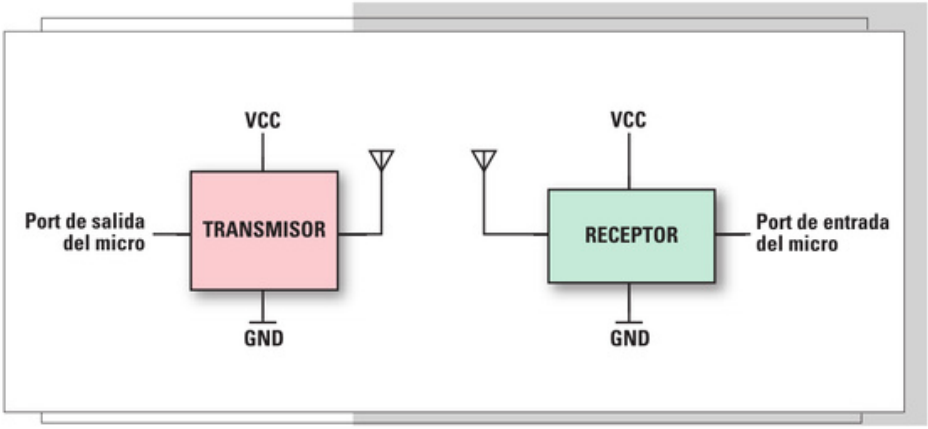


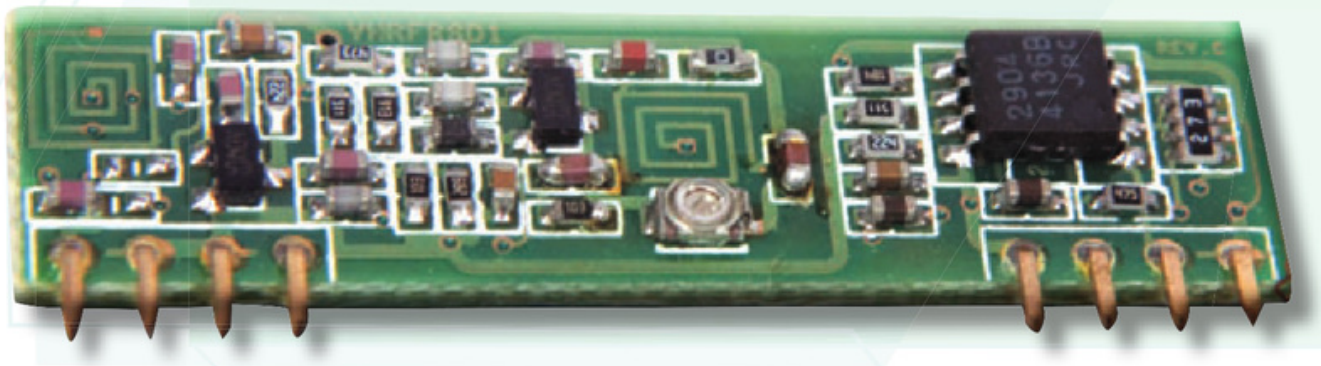
Figura 8. Uso de los módulos transmisores y receptores de RF.



> HISTORIAS DE RECEPTORES

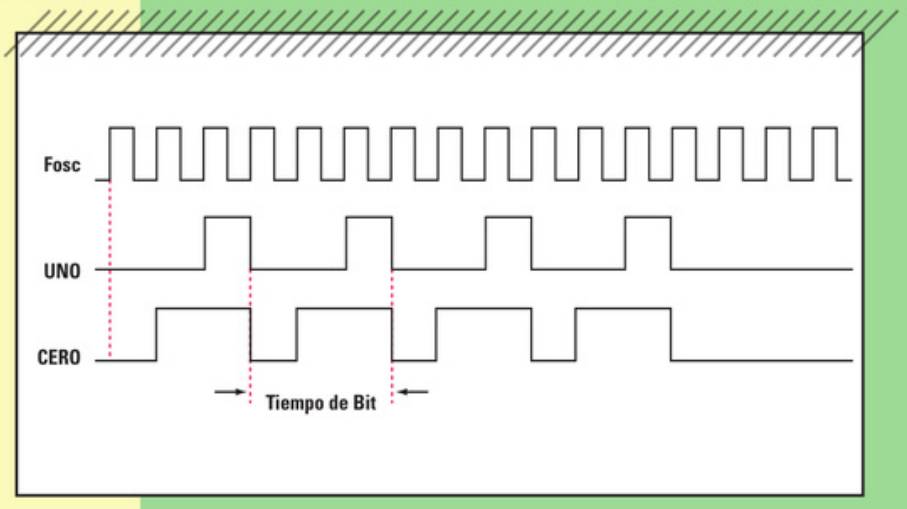
El receptor regenerativo se basa en la aplicación de realimentación positiva, formando un amplificador a punto de oscilar. Simple y económico, fue inventado en 1914 por Edwin Armstrong, quien además desarrolló el receptor superregenerativo en 1922, y el superheterodino en 1918. Este último traslada la señal a una frecuencia intermedia fija, y la amplifica con un amplificador sintonizado fijo.

> Un módulo receptor de RF. El circuito integrado que se observa es el comparador.



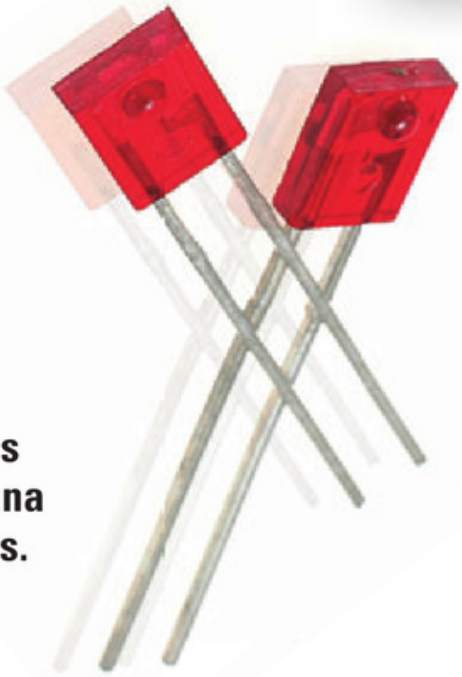
Se trata de un receptor sencillo, económico y, por lo general, sin estabilización de frecuencia. Su diseño los hace sensibles y poco selectivos, es decir, capaces de detectar señales débiles sin ser muy exigentes con la frecuencia. Estos módulos disponen de un pin que entrega la salida de datos (la salida del comparador) y de dos pines de alimentación. Un receptor de este tipo, en ausencia de señal, presenta ruido a la salida. En presencia de una portadora constante, la señal de salida indica el estado correspondiente (en general, un 1 lógico). En muchos casos, retorna otra vez al estado inactivo, debido a que estos módulos suelen estar diseñados para trabajar con señales que varían constantemente y no, para transmitir estados lógicos permanentes. Por estos motivos, no se los suele conectar directamente a una **UART**, sino que se codifican los datos, y se agrega un preámbulo al inicio de la transmisión para detectar la presencia de una señal válida frente al ruido. La codificación usada suele mantener un tiempo de pulso constante, y emplea un ciclo de trabajo para un nivel lógico y uno diferente para el otro. Como vemos en la **Figura 9**, es común aplicar la relación **1/3** para un estado y **2/3** para el otro. De este modo, se puede recuperar la información aun frente a modificaciones del ancho de pulso, lo cual es frecuente en este tipo de receptores. Para recuperar la información codificada de esta manera, podemos observar la duración del pulso y compararla con un tiempo preestablecido, u observar el estado en la mitad del tiempo de bit. Algunos módulos entregan también la salida analógica del receptor propiamente dicho, de forma que el usuario pueda discernir entre señal y ruido, o proveer un mejor detector. El rango de frecuencias utilizado suele ser la banda de **433 MHz**, destinada a este tipo de usos sin requerir licencia; aunque hay módulos para otras frecuencias. ■

Muchos codificadores para módulos de RF usan un ancho de pulso diferente para el 1 y el 0.



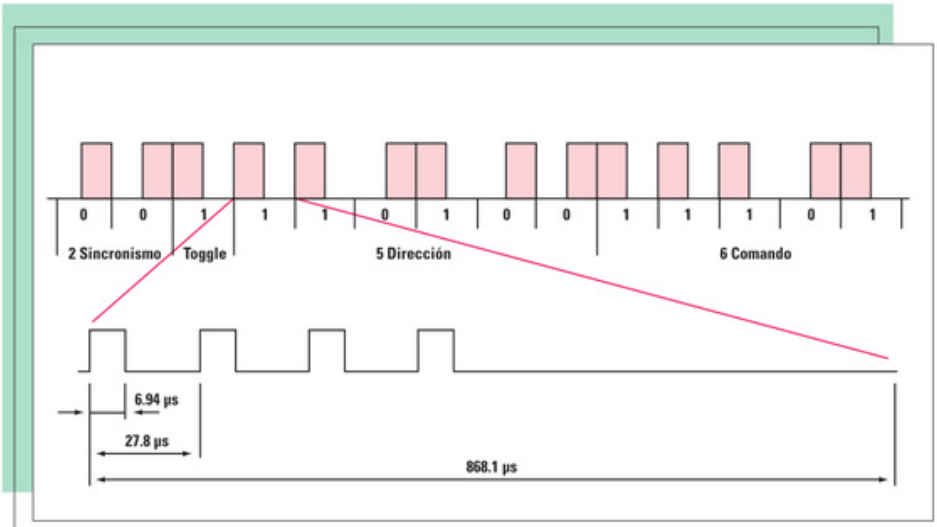
> Figura 9. La codificación de unos y ceros en un HT12E, muy utilizado como transmisor de telecontrol.

CONTROL POR INFRARROJOS



Todos tenemos en nuestros hogares al menos un control remoto, que puede usarse como una alternativa para controlar nuestros proyectos.

El control remoto por infrarrojos tiene algunos estándares de facto, como **RC5** y **RECS80**; se trata de maneras de codificar la información que han sido adoptadas por un grupo de fabricantes. El esquema general es similar al visto para los módulos de **RF**. Las diferencias se producen en la forma de codificación, los tiempos asignados al preámbulo y los bits, así como en la generación de un **anticódigo** para realizar detección de errores en el receptor. Un transmisor de control remoto envía la información así codificada pulsando la emisión de un LED infrarrojo a una alta frecuencia; las más utilizadas están entre **30** y **38 KHz**. De este modo, durante la porción activa del ciclo, se transmiten pequeños pulsos de luz, y no se transmite nada durante la porción inactiva.



> **Figura 10. El código RC5. Vemos el envío del comando 011101 a la dirección 11010.**

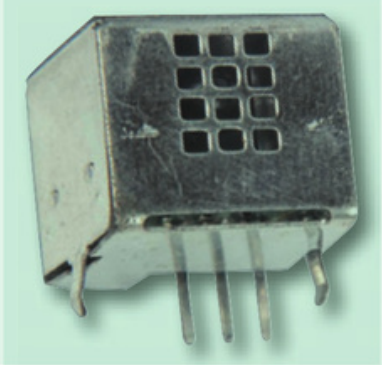
Estándar RECS80

El código **RECS80** utiliza un esquema de ancho de pulso variable. Sus características principales son:

- Un pulso inicial para sincronización que dura **10 T** (10 veces un cierto tiempo T), seguido de un silencio de 4 T.
- Los 1 son codificados con un pulso de **1 T** y silencio de **3 T**.
- Los 0 se codifican como pulso de **1 T** y silencio de **1 T**. Un valor común de **T** es **400** microsegundos.

Se envían **32 bits**, de los cuales los últimos 8 corresponden al **anticódigo**, y los 8 anteriores, al código transmitido.

> **Blindaje. Un receptor infrarrojo blindado. Todo el circuito receptor y preamplificador se encuentra dentro del blindaje.**



Estándar RC5

RC5 emplea una codificación similar a **Manchester** (utilizada por Ethernet), en cuanto a que siempre presenta transiciones en la mitad del tiempo de bit,

y el sentido de esta transición identifica al bit transmitido. Por ende, podemos observar el estado durante la primera parte del bit, antes de la transición, para recuperar el valor. La trama transmitida contiene:

- 2 bits de sincronismo (00).
- 1 bit de control (toggle, cambia cada vez que se presiona una tecla).
- 5 bits para indicar el dispositivo.
- 6 bits para el comando.

El tiempo de bit consta de 32 ciclos de la portadora de 36 KHz. Durante cada ciclo de la portadora, si se transmite, el ciclo de actividad del transmisor es de 1/5; es decir, un pulso de 6,94 microsegundos seguido de un silencio hasta totalizar 27,8 microsegundos. Podemos apreciar esto en la **Figura 10** de la página 186.

> SOBRE EL INFRARROJO

La radiación infrarroja tiene menor frecuencia que la luz visible y mayor que las microondas. Cuando se logró dividir la luz solar en diferentes colores por medio de un prisma, que separaba la luz en su espectro, se observó que en ambos extremos aparecían visibles los componentes del rojo al violeta. Al situar medidores de calor en las diferentes zonas no visiblemente irradiadas por el espectro, se comprobó que se recibía radiación por debajo del rojo. Por este motivo, se estableció el nombre de infrarrojo, que significa 'por debajo del rojo'.



>Filtro. Un receptor infrarrojo integrado. Observemos el filtro que aparece en su frente.

El código RC5 es el más utilizado entre los fabricantes de controles remotos.

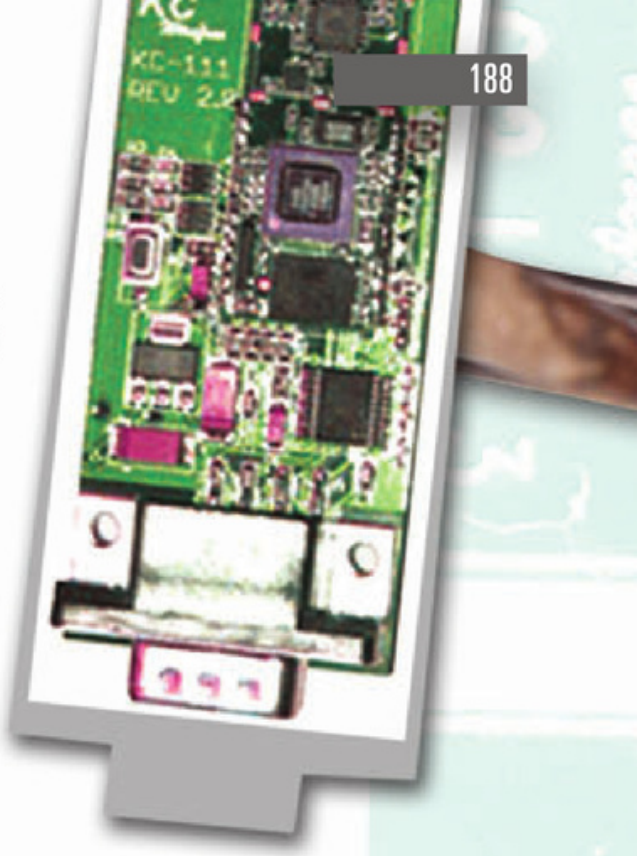
Una manera muy simple de generar estos códigos es empleando un puerto de salida de un microcontrolador. También podemos generar la portadora con un oscilador externo, al cual controlamos mediante una compuerta AND comandada por el pin del microcontrolador.

El receptor

Un receptor de control remoto posee un elemento sensible a los infrarrojos, como un fotodiodo o un fototransistor. Estos elementos generan una pequeña tensión al recibir luz, por efecto fotoeléctrico. Aun los sensibles al espectro infrarrojo tienen una cierta sensibilidad a la luz del día y, particularmente, al efecto de encendido y apagado de los tubos fluorescentes en la iluminación hogareña. Por este motivo, suelen incorporar un filtro óptico. Las señales recuperadas son muy pequeñas, por lo que los módulos incorporan un preamplificador. En muchos casos, el conjunto se blinda y se entrega de ese modo. La transmisión de la información pulsada que realiza el transmisor permite que el receptor separe fácilmente la información de la interferencia causada por otras fuentes de luz existentes. La salida de un módulo receptor es una señal discreta que puede leerse directamente por un puerto de entrada de un microcontrolador. La señal entregada es la envolvente, es decir, los bits de información. Leyendo esta señal en determinados instantes, comandados generalmente por un timer del microcontrolador, podemos extraer la información enviada y usarla para nuestros propósitos. ■

MÓDULOS PREARMADOS (KCWIREFREE)

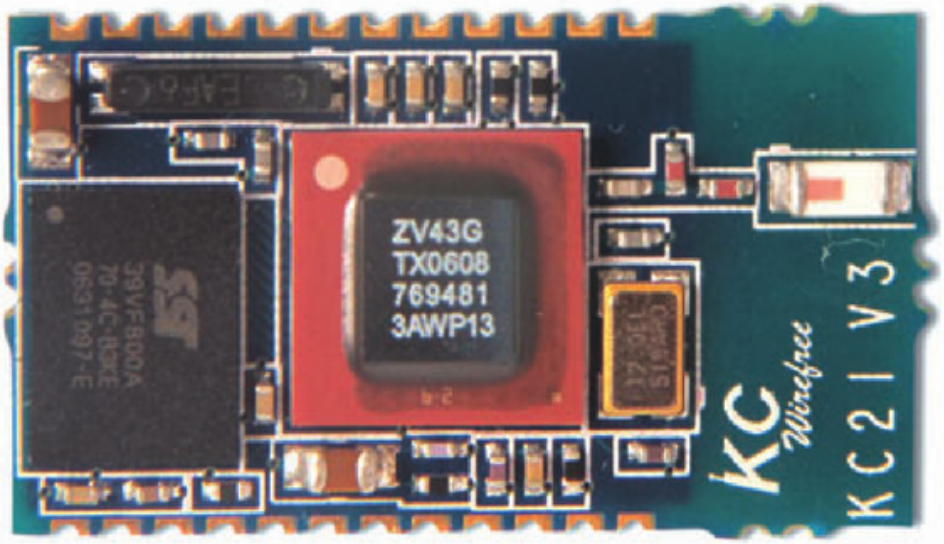
Veremos unos módulos que implementan internamente el stack Bluetooth, y que nos permiten acceder a él mediante simples operaciones a través de un puerto serie.



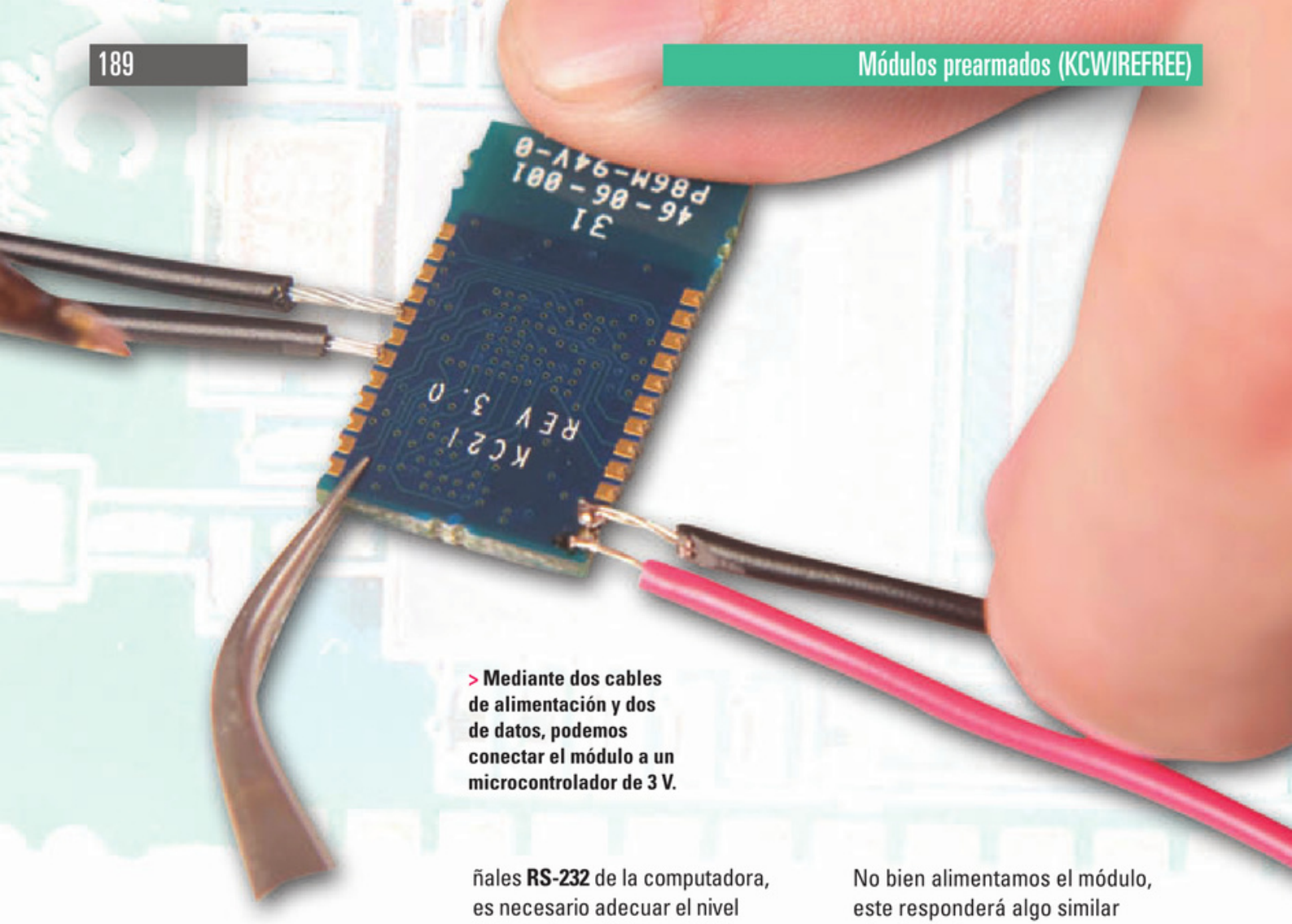
La adopción de un módulo prearmado y probado hará que nuestros proyectos sean un verdadero placer, y nos evitará tener que armarlos y depurarlos, lo cual elevaría su costo final. Para ejemplificar el uso de estos módulos, tomaremos los **Bluetooth de Kcwirefree**. Recordemos que Bluetooth permite la comunicación de datos, pero también la de voz y audio. Dado que el enlace de audio no reviste mayores inconvenientes al momento de utilizarlo –porque los módulos pueden comandarse directamente por medio de **dip-switches** (o un microcontrolador, por supuesto)–, veremos los de comunicación de datos, que nos permitirán enlazar nuestros circuitos entre sí, con una PC, un celular o una PDA. Para utilizar y controlar este módulo (al igual que cualquier otro, aunque hay excepciones, claro), se emplean unos comandos inventados por la compañía **Hayes**, llamados **comandos AT**. Esta firma desarrolló este lenguaje como un método simple de comunicación entre módems, que, por aquella época, eran de acople telefónico. En esos dispositivos, el auricular de un aparato telefónico se colocaba encima del módem mismo para captar los sonidos digitales y, así, convertirlos en datos. En la actualidad, los comandos AT son un estándar de la industria para controlar todo tipo de dispositivos inalámbricos o alámbricos, teléfonos móviles, y otros; además de seguir manejando los obsoletos módems telefónicos dial-up. Las letras **AT** provienen de ATention (atención), y le indican al módem o módulo que inmediatamente se enviará el comando propiamente dicho. Todo comando es

> PERFILES DUN Y SPP

Las características de operación en Bluetooth se agrupan en perfiles. El perfil **DUN** permite utilizar un dispositivo como módem; es decir que podemos conectarnos a Internet mediante nuestro celular. Por su parte, el **SPP** simula una conexión serie, de manera que vemos el equipo remoto como si estuviéramos conectados a él por un cable.



> Módulo prearmado de Bluetooth de la firma Kcwirefree. Las conexiones en todo su perímetro se utilizan para controlar y alimentar el módulo.



> **Mediante dos cables de alimentación y dos de datos, podemos conectar el módulo a un microcontrolador de 3 V.**

En una Palm con Bluetooth es posible usar el programa TriConnect para ver un proyecto.

antecedido por el prefijo AT.
La conexión a los módulos es de tipo serie. Para el caso particular del módulo Bluetooth, utilizaremos una conexión serie a **115.200 baudios, 8N1**. El módulo viene configurado por defecto a ese valor, pero permite alcanzar una velocidad de control de **3 Mbps**. Podemos utilizar solo tres cables (TX, RX y masa) o cinco, para efectuar un control de flujo por medio de las señales **CTS** y **RTS**.
Un dato importante es que, además de la conversión de las señales **RS-232** de la computadora, es necesario adecuar el nivel de los valores lógicos a **3,3 V.**
Es posible utilizar el programa **Docklight** para acceder al módulo, o gestionarlo directamente desde un microcontrolador. Pero como en este caso estamos estudiándolo, lo más conveniente es emplear la PC.

> CONECTÁNDONOS

Cualquier dispositivo Bluetooth que inicie un proceso de **discovery** reconocerá nuestro módulo y podrá conectarse a él. También, desde el módulo y mediante el comando **AT+ZV Discovery**, podemos obtener un listado de los dispositivos Bluetooth próximos. Tomamos la <dirección> del que nos interesa y, enviando **AT+ZV SPPConnect** <dirección>, nos conectamos a él.

No bien alimentamos el módulo, este responderá algo similar a lo siguiente:

```
AT-ZV -CommandMode-
AT-ZV BDAddress 00043e3a3111
```

En la hoja de datos del fabricante, se encuentran todos los comandos disponibles y su explicación en detalle. Incluso, se provee de un software especial para ensayar el módulo, sin necesidad de escribir todos los comandos. Una última consideración para tener en cuenta es el modo de operación del módulo. Al momento de alimentarlo, este se encuentra en el modo **COMANDO**, quiere decir que podemos configurarlo e interactuar con todos los comandos disponibles. Luego de realizar una conexión con otro terminal, el módulo pasa al modo **BYPASS**, siendo este transpa-rente para los datos emitidos y recibidos. Para regresar al anterior, debemos enviarle una secuencia denominada "de escape", que consiste en los caracteres: **^#^$^***. La secuencia que fuerza al modo BYPASS –estando en COMANDO– es: **AT + ZV Bypass.** ■

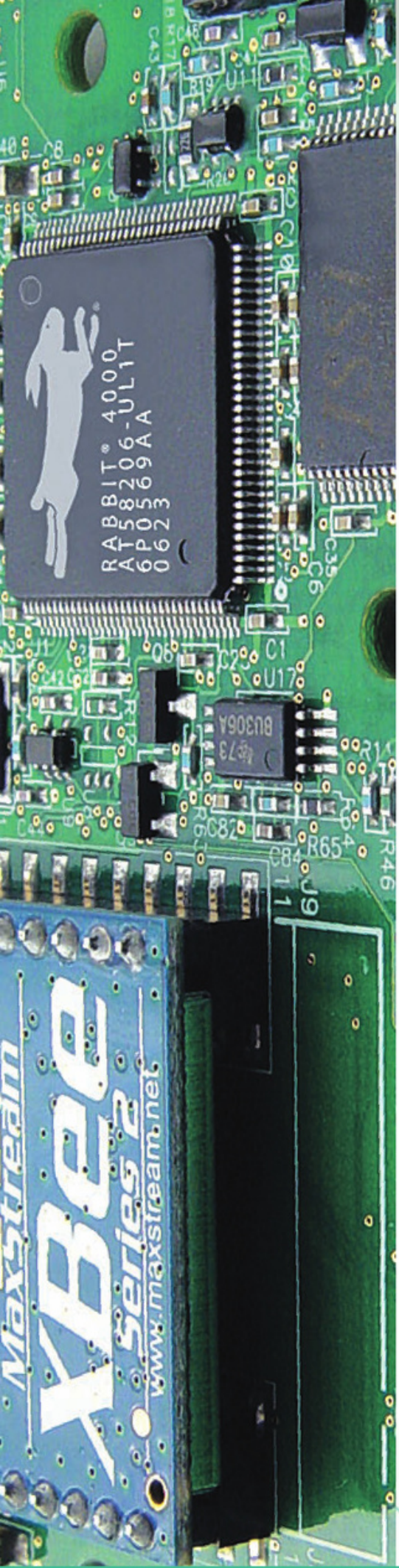
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Mencione algunas alternativas de conectividad.
2. ¿Qué es Zigbee?
3. ¿Existen alternativas para ZigBee?
4. Enumere las características del ruteo en ZigBee.
5. ¿Qué es el módulo X.Bee 802.15.4?
6. Caracterice al control por RF.
7. ¿Qué es el control por infrarrojos?
8. ¿Para qué sirve el código RC5?
9. Mencione las ventajas de la especificación Bluetooth.

EJERCICIOS PRÁCTICOS

1. Instale la aplicación X-CTU en la PC.
2. Conecte el módulo XBee a la PC y configúrelo.
3. Arme una red de sensores ZigBee.
4. Utilice las ventajas de la aplicación TriConnect.



CAPÍTULO 7

MÓDULOS RABBIT

Módulos Rabbit
Dynamic C
Mundo Rabbit
Actividades

07

MÓDULOS RABBIT



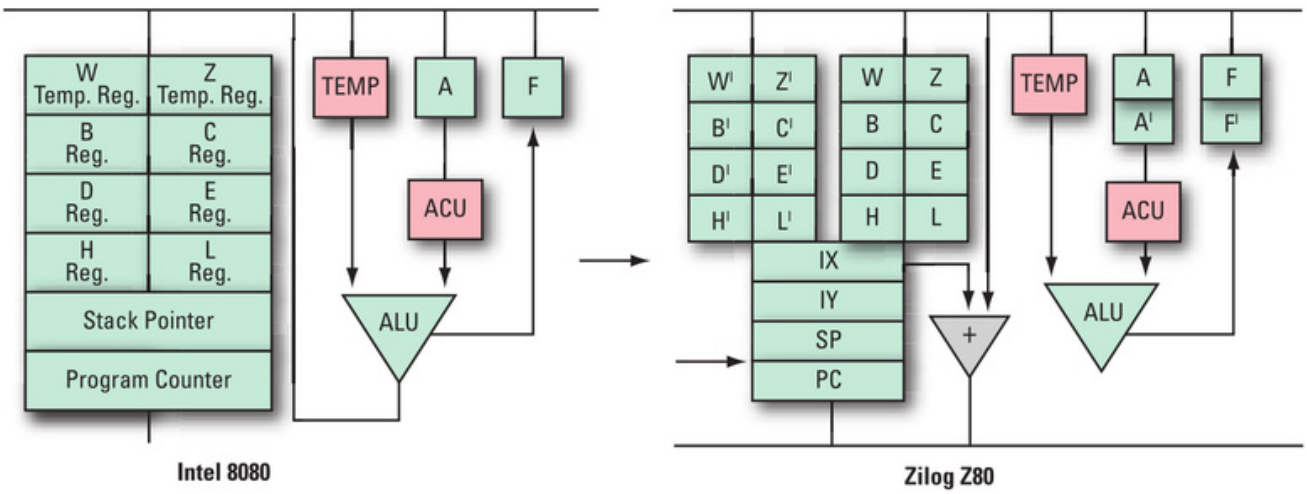
Un módulo Rabbit es una placa de circuito impreso con conectores que le permiten vincularse al resto del circuito que controla. Cumple la función de un microcontrolador, pero algunos modelos agregan, además, ciertas características adicionales, como zócalo para tarjetas **mini-SD**, conector **RJ-45** para Ethernet e interfaz de antena para **Wi-Fi**.

La arquitectura

Como ya conocemos algunas características fundamentales de los micros y, también, pudimos aprender que la familia **Rabbit** se basa en el **Z80**, en cierto modo, sucesor del famoso **8080**. La arquitectura empleada es **Von Neumann**, que, si bien tiene todas las desventajas que ya hemos analizado, también presenta varios puntos a favor:

- Permite mayor sencillez al requerir solo un bus de datos y direcciones; por consiguiente, un área de memoria.
- No requiere que los programas en lenguajes de alto nivel, como C, deban preocuparse de a qué área apuntan los punteros.

En particular, nos ocuparemos del **Rabbit 5000** que, si bien es mucho más complejo internamente que el **2000** –el primero, nacido en ese año–, resulta más simple de manejar, sin necesidad de conocer todos sus vericuetos. Este core es capaz de operar a **100 MHz**, empleando desde **dos** ciclos para resolver la operación más simple (**50 MIPS** pico), con un consumo de **0,57 mA/MHz.** El modelo de CPU es **CISC** (*Complex Instruction-Set Computer,* o computador con set de instrucciones complejas), donde se busca generar código más compacto proveyendo instrucciones que resuelven la mayoría de los problemas. Sus **más de 400** instrucciones permiten resolver desde sumas entre registros, hasta movimiento de bloques de datos,



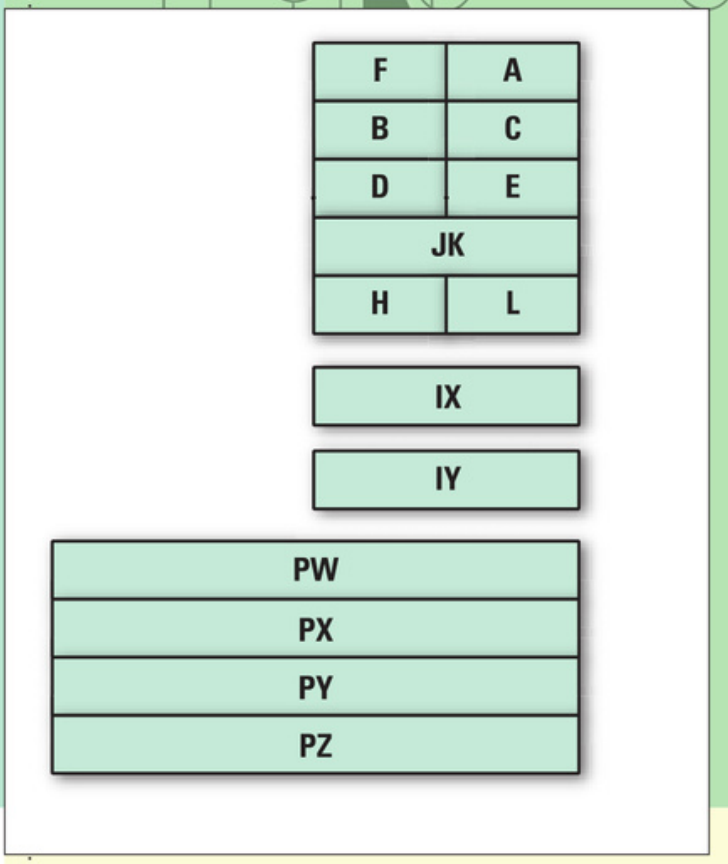
W y Z son registros temporarios inaccesibles para el programador F (Flags) es el registro de Estado.

> **Figura 1. La evolución de la estructura interna del 8080 al Z80, en el que se basa el Rabbit 2000.**

incluyendo, por supuesto, multiplicaciones (en 32 bits).

El micro por dentro

Internamente, el micro tiene un **bus de 16 bits** e instrucciones de 8 y 16 bits, con algunas de 32 bits. En su set de registros (**Figura 2**) vemos el acumulador **A**, de **8 bits**, y varios registros de 8 bits que pueden agruparse de a pares y utilizarse en **16 bits**. Por ejemplo, el par **HL**, que es la concatenación de los registros **H** y **L**, funciona como un acumulador de 16 bits. A su vez, el par **JK** puede agruparse con **HL**, y el **BC**, con **DE**, formando **JKHL** y **BCDE**, dos registros de 32 bits que permiten algunas operaciones lógicas y aritméticas. También disponemos de un **set alternativo**, como en el **Z80**, pero en este caso, dichos registros pueden emplearse más libremente, casi duplicando la cantidad de registros.



> **Figura 2. Aquí podemos ver el mapa de registros de la CPU Rabbit 5000, donde apreciamos registros desde 8 hasta 32 bits.**

Puertos de entrada/salida

Al igual que el **8080** y los **80x86** (que, en cierto modo, también derivan del 8080), Rabbit posee un área especial para **I/O**. Estos puertos no comparten el mapa de memoria; por este motivo, tal como en las PCs, se requiere de una instrucción especial en **Assembler** y una función especial en C para escribir o leer un puerto de I/O. El **Rabbit 5000** incorpora seis puertos de I/O: **A**, **B**, **C**, **D**, **E** y **H**. Los pines se comparten con los demás periféricos. El puerto **H** normalmente se dedica a proveer los 8 bits altos para acceder a la memoria en 16 bits.

Un módulo Rabbit es una microcomputadora en un módulo con pines de interfaz.

La arquitectura Rabbit diferencia, además, entre **espacio de I/O interno y externo**. Los periféricos del micro se hallan en el espacio interno, mientras que en el externo es posible conectar cualquier periférico y direccionarlo en el **mapa de I/O externo**. Algunos de los puertos de I/O permiten configurar sus pines como decodificadores de espacio de direcciones y seleccionar periféricos externos, eliminando la decodificación externa (glue-logic).



> PRIORIDAD DE EJECUCIÓN

La CPU puede operar en cuatro niveles distintos de prioridad. Las interrupciones se configuran para un nivel de operación. En todo momento, solo son aceptadas aquellas cuyo nivel de prioridad supera al de ejecución en dicho instante. Si la CPU está operando a nivel cero, las acepta todas; si está a nivel tres, no acepta ninguna.

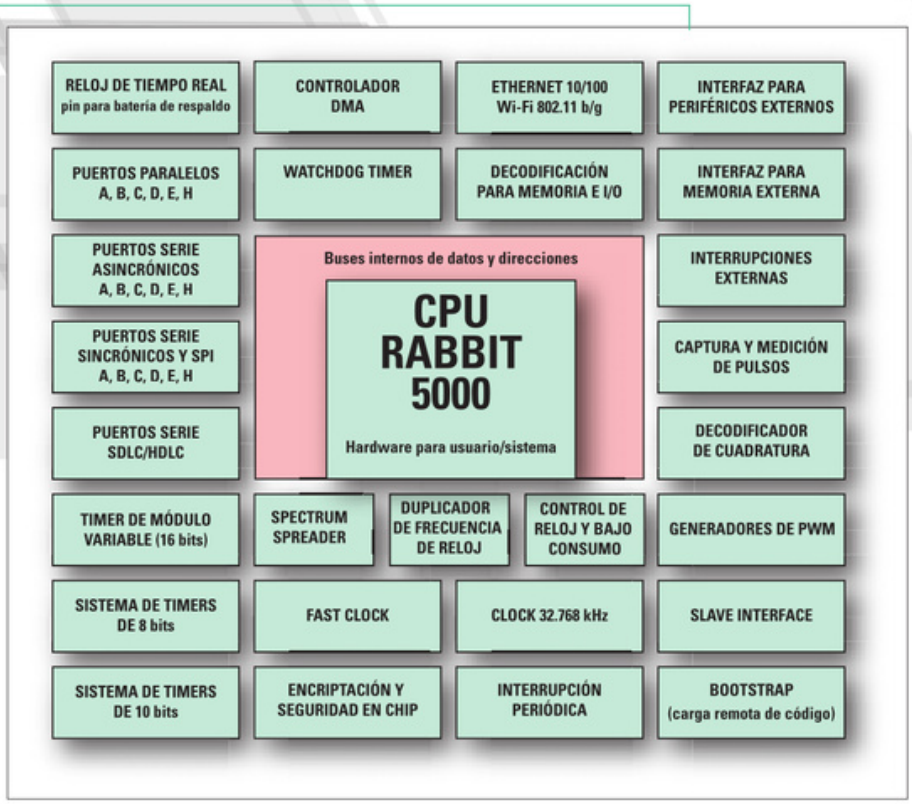


Figura 3. Diagrama en bloques del Rabbit 5000, junto con sus periféricos y bloques constitutivos.

> UNIDAD DE MANEJO DE MEMORIA

Este concepto de manejo de memoria nace de la necesidad de operar sobre una gran cantidad de memoria, sin perder la simplicidad y eficiencia de un direccionamiento de 16 bits en un micro de 8 bits, y sin emplear bancos. La MMU (unidad de manejo de memoria) controla la ventana en el espacio de 64 KB que nos permite ver una porción del espacio total de 16 MB.

Manejo de memoria

El Rabbit 5000 puede manejar hasta **16 MB** de memoria, mediante sus cuatro registros de 32 bits (**PW**, **PX**, **PY**, **PZ**) e instrucciones de acceso a memoria en **24 bits**. Para mayor economía, también incluye una unidad de manejo de memoria (**MMU**, *Memory Management Unit*) que permite emplear punteros y direccionamiento de **16 bits**.

Periféricos

Dentro del chip **Rabbit 5000** encontramos los siguientes periféricos:

- Seis puertos serie (**A**, **B**, **C**, **D**, **E** y **F**) asincrónicos con capacidad IrDA (comunicación por infrarrojo).
- Un conjunto de timers de **8 bits (A)** para relojes de los demás periféricos.
- Un timer de **10 bits (B)** con dos módulos de comparación.

La unidad de captura de eventos permite medir pulsos del orden de los nanosegundos.

- Un contador de módulo variable hasta **16 bits (C)** con cuatro juegos de registros de comparación set/reset. Este esquema es muy útil para controlar motores paso a paso y generar señales **PWM** (modulación de ancho de pulso) multifase, para regular la velocidad de motores trifásicos.
- Un contador asincrónico (*ripple counter*) de **48 bits** con alimentación independiente para operar como **RTC** (*Real Time Clock*, o reloj de tiempo real).
- **IOSTROBEs** (chip selects para periféricos externos).
- Ocho canales de **DMA** (*Direct Memory Access*, o acceso directo a memoria).
- **Dos detectores de cuadratura** de 8 o 10 bits, que permiten leer encoders rotativos y determinar, por ejemplo, el sentido de giro de motores.
- Cuatro canales **PWM** con posibilidad de interrupción y supresión de pulsos, más DMA.
- **Dos unidades de captura y cuenta de eventos**, para medir pulsos de muy corta duración, del orden de los nanosegundos.
- **Ethernet 10/100** integrada y con DMA.
- **Wi-Fi 802.11b/g** integrada y con DMA.

En la página del fabricante, **www.rabbit.com**, podemos obtener el manual del usuario, donde consultar una descripción detallada del micro y todos sus periféricos. ■

DYNAMIC C

Es el entorno de desarrollo (IDE) y compilador C que el fabricante nos brinda para realizar nuestros proyectos. Incluye depuración en circuito y las herramientas necesarias.

La historia de Dynamic C es más larga que la de las **CPU Rabbit**. Comenzó como un compilador propietario, y fue lentamente transformándose en un entorno de desarrollo completo. En un principio, no era compatible con el estándar **ANSI** (*American National Standards Institute*), pero poco a poco fue incorporando las fortalezas de este, sin perder las propias. Podemos decir que Dynamic C es una implementación de C con funciones específicas para Rabbit.

Características

El compilador se aleja del estándar **ANSI**, que el fabricante considera preferible para los sistemas dedicados, por ejemplo, Dynamic C:

- No pone a cero las variables estáticas que no fueron inicializadas.
- No utiliza **include files** por defecto, sino que la directiva ***#use*** reemplaza a ***#include***.
- Compila directamente a la memoria del sistema objeto.
- Incluye soporte para **multitarea cooperativo**.
- Agrega tipos para identificar variables compartidas por contextos diferentes o almacenados en memoria no volátil.

En Dynamic C, los enteros son de **16 bits**, y los enteros largos, de **32 bits**. Los números en coma flotante son solo de simple precisión (**32 bits**). Para indicar una función o un fragmento de código Assembler, solo tenemos que escribirlo entre las directivas **#asm y #endasm**.

> C Y VON NEUMANN

Muchos de los lenguajes de alto nivel contemporáneos, como C, fueron desarrollados sobre máquinas con arquitectura Von Neumann, y mantienen el concepto de "ver toda la memoria como una", sin distinguir espacios de instrucciones y de datos. En Dynamic C, como en el lenguaje C estándar, no se requieren directivas para identificar el área adonde apuntan los punteros.

Variables protegidas

Si nuestro sistema se resetea en medio de la actualización de una variable **multibyte**, como un **array**, este quedará dañado. Si declaramos la variable de tipo **protected**, el compilador generará código que creará una **copia de seguridad** de la variable **antes de modificarla**. Si el sistema llegara a resetearse al momento de modificar dicha variable, el valor original podría recuperarse al reiniciar el sistema. La declaración es, por ejemplo: **protected long enterolargo.**

```
main()
{
   WrPortI(PDDDR, &PDDDRShadow, PDDDRShadow | (1<<3));
   WrPortI(PDDR, &PDDRShadow, PDDRShadow & ~(1<<3));


   WrPortI(ICCSR,&ICCSRShadow,1<<2);   // reset módulo 1, no ints

   WrPortI(ICCR,&ICCRShadow,0);  // no ints

   WrPortI(ICT1R,&ICT1RShadow,'\B01010110'); /*

01 = el contador arranca al ocurrir start y se detiene al ocurrir stop
01 = el módulo registra la cuenta ante la condición de stop
01 = asigna flanco ascendente a condición de start
10 = asigna flanco descendente a condición de stop
*/

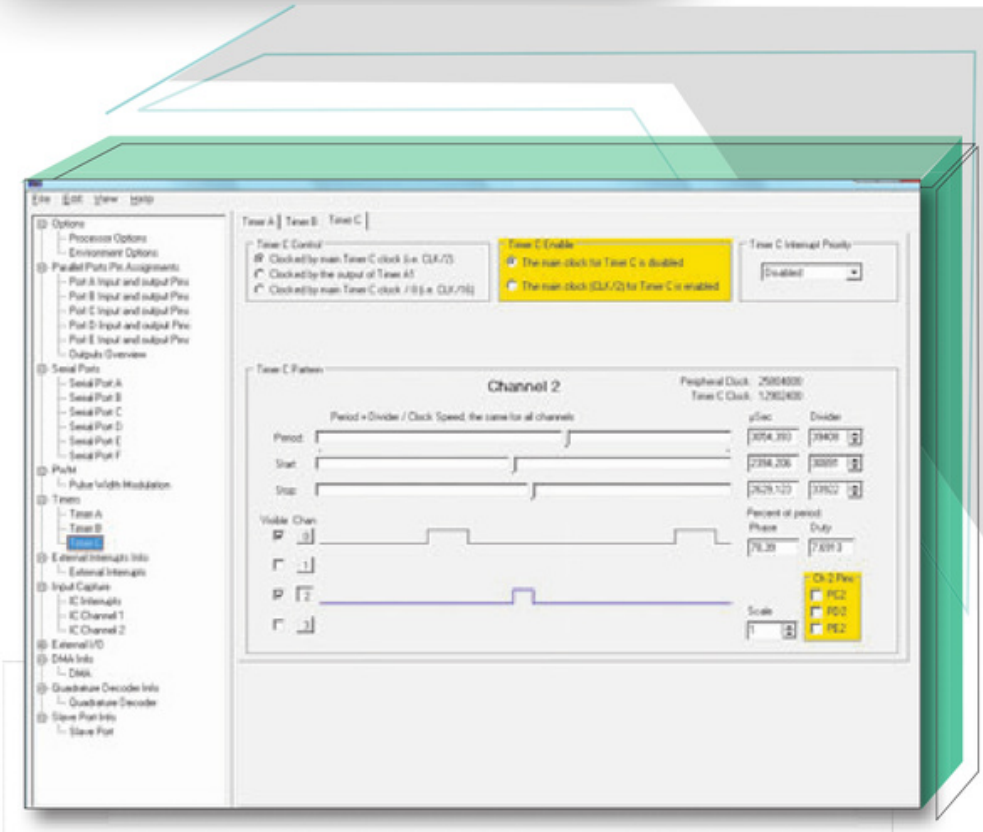
   WrPortI(ICS1R,&ICS1RShadow,'\B01010101'); // PD.3 start y stop

   WrPortI(TAT0R,&TAT0RShadow,0);            // max speed

   RdPortI(ICCSR);

#asm
```

> Aquí observamos el entorno de Dynamic C. El cursor verde indica la sentencia actual para depuración en circuito.



> **La utilidad de configuración operando sobre el timer C. El recuadro amarillo indica que prestemos atención.**

Dynamic C es una implementación de C con funciones específicas para Rabbit.

Variables compartidas

Si compartimos variables entre el programa principal y una rutina de interrupciones, una interrupción en medio de un acceso del programa principal a una de estas variables puede ser catastrófica. La rutina de interrupciones modifica el contenido de la variable al momento en que el programa principal había accedido a parte de ella. Entonces, la lectura que hará el programa principal será, cuando menos, incorrecta.
Si declaramos la variable de tipo **shared**, todos los **cambios** a ella son atómicos, es decir que se inhiben las interrupciones mientras se la modifica. Por ejemplo

```
shared long varcomp;
```

Algunas variables globales compartidas que incorpora el sistema son:

SEC_TIMER: contiene el tiempo del sistema en segundos.
MS_TIMER: lo contiene en milisegundos.

> ATOMICIDAD, CAMBIOS ATÓMICOS

Cuando determinada área se accede mediante contextos diferentes, es preciso que los cambios dentro de cada ente de dicha área se realicen de forma atómica. Es decir, toda la operación debe desarrollarse sin interrupciones, como una única acción indivisible. Algunos procesadores incorporan instrucciones para facilitar esta operación.

Utilidad de configuración de I/O

El Rabbit 5000 incluye varios niveles de asignación de periféricos a los pines del encapsulado.
Para facilitar la tarea al programador, existe un programa que permite realizar este proceso de forma gráfica, una tendencia actual en muchos fabricantes.

Rabbit BIOS

El BIOS (*Basic Input/Output System*, o sistema básico de entrada/salida) es, en este caso, **la interfaz entre el hardware y Dynamic C**.
Provee los servicios de bajo nivel y la inicialización del procesador, y es el encargado de cargar el código y la **depuración (in-circuit debugging)**.
Además, da la posibilidad de identificar el módulo mediante un área especial de la memoria flash, denominada **IDblock**, en la que se guardan estos datos.
También provee de soporte para que el programador guarde sus datos en un sector llamado **User Block**.

Bibliotecas de funciones

Junto con el compilador, el fabricante distribuye una serie de bibliotecas de funciones que resuelven tareas comunes. De este modo, el programador puede enfocarse en su aplicación. Excepto en algún caso en particular, como el de **Wi-Fi**, donde existe propiedad intelectual del fabricante, las bibliotecas de funciones se encuentran en código fuente (C y Assembler), que el usuario puede observar y modificar a gusto.
Para utilizar funciones de una **library**, debemos incluirla mediante la directiva **#use**. Esto es, en cierto modo, equivalente a **#include**, solo que **Dynamic C** maneja un sistema de ayuda incluido en las **libraries**. Las siguientes son las más comunes, excluyendo lo relativo a **networking**.

Puertos de I/O

Cuando el micro presenta un espacio de I/O independiente del de memoria, como en las PCs, necesitamos funciones para acceder a los puertos. Como Rabbit, además del espacio de I/O de un microprocesador, incorpora periféricos internos, tenemos funciones para ambos espacios, cuyos nombres terminan en las letras **E** (espacio **E**xterno) e **I** (espacio **I**nterno).

Grabar datos en flash

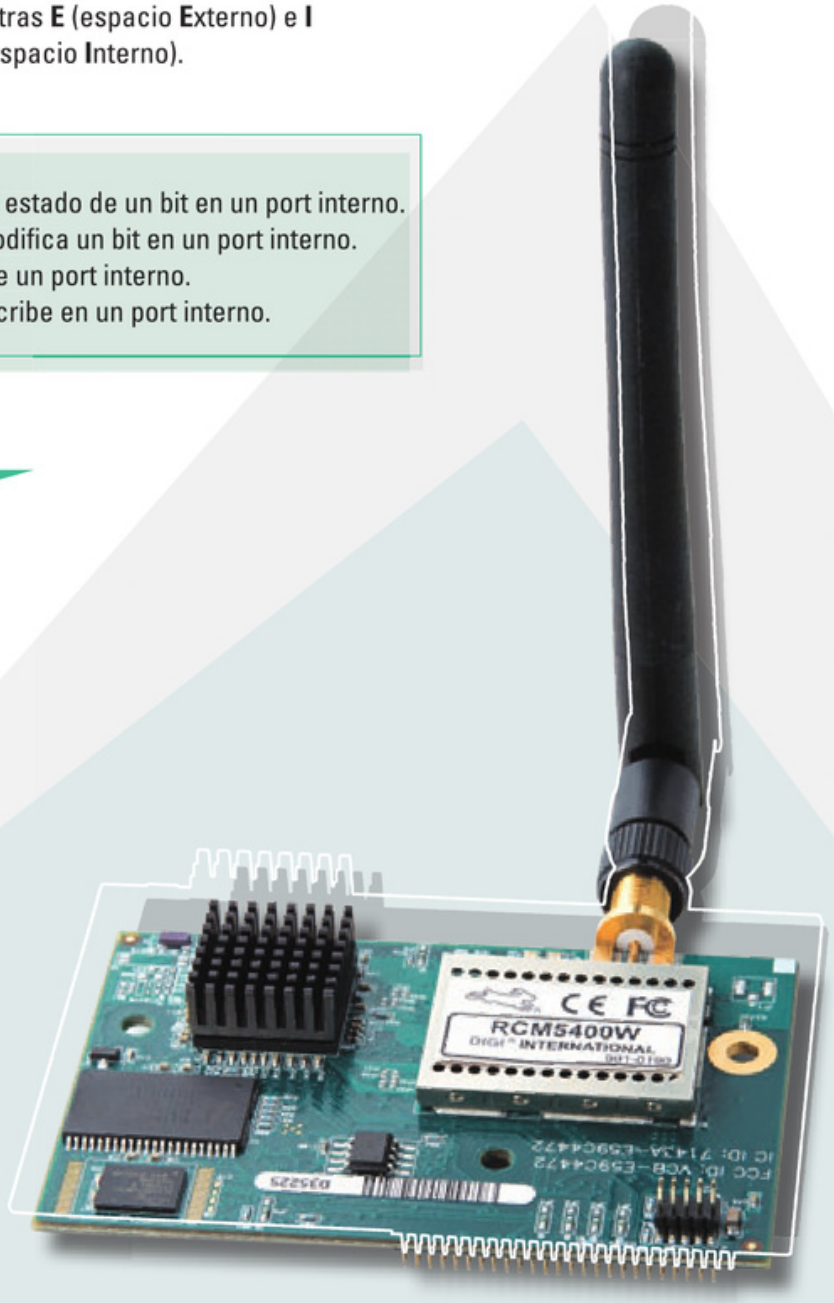
Esto se realiza, como dijimos, en el User Block.
Para leer y escribir en esta aplicación, disponemos de dos funciones:

- Una función lee **num** bytes del User Block, desde **offset** hacia **dest_addr**:
  readUserBlock(dest_addr,offset,num);
- Otra función graba **num** bytes en el User Block, desde **src_addr** hacia **offset:**
  writeUserBlock(offset,src_addr,num);

```
BitRdPortI(port,bit);          // Devuelve el estado de un bit en un port interno.
BitWrPortI(port,shadow,estado,bit);// Modifica un bit en un port interno.
RdPortI(port);                 // Lee un port interno.
WrPortI(port,shadow,data);     // Escribe en un port interno.
```

> INSTRUCCIONES DE I/O

Las CPUs como Rabbit y 80x86 poseen un espacio de I/O separado del mapa de memoria. La operación de I/O se realiza mediante funciones especiales, de modo que el compilador pueda insertar las instrucciones Assembler correspondientes para operar sobre este espacio.
La CPU Rabbit puede utilizar sobre el espacio de I/O los mismos modos de direccionamiento que sobre la memoria.



Librería RS-232

Para operar sobre los puertos serie, disponemos de la library **rs232.lib**, que provee de una interfaz (API) similar a la que encontramos en otros sistemas (**código 01**). Para operaciones complejas existe **packet.lib**, que permite operar sobre mensajes delimitados por un carácter especial, un tiempo de silencio o una ruptura de formato.

El GPS

La mayoría de los módulos **GPS** (*Global Positioning System*, o sistema de posicionamiento global) entrega la información de posición y fecha/hora empleando un protocolo llamado **NMEA-0183**. La información es texto, y el formato es simple y relativamente fácil de procesar, más aún si llamamos a esta función de **gps.lib**:

```
gps_get_position(newpos, sentence);
```

El parámetro **newpos** es un puntero a una estructura **GPSPosition**, con latitud y longitud separados en los elementos de esta; y **sentence** es un puntero a un array donde está la sentencia NMEA que obtuvimos del GPS por el puerto serie.

La librería PWM

Esta library provee de la interfaz con los generadores de PWM. (**código 02**). Recordemos que el **valor medio** de una señal PWM es el valor de tensión del pulso por el **ciclo de trabajo**. En este caso: **V = Vpulso x pw / 1024**.

> POSICIONAMIENTO GLOBAL

La constelación de satélites mantiene una base de tiempo común y coordinada. El GPS recibe la posición y el tiempo de cada satélite, y puede estimar la distancia de cada uno y determinar su posición en el espacio (x, y, z) y tiempo local (t). Esto, luego, se traduce a coordenadas de latitud y longitud mediante un sistema geodésico (WGS84), que aplica un modelo matemático de la Tierra.

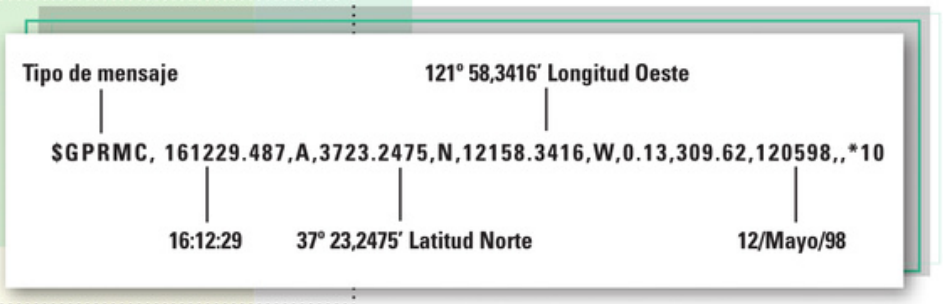
La library rs232.lib permite operar sobre un puerto serie de forma similar a como se hace en una PC.

> Código 01

```
serBopen(9600);                    // elige 9600 bps
serBdatabits(PARAM_8BIT);          // elige 8 bits
serBparity(PARAM_NOPARITY);        // elige sin paridad
serBputs(string);          // envía el texto apuntado por string
serBread(data, num, tmout);        // recibe hasta num bytes, los coloca
                                   // en el array data, mientras no
                           // transcurra un tiempo tmout sin
                           // que haya datos
      serBwrite(data,num);         // escribe num bytes del array data
```

> Código 02

```
pwm_init(frec);            // configura la frecuencia de operación
pwm_set(ch, pw, NULL);     // setea el canal ch con un ciclo de
                           // trabajo pw de 0 a 1024
```



> Figura 4. Detalles de un mensaje de un módulo GPS en protocolo NMEA-0183. Para extraer la información, solo debemos seguir las comas.

Reloj de tiempo real

Operamos sobre el **RTC** mediante funciones de **rtclock.lib**. Por ejemplo, en el siguiente fragmento:

```
unsigned long secs ;
struct tm thetm;

      secs = read_rtc();
      mktm(&thetm, SEC_TIMER);
```

Así obtuvimos en la estructura **thetm** la información de fecha y hora con cada dato en un elemento de ella.

Detectores de cuadratura

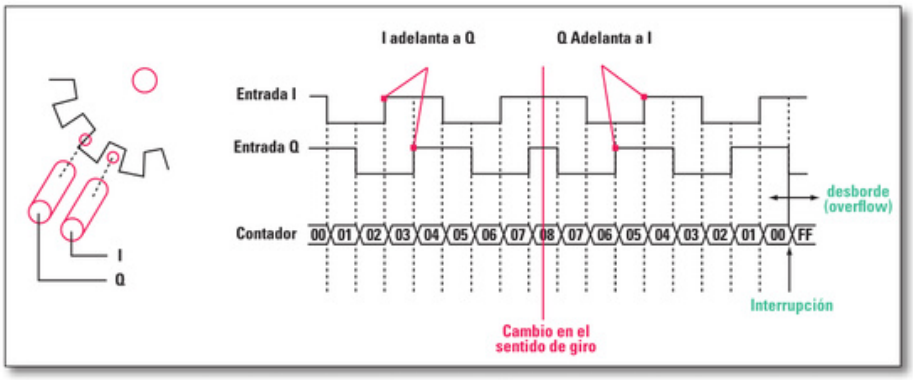
La library **qd.lib** permite trabajar con los detectores de cuadratura. Leyendo el valor actual y comparando con el anterior, podemos detectar la velocidad y el sentido de giro. Esto permite leer, por ejemplo, un encoder rotativo para una interfaz de usuario (**código 03**).

Displays con pantalla sensible

Existen dos libraries con funciones de soporte para displays monocromáticos de 320 x 240 pixeles y touchscreens. Ambas manejan todo lo relativo a posicionamiento de gráficos, texto, dibujo, lectura de botones con texto y gráficos, para realizar interfaces de control al tacto.

I²C y SPI

Las libraries **i2c.lib** y **spi.lib** ofrecen funciones para controlar dispositivos mediante estos protocolos. Por ejemplo, las funciones **i2c_init**() y **SPIinit**() inicializan el hardware (**código 04**).
Aquí, **slave** es la dirección del dispositivo (I2C), e **index**, la dirección en memoria de este; **num** es la cantidad de bytes (I2C) o bits (SPI), **src** es la dirección donde están los datos en memoria, y **dest**, donde se guardan. ■



> Figura 5. Un codificador en cuadratura y la operación del detector de Rabbit. El principio se basa en detectar los flancos y recordar el estado anterior.

> CODIFICADOR EN CUADRATURA

Permite codificar la velocidad y el sentido de giro mediante dos señales desfasadas 90º. Por ejemplo, una rueda dentada en la que un transmisor y dos receptores infrarrojos están mecánicamente separados la mitad del ángulo ocupado por los dientes (Figura 5). Estos dispositivos son muy utilizados en robótica.

> Código 03

```
qd_init(int);       // inicializa y configura prioridad de interrupciones
qd_zero(mod);       // resetea el contador del módulo mod
qd_read(mod);       // devuelve el valor del contador del módulo mod
```

> Código 04

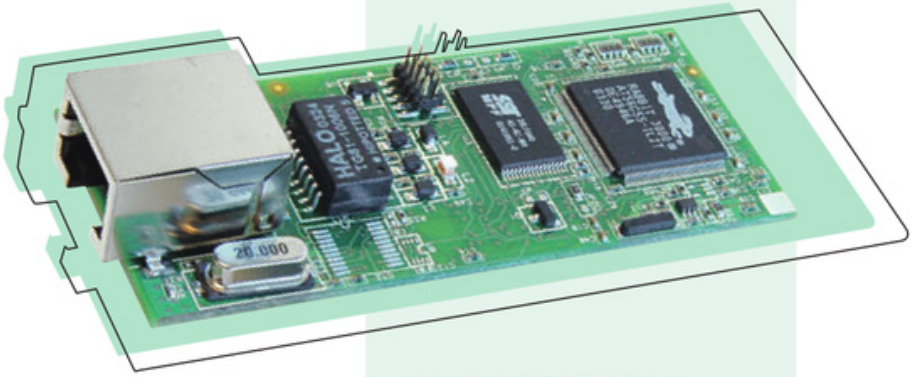
```
I2CWrite(slave, index, src, num);
I2CRead(slave, index, dest, num);

SPIRead(dest, num);
SPIWrite(src, num);
SPIWrRd(src, dest, num);   // lee y escribe a la vez
```



MUNDO RABBIT

¿Cómo está conformado el mundo Rabbit? En estas páginas nos interiorizaremos acerca de las posibilidades que nos brinda esta familia de módulos.



> El módulo RCM3720 tiene pines de paso .1" y posee conexión Ethernet 10BaseT mediante un chip externo al micro.

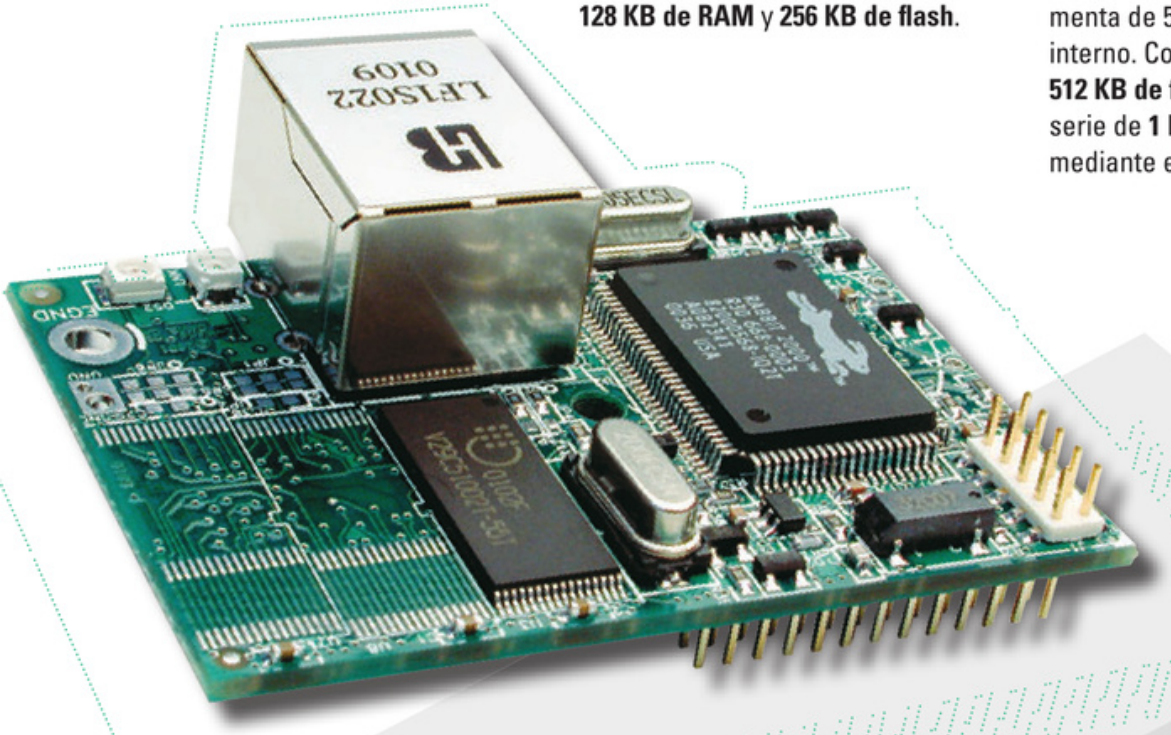
Existen al momento cuatro CPUs Rabbit: 2000, 3000, 4000 y 5000. La última generación, el Rabbit 6000, se espera dentro del año 2010. Las dos primeras CPUs contienen una unidad de manejo de memoria que les permite direccionar hasta **1 MB** de código o datos (recordemos que utiliza la arquitectura Von Neumann). Esa memoria es vista a través de un espacio lógico de **64 KB**. Para cantidades de datos que no quepan en ese espacio, es necesario utilizar técnicas de programación no muy simples para el principiante. A partir del Rabbit 4000, los **punteros de 32 bits** y las nuevas instrucciones permiten que el compilador utilice el concepto de **far pointers** (punteros lejanos) y direccione toda la memoria (16 MB) de manera más eficiente, sin que el programador deba ocuparse de hacerlo.

Módulos de 5 Volts

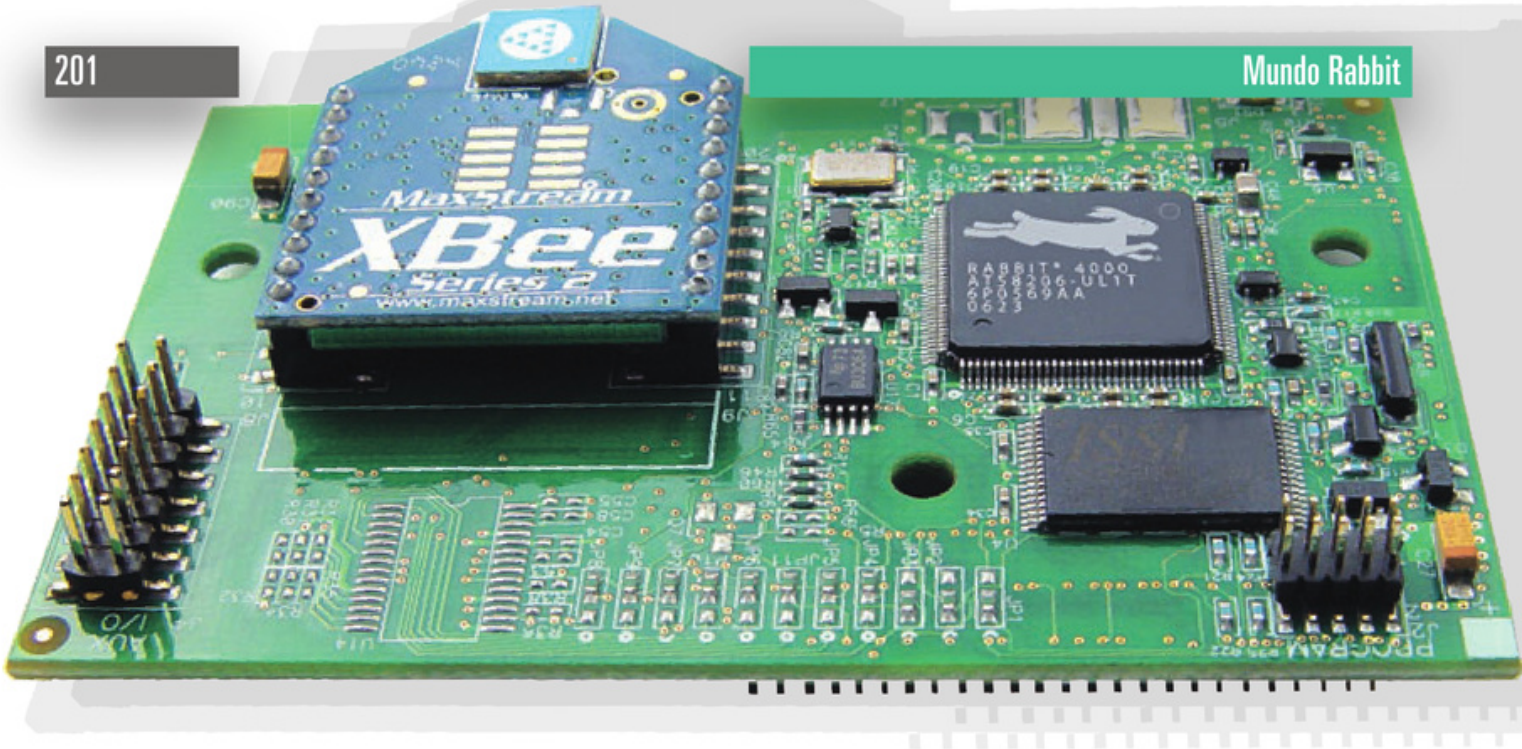
La familia de módulos **RCM2xxx** está basada en la CPU Rabbit 2000, la primera de todas, bastante menos poderosa. Son módulos de **5 V**. El más versátil y utilizado es el **RCM2200**, que incorpora **128 KB de RAM** y **256 KB de flash**.

Módulos de 3,3 Volts

La CPU **Rabbit 3000** funciona a **3,3 V.** La familia de módulos **RCM3xxx** está basada en esta **CPU**, y son módulos con puertos de **I/O** de **3,3 V**, pero tolerantes a **5 V**. Esto significa que, si bien el micro entrega tensiones de 3,3 V, permite recibir datos de un periférico de 5 V. Si los niveles lógicos del periférico pueden reconocer el valor de tensión entregado por el Rabbit 3000 como un **1 lógico**, la conexión es posible. El más simple y utilizado de estos módulos es el **RCM3720**, que se alimenta de 5 V e incluye un regulador interno. Contiene **256 KB de RAM**, **512 KB de flash**, y una memoria flash serie de **1 MB** que puede utilizarse mediante el sistema de archivos **FAT**.



> La imagen corresponde a un módulo RCM2200, con Ethernet 10baseT incluido, provisto por un chip externo al microprocesador.



El módulo RCM4510W ofrece 512 KB de RAM, 512 KB de flash y conectividad ZigBee mediante un módulo XBee ZB.

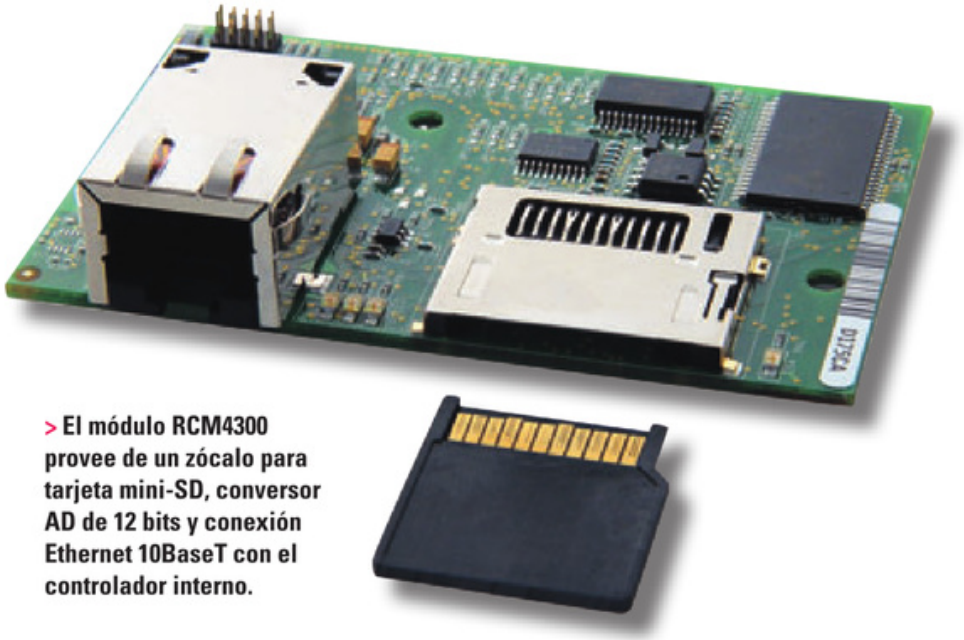
Módulos de 3,3 Volts

El chip anterior al **Rabbit 5000** es el Rabbit 4000, que tiene un controlador **Ethernet** en el chip, no posee controlador Wi-Fi y su bus interno de datos es de 8 bits, como los anteriores. Es el primero en incorporar **punteros de 32 bits** para facilitar el manejo de memoria cuando la cantidad de datos no cabe en el espacio de **64 KB** que sus antecesores manejan directamente. Se trata de módulos de **3,3 V** que no toleran **5 V** en sus entradas. En este caso, ya disponemos de más opciones, como tarjetas **mini-SD** y **módulos ZigBee**. Las tarjetas pueden ser leídas en cualquier otro sistema, pues emplean archivos FAT (*File Allocation Table*, o tabla de asignación de archivos). También existen módulos con **Wi-Fi 802.11 b/g** basados en el Rabbit 5000, como el RCM5400W, que incorpora **512 KB de RAM** y **512 KB de flash**. Dada la velocidad de la CPU, este micro incluye otros **512 KB RAM** para copiar el programa de la memoria flash y **ejecutarlo en RAM**, y una **flash serie de 1 MB**.

Minicores (3,3 V)

Basados en el Rabbit 5000, los minicores son la opción más económica del fabricante y son los que empleamos. El **RCM5600W** no tiene memoria flash paralelo, sino que emplea una **flash serie de 1 MB**, cuyo contenido es copiado por el micro a la **RAM de 1 MB** al inicializar, y se ejecuta desde allí. La CPU opera a **74 MHz**, y el tiempo de acceso requerido (2 ciclos) es de aproximadamente 25 nanosegundos (**~25 ns**), imposible con una memoria flash en la actualidad. El **RCM5700** incorpora una **flash paralela de 1 MB**, pero su CPU opera a menor frecuencia (**50 MHz**) y se configura para insertar tiempos de espera (wait-states) al acceder a la flash. Como la RAM, utiliza los **128 KB internos** del Rabbit 5000. ■

Los minicores son módulos con prestaciones orientadas a los aficionados.



> El módulo RCM4300 provee de un zócalo para tarjeta mini-SD, conversor AD de 12 bits y conexión Ethernet 10BaseT con el controlador interno.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué es un módulo Rabbit?

2. Mencione algunas de las características de la arquitectura de un módulo Rabbit.

3. Enumere algunos de los periféricos que encontramos en el módulo Rabbit.

4. ¿Cuáles son las principales características de Dynamic C?

5. ¿Qué son las variables protegidas?

6. Caracterice las funciones de algunas librerías de Dynamic C.

7. Mencione los procesadores Rabbit existentes.

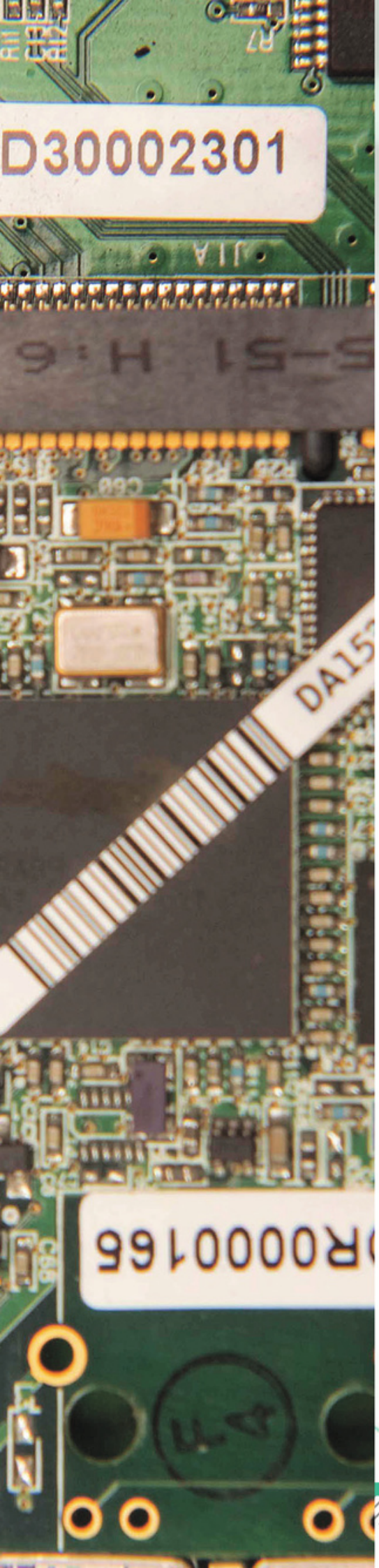
8. ¿Qué son los minicores?

EJERCICIOS PRÁCTICOS

1. Realice un diagrama que contenga los periféricos y bloques constitutivos del módulo Rabbit.

2. Instale Dynamic C en la computadora.

3. Conozca la interfaz de Dynamic C.



CAPÍTULO 8

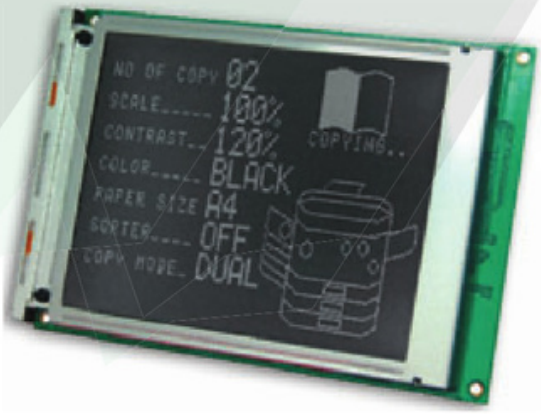
DISPLAYS GRÁFICOS LCD

Displays gráficos LCD
Arquitectura de un display digital
Cómo dibujar letras
Cómo dibujar gráficos y textos
Actividades

08

DISPLAYS GRÁFICOS LCD

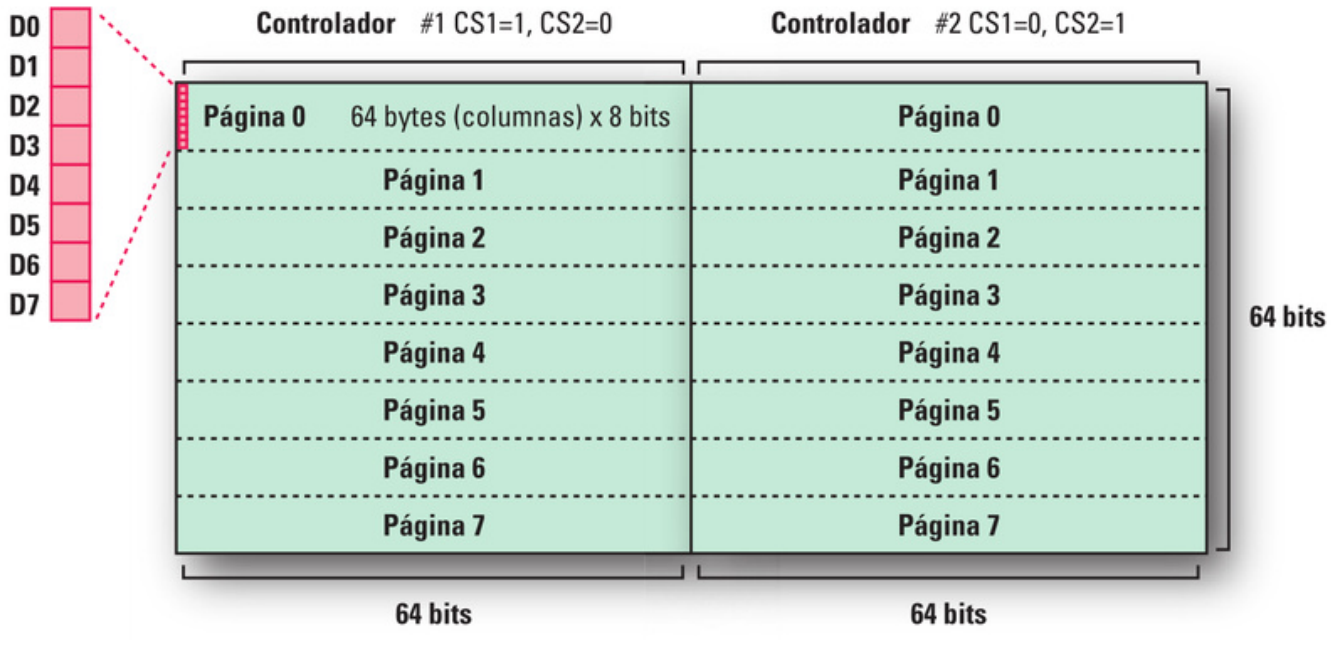
Veamos los detalles y la forma de controlar displays gráficos para agregar mayor profesionalismo y atractivo visual a nuestros proyectos.



Existen varias clases de displays gráficos LCD. Esto se debe a que distintos tamaños de display llevan a diferentes resoluciones (puntos) y, a su vez, existen distintos tipos de controladores, orientados más a uno o a otro tipo de aplicación. Nos concentraremos aquí en un tipo de displays cuyo controlador es similar al utilizado en los alfanuméricos; de hecho, la interfaz eléctrica es prácticamente la misma. El controlador original es el **HD61202**, desarrollado por Hitachi; en la actualidad muchos displays emplean el **KS0108** de Samsung, que es virtualmente idéntico en operación, aunque presenta algunas diferencias en los niveles lógicos. El **KS0108** acepta señales de **3,3 V**, mientras que el HD61202 no lo hace. Estos displays suelen presentarse en dos formatos: **64 x 64 pixeles** y **128 x 64 pixeles**. En este último caso, el display dispone de dos controladores, uno para el lado izquierdo y el otro para el lado derecho, pues el chip no puede controlar más de **64 columnas y 8 filas**.

El display

La **Figura 1** muestra la estructura de memoria de estos módulos gráficos. El primer punto de pantalla –arriba a la izquierda– corresponde al bit menos significativo del primer byte de memoria, del primer controlador. El punto en pantalla –abajo a la derecha– corresponde al bit más significativo del último byte de memoria, del segundo controlador.



> **Figura 1. Estructura de la memoria (mapa de la imagen) en el display utilizado. Cada punto encendido es un bit en una posición de memoria.**

El control de contraste requiere una tensión negativa que los displays generan en el módulo.

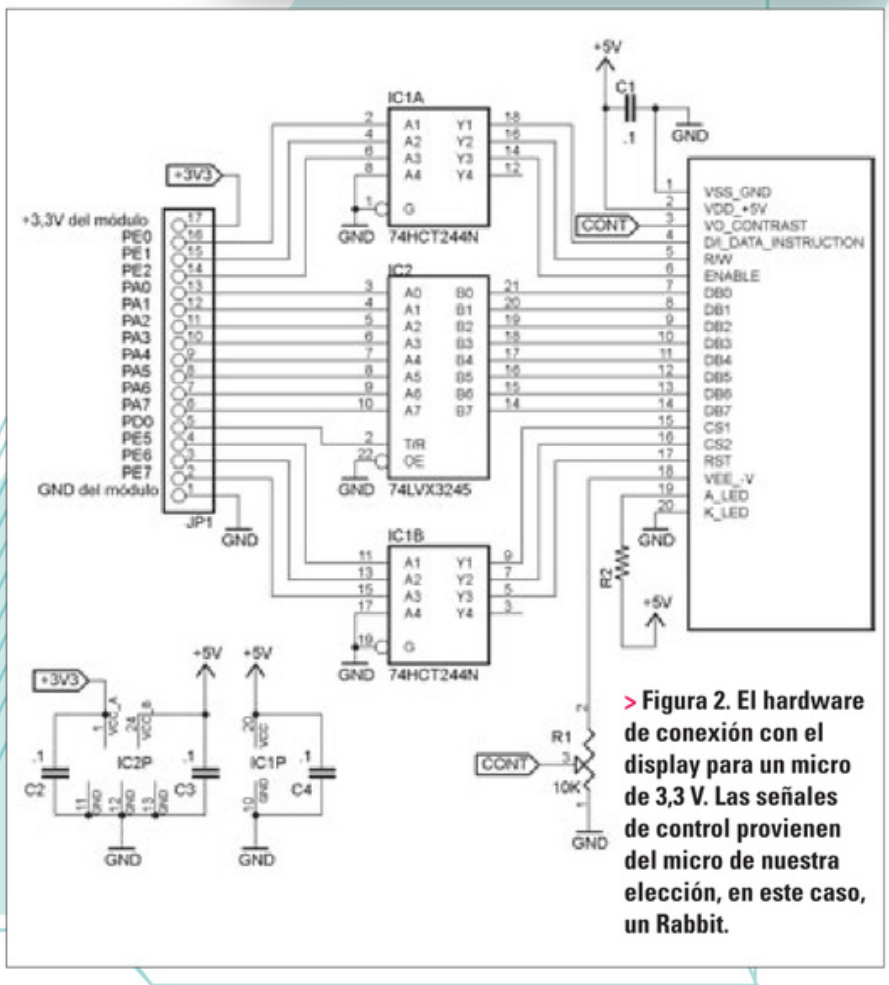
Seleccionamos el byte por leer o escribir en memoria a través de comandos, como en los displays alfanuméricos, pero ahora, tenemos dos direcciones:

- horizontal (número de byte, de 0 a 63)
- vertical (número de página, de 0 a 7)

Un contador autoincrementado, que apunta a la dirección siguiente, luego de una lectura o escritura, controla la dirección horizontal.

Control de contraste

Estos displays requieren una tensión negativa para el control de contraste. Afortunadamente, gran cantidad de módulos la generan de manera interna y la proveen en un pin.
La interfaz es la ya familiar originada en el bus del **procesador 6800**. Para controlar el display con un micro-controlador, imitamos la operación de ese bus, asignando las señales de control a puertos de I/O del micro; esto se conoce como **bit-banging**.
En el caso de displays de **128 x 64 pixeles**, tenemos dos pines de control adicionales, cada uno de los cuales habilita a un controlador.
Para un micro de **5 V**, conectamos directamente a los puertos de **I/O**, en tanto que para micros de **3,3 V**, debemos agregar un **conversor de nivel**. La opción genérica es emplear un **bus transceiver**, como el **74LVX3245**, que se controla desde el lado del micro a 3,3 V, y admite del otro lado un periférico a 5 V, convirtiendo niveles en uno u otro sentido.



> Figura 2. El hardware de conexión con el display para un micro de 3,3 V. Las señales de control provienen del micro de nuestra elección, en este caso, un Rabbit.

> DISPLAYS Y CONTROLADORES

Hay displays gráficos que incluyen al controlador, y otros que no lo hacen. Existen varios tipos de controladores, cada uno con interfaz y comandos diferentes. El HD61202 y su clon, el KS0108, son solo una alternativa, pero resultan la más simple y económica. Su similitud con los controladores de displays alfanuméricos reduce la cantidad de material nuevo que se debe aprender.

Para las señales de control, utilizamos un **buffer** de tecnología **HCT** (*High-speed CMOS with TTL levels*). Sus niveles lógicos de entrada le permiten reconocer 3,3 V como un 1 lógico, alimentado a 5 V. En el circuito de la **Figura 2**, tomamos los 3,3 V del módulo Rabbit y los 5 V de la fuente.

Control del display

El manejo de los pines de control del display requiere tener en cuenta el control del **74LVX3245** y el orden de activación de las salidas (**Figura 3**).
El pin **I/D** selecciona el envío de comandos (**I**) o el operar sobre la memoria (**D**). La **Figura 4** muestra la operatoria de manejo de control en bajo nivel del display, cuyas actividades usuales son: leer un dato (la memoria) del display, escribir un dato en el display y enviar un comando.
Si el micro es muy rápido, deberemos insertar demoras para cumplir con los **tiempos de acceso** del display, especificados en su hoja de datos.

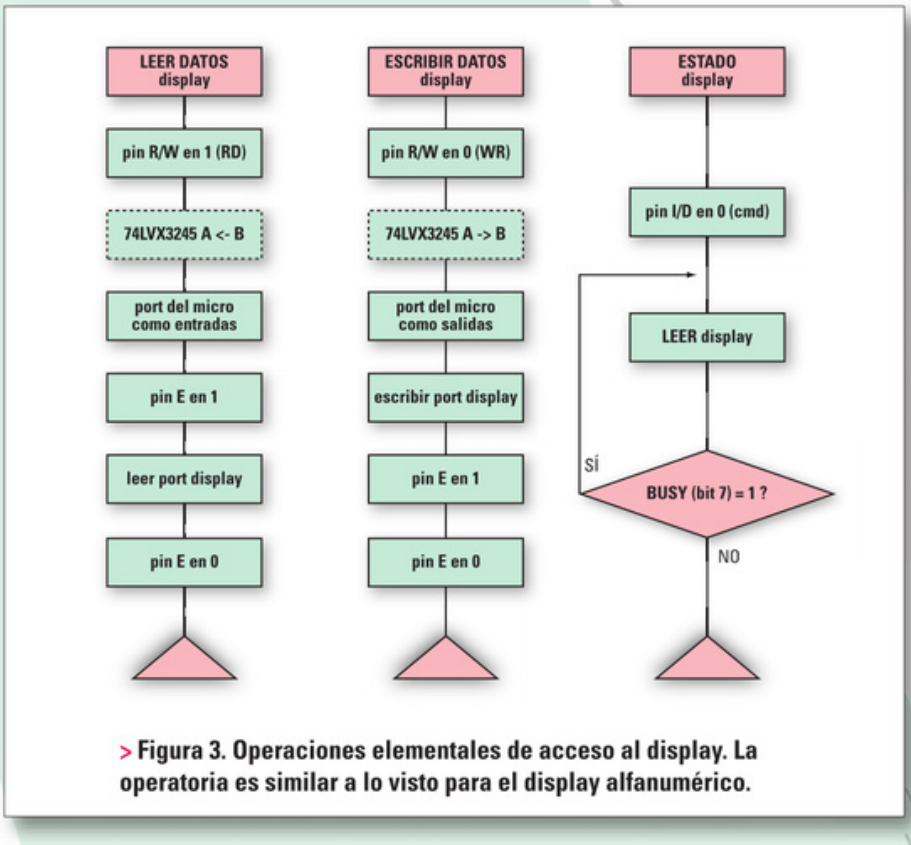
Algoritmos

Ya estamos en condiciones de desarrollar los algoritmos que nos permitirán mostrar textos, gráficos (iconos) y funciones en el display. Comenzamos definiendo que el punto con coordenadas (0;0) es el del extremo superior izquierdo de la pantalla. Hecho esto, podemos enunciar algunos algoritmos:

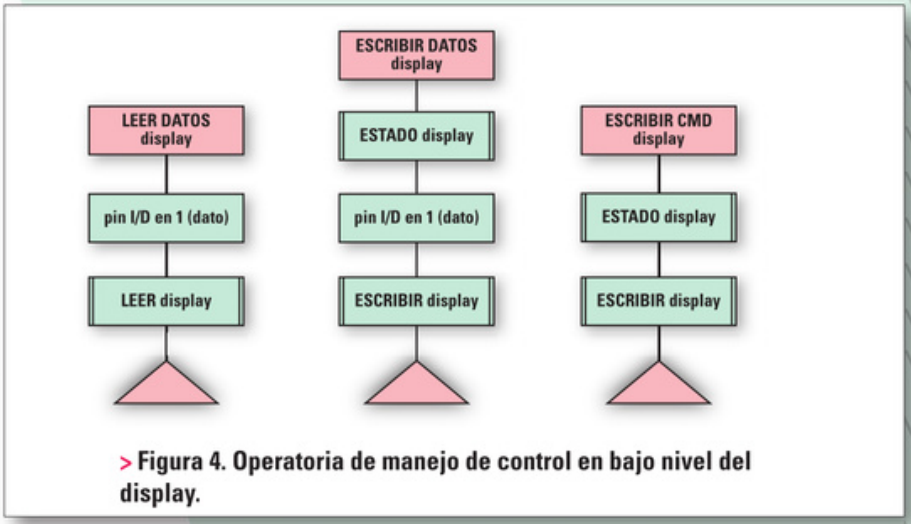
- Dado un punto con coordenadas **(x ; y)**, el controlador correspondiente es el izquierdo si **x < 64**. En caso contrario, es el derecho.
- La dirección dentro del controlador debe modificarse a **x = x - 64 si x ≥ 64**. En caso contrario, es el derecho.
- El número de página es la **parte entera** de dividir **y** por **8**.
- El bit es el **resto** de la división anterior; es decir, **y - 8 * página.**

PLOT, encender pixeles

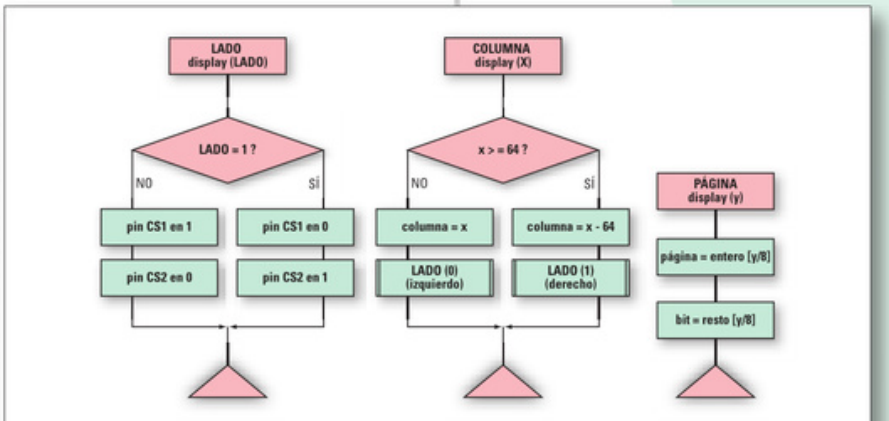
Para encender un punto determinado, leemos del display el contenido del byte que lo contiene, configuramos el bit que le corresponde, y lo volvemos a escribir. Al indicar la dirección, debemos hacer una lectura ficticia para darle tiempo al controlador. Antes de escribir, como el contador se autoincrementa, tenemos que volver a indicar la dirección.



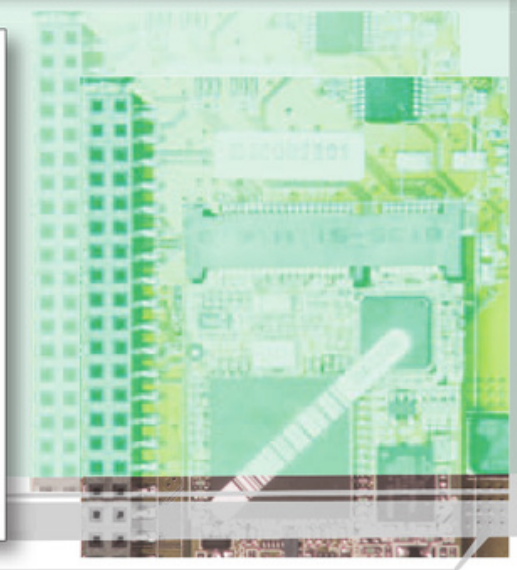
> Figura 3. Operaciones elementales de acceso al display. La operatoria es similar a lo visto para el display alfanumérico.



> Figura 4. Operatoria de manejo de control en bajo nivel del display.



> Figura 5. Algoritmos de posicionamiento en el display. Esta es la clave para graficar imágenes, textos y funciones.

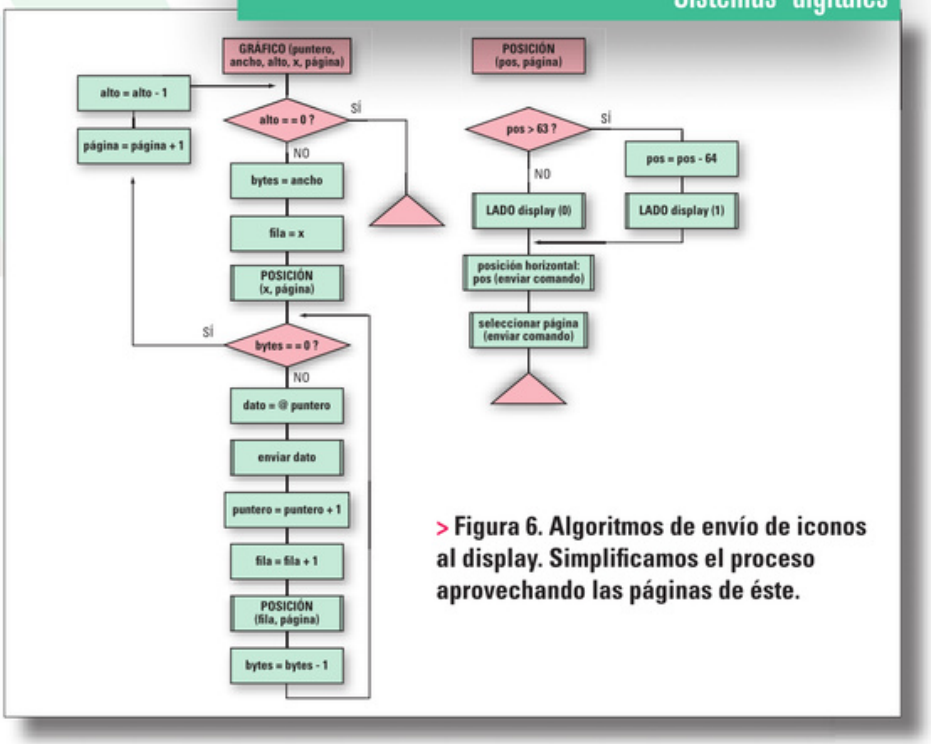


Cómo dibujar gráficos y funciones

Para enviar gráficos al display, conviene que estos sean de un tamaño vertical y que quepan en un número entero de páginas, es decir, 8, 16, 24 puntos, etc. De esta forma, el dibujo puede imaginarse dividido en páginas y columnas. La tarea requerida es la siguiente:

- Seleccionar la dirección inicial para la página.
- Enviar todos los bytes de dicha página, cuidando que no se produzca un cambio de controlador (**x≥64**), en cuyo caso deberemos repetir la operación de seteo de página para el controlador derecho.
- Repetir por el número de páginas.

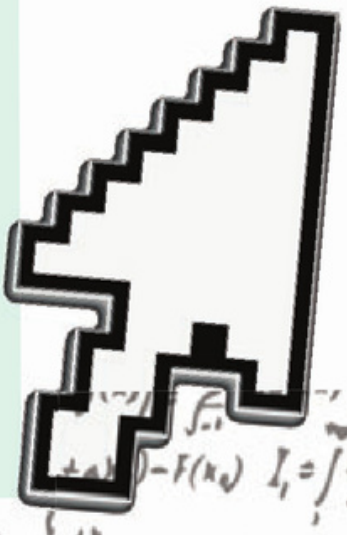
Por ejemplo, para mostrar un gráfico alojado en **imgdata**, de un cierto **alto** y **ancho**, en la página **pag**, posición **x** (**código 01**).



> Figura 6. Algoritmos de envío de iconos al display. Simplificamos el proceso aprovechando las páginas de éste.

> Código 01

```
while(alto--) {                                  // para cada fila
        bytes = ancho;                           // bytes a enviar por fila
        fila=x;                                  // posicion de inicio en la fila
        LADOdisplay(x>63);                       // 0a63 => lado 0; 63a127 => lado 1
        x=(x>63)?(x-64):x;                       // corrige x si x > 63
        ESCRIBIRCMDdisplay (0x40+x); // horizontal = x
        ESCRIBIRCMDdisplay(0xB8+pag); // página
        while(bytes--) {
                ESCRIBIRDATdisplay(*(imgdata++)); // datos
                if(++fila == 64) {               // cambio de controlador ?
                        fila=0;                  // corrige columna
                        LADOdisplay(1);          // controlador derecho
                        ESCRIBIRCMDdisplay (0x40); // horizontal = 0
                        ESCRIBIRCMDdisplay(0xB8+page); // página
                }
        }
        pag++;
}
```



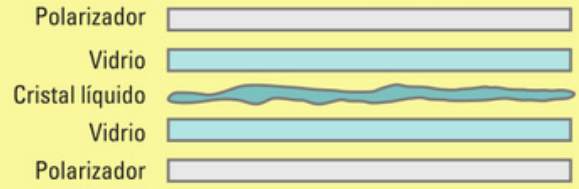
Dibujo de funciones

Para dibujar funciones, tomamos una muestra de esta para cada punto horizontal y calculamos la coordenada vertical de cada punto de la función. Por ejemplo, un coseno:

```
for(i = -64 ; i < 63 ; i++)
        PUNTOdisplay( 64 + i, 32 - (int) (32 * (cos (i * 3.14159 / 64))));
```

ARQUITECTURA DE UN DISPLAY DIGITAL

En esta infografía veremos la arquitectura interna de un display digital de tecnología LCD (Liquid Crystal Display) y las características técnicas de cuatro de los dispositivos más famosos del mercado.



Displays tipo TN (*Twisted Nematic*):

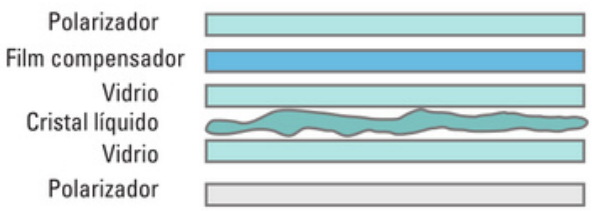
En estado de reposo, estos displays tienen 90° de rotación en la alineación de las moléculas de cristal líquido con respecto al área de transición de las dos superficies de vidrio. Los polarizadores utilizados en las superficies exteriores están orientados de manera de permitir el paso de la luz. Como sabemos, una vez aplicado un campo eléctrico, las moléculas se alinean con este impidiendo la rotación de los haces de luz y generando contraste entre zonas de "encendido" y "apagado".

Displays tipo HTN (*High Performance Twisted Nematic*):

Ofrecen mayor rotación, mejor ángulo de visión y menor costo que los materiales TN.

Displays tipo STN (*Super Twisted Nematic*):

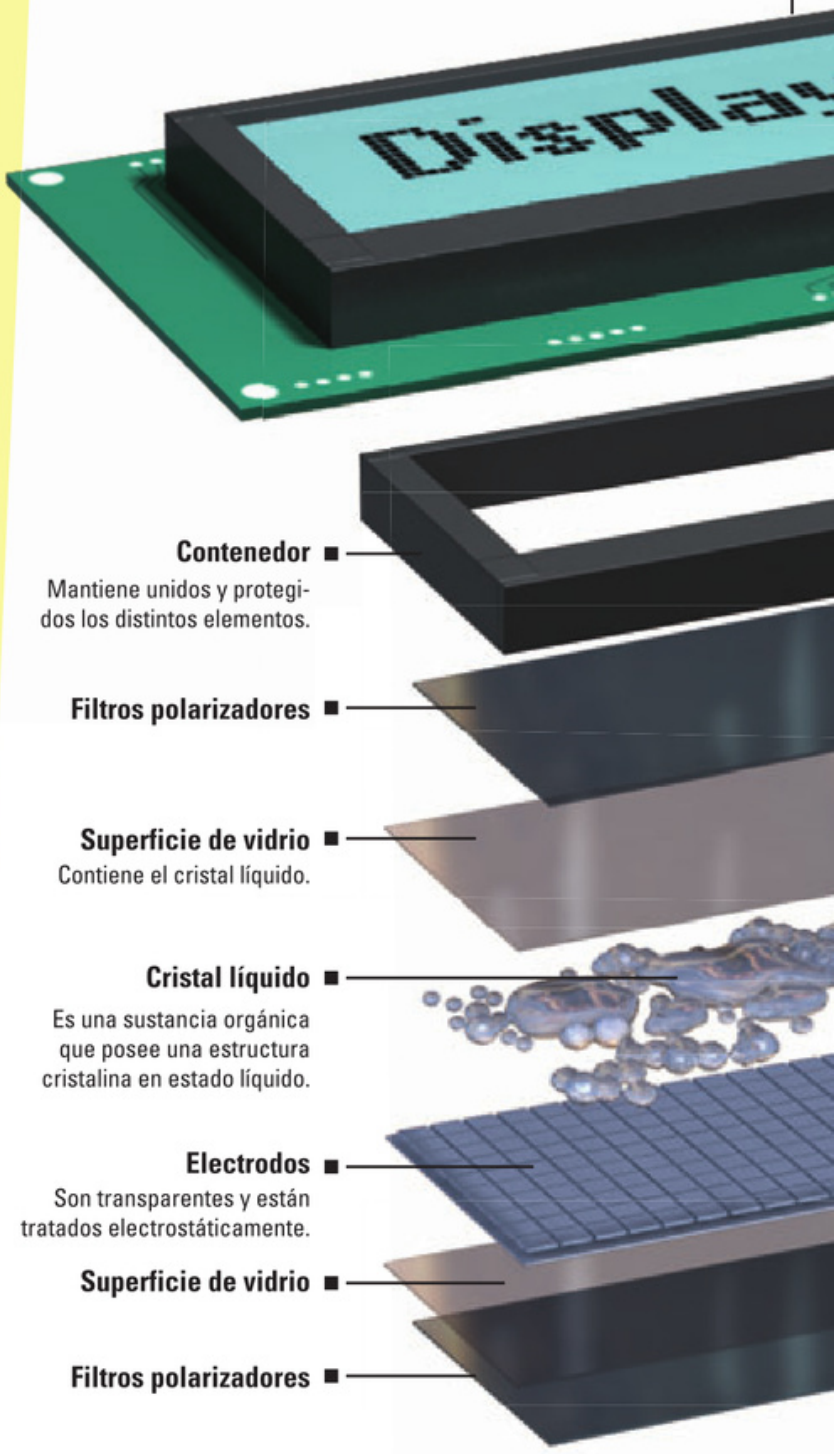
Ofrecen mayor ángulo de rotación (240° contra 90° del TN), con mayor contraste y ángulo de vista. Posee una desventaja, que es el efecto de birrefringencia o "doble refracción". Estos materiales desdoblan un rayo de luz incidente en dos linealmente polarizados en forma perpendicular entre sí. El efecto causado es que cambia el color de fondo a amarillo - verde y el de los caracteres a azul.



Displays tipo FSTN (Film Super Twisted Nematic):

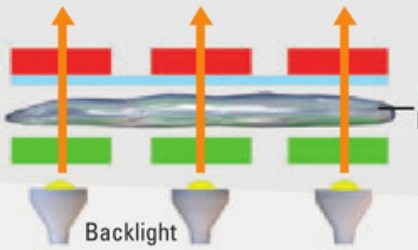
Este material agrega un film de retardo que compensa el efecto de birrefringencia. Este permite la existencia de displays blanco-negro, con excelente visualización cuando se utiliza backlight.

Los displays LCD no emiten luz ni ningún tipo de radiación, es por ese motivo se los agrupa dentro de los dispositivos "pasivos" de visualización. Utilizan la luz ambiente del entorno, por lo que consumen muy poca energía.



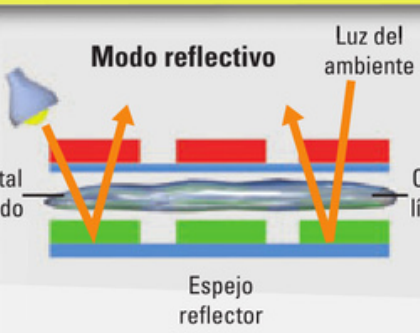
MÉTODOS DE ILUMINACIÓN

Modo transmisivo



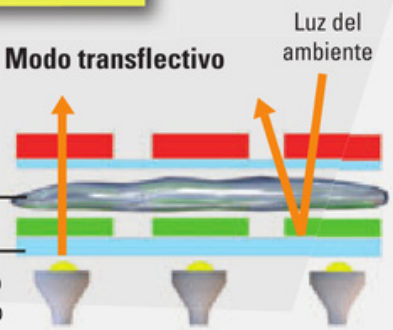
Ofrecen un aspecto distinto del resto debido a la luminosidad que ofrece el backlight siempre presente. Ideales para ser usados en ambientes oscuros.

Modo reflectivo

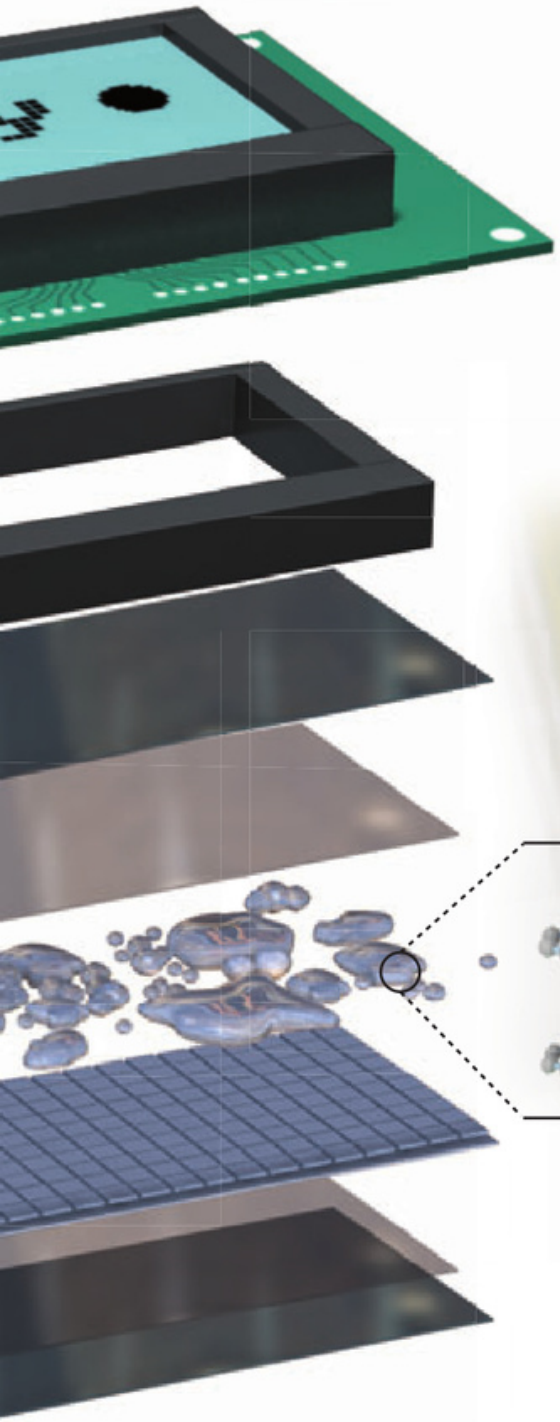


Poseen sobre el fondo un espejo que brinda reflexión completa. No usan backlights: son dispositivos de bajo consumo, ya que utilizan la luz del ambiente.

Modo transflectivo

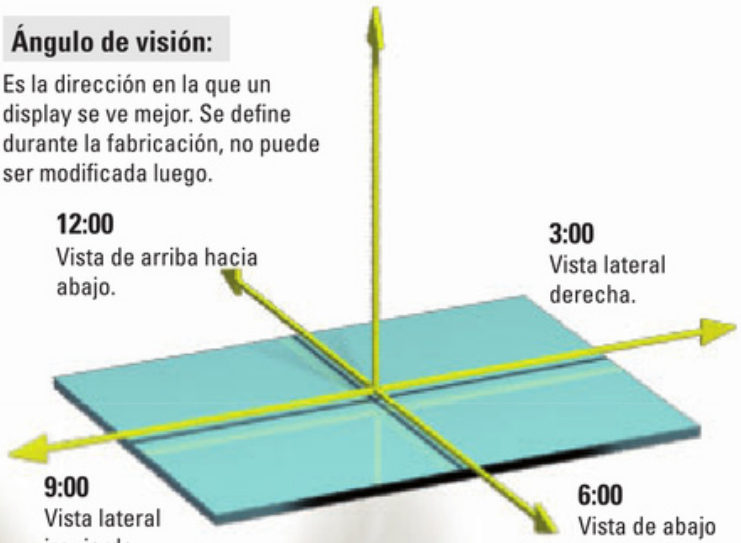


Combinan los efectos reflectivos y transmisivos. Pueden ser utilizados en cualquier situación de iluminación, contemplando, en malas condiciones, el encendido del backlight.



Ángulo de visión:

Es la dirección en la que un display se ve mejor. Se define durante la fabricación, no puede ser modificada luego.



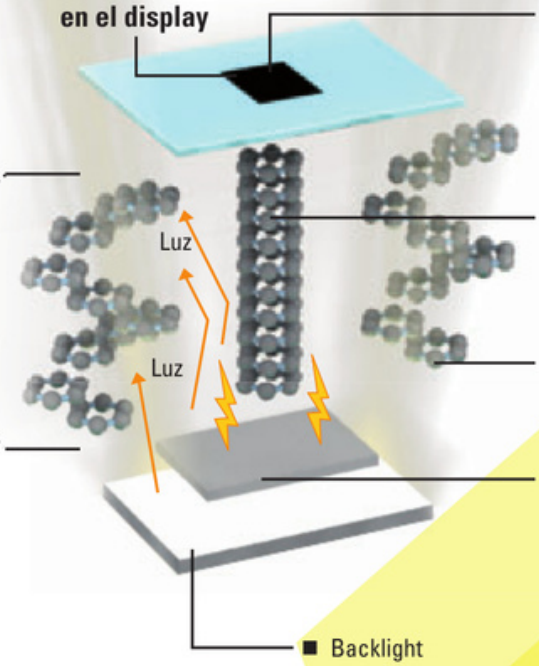
12:00
Vista de arriba hacia abajo.

3:00
Vista lateral derecha.

9:00
Vista lateral izquierda

6:00
Vista de abajo hacia arriba.

Zona oscura en el display



- Las moléculas alineadas crean un desfase de la luz producida por el backlight, que provoca un bloqueo de la luminosidad, y generando una zona oscura.
- El cristal líquido reacciona a la aplicación de un campo eléctrico, alineando sus moléculas con él.
- Las moléculas orgánicas de cristal líquido están alineadas en una estructura helicoidal.
- El electrodo aplica un campo eléctrico.
- Backlight

CÓMO DIBUJAR LETRAS

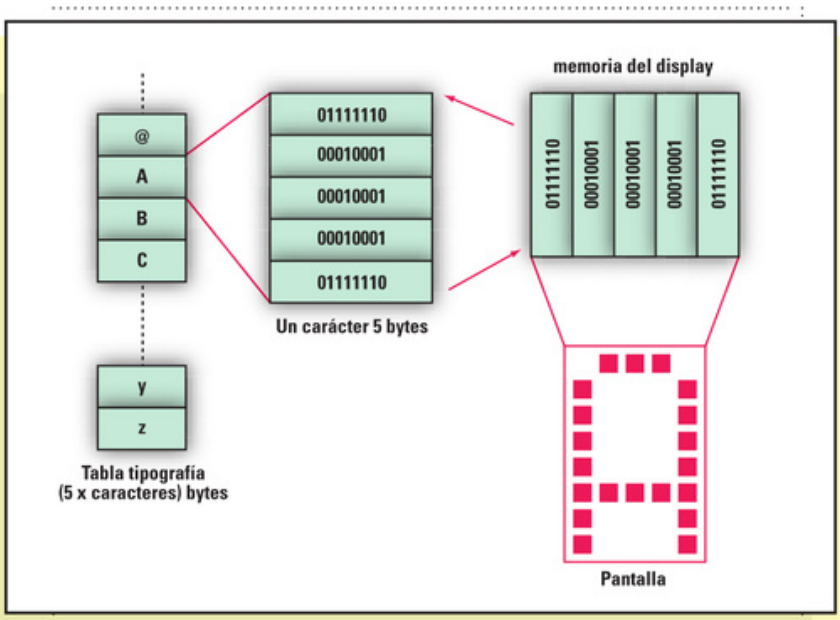
El controlador utilizado requiere que nosotros dibujemos los textos sobre la pantalla. Veremos aquí un algoritmo para resolverlo.

Asociando un **"renglón"** a cada página del display, y manteniendo todos los puntos de un carácter dentro del espacio de un mismo controlador, podemos simplificar en gran medida el proceso de dibujar caracteres. Si utilizamos una tipografía de **5 x 7**, tenemos 7 puntos de alto, que caben dentro de los 8 bits de una página, dejando separación entre líneas. Si dejamos un punto de separación entre caracteres en sentido horizontal, tenemos ocupados 6 puntos por carácter. Dentro del espacio de un controlador (64 puntos) caben, entonces, 10 caracteres (60 puntos), con 4 puntos sin usar a ambos lados. De este modo, tenemos la posibilidad de imprimir en una hoja de 8 renglones con 20 filas cada uno (10 filas por controlador). Por ejemplo, para imprimir un texto apuntado por string en el renglón **fila**, columna **col**, hacemos lo siguiente:

```
while (chr=*(string++)){
        x = (col>9)? 6 * (col-10) : 6*col; // corrige si col > 9
        if(col > 9) // 0 a 9 => lado 0; 10 a 20 => lado 1
                LADOdisplay(1);
        else {
                LADOdisplay(0);
                x += 4;    // dejo un borde de 4 pixels a los lados
        }
        ESCRIBIRCMDdisplay (0x40+x); // horizontal = x
        ESCRIBIRCMDdisplay(0xB8+fila); // página
        base = 5 * (chr - 0x20);
        for(i=0; i < 5; i++)
                ESCRIBIRDATdisplay( tabla5x7[ base + i] );
        ESCRIBIRDATdisplay(0);    // separador
        col++;
}
```

Dibujar caracteres

Los caracteres se guardan en una tabla de **5 bytes** por carácter; cada byte contiene una columna del carácter. Copiamos esos 5 bytes a las 5 columnas de la página deseada del display. Recordemos que una columna del display contiene **8 bits** en sentido vertical, con el menos significativo arriba. El carácter se forma en pantalla como vemos en la **Figura 7**.

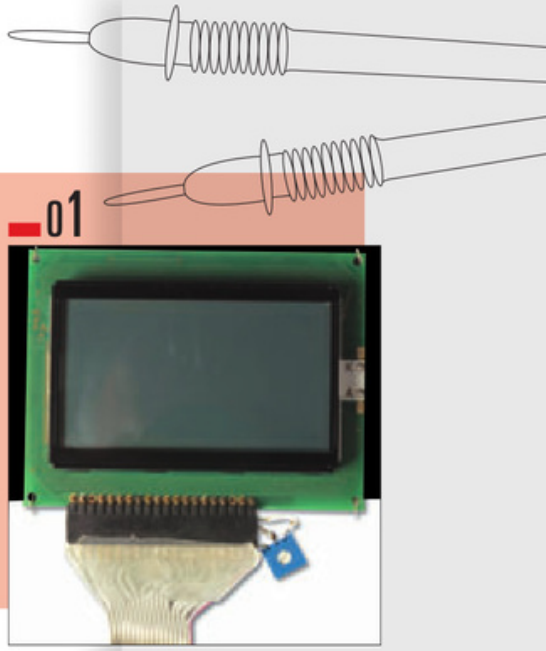


> **Figura 7. El proceso de dibujar una letra en pantalla. Las tipografías se arman en una tabla.**

PASO A PASO

CÓMO DIBUJAR GRÁFICOS Y TEXTOS

Analizaremos algunos detalles prácticos del diseño y armado de la placa de conexión con el display.

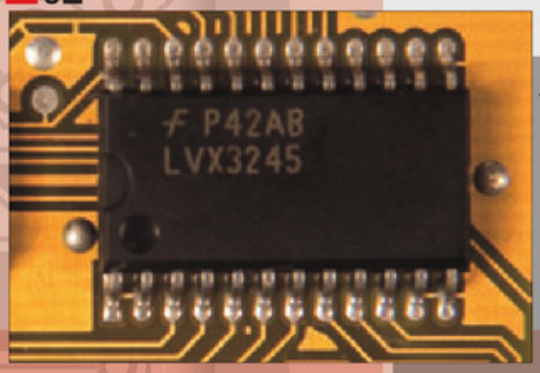


01

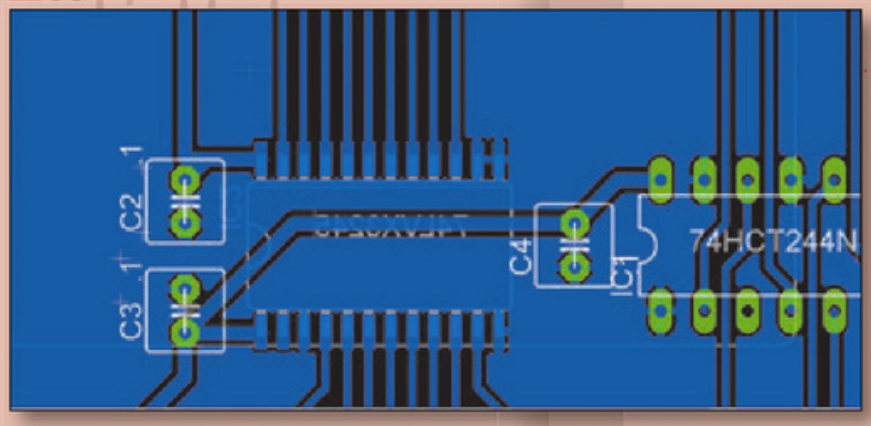
Vamos a compilar, depurar y observar algunos detalles de operación para dibujar gráficos y textos. Recordemos que si deseamos trabajar con un micro de 5 V, podemos conectarnos directamente al display. En este caso, las instrucciones son para el kit de desarrollo de RCM5700, que incluye el cable USB y una placa de soporte para conectarnos con el módulo. ■



02



03



01. Los displays vienen en varios formatos. Los más comunes son pines en una línea y en dos líneas; en nuestro caso, usamos el primero. Es ideal soldar una tira de pines a 90° del display, y usar una hembra para pines a la que le soldamos un cable plano.

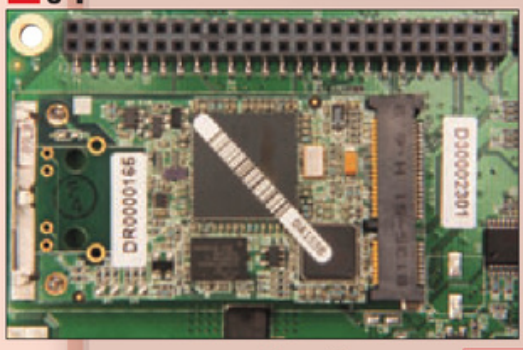
02. El 74HCT244 se consigue en encapsulado DIP, pero el 74LVX3245 es SMD. Por esta razón, recomendamos realizar una placa de circuito impreso en Pertinax. De igual modo, podemos hacer el diseño en Eagle.

03. En el diseño del PCB, debemos prestar atención a los capacitores de 1 µF, que deben estar cerca de los chips correspondientes. La conexión a la fuente de alimentación (5 V) tiene que hacerse cerca del display. Para la conexión al display y micro, prevemos tiras de pines.

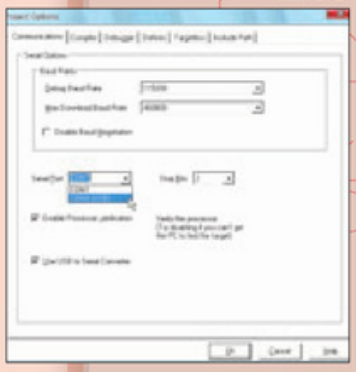
ELECTRONICAL CHARACTERISTICS						
ITEM	SYMBOL	CONDITION	STANDARD			UNIT
			MIN	TYP	MAX	
Input voltage	Vdd	+ 5V	4.7	5.0	5.5	V
Supply current	Idd	Vdd=5V	---	8	---	mA
Recommended LCD driving voltage for normal temp version module	Vdd-V0	0 °C	---	9.8	---	V
		25 °C	---	9.5	---	
		50 °C	---	9.3	---	
LED forward voltage	VF	25 °C	---	4.2	4.5	V
LED forward current	IF	25 °C	---	360	---	mA

Verificamos la conexión del backlight en la hoja de datos del display y calculamos el valor de R2 = (5 V - VLED) / ILED. El fabricante nos da VLED e ILED máximas. Debemos elegir una corriente menor, digamos, de la mitad.

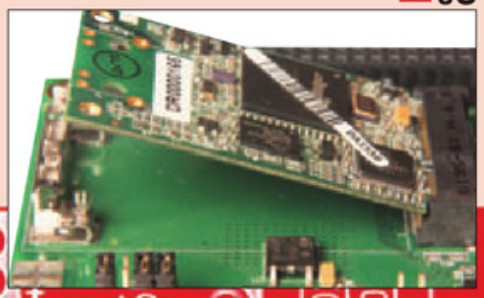
04



06



05





09

```
void ESCRIBIRdisplay(int data)
{

   BitWrPortI ( PEDR, &PEDRShadow, 0, 0 );       // Baja R/W (Write)

   BitWrPortI ( PDDR, &PDDRShadow, 1, 0 );       // LVX como salidas

   WrPortI ( SPCR, &SPCRShadow, 0x84 );          // PA0-7 como salidas
   WrPortI ( PADR, &PADRShadow, data );          // escribe datos

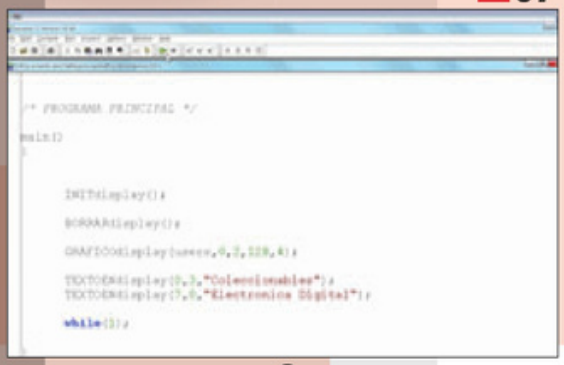
   // tiempo de setup, R/W antes de E: ~150ns
   BitWrPortI ( PEDR, &PEDRShadow, 1, 2 );       // Sube E

   // ancho mínimo de E, ~500ns
   BitWrPortI ( PEDR, &PEDRShadow, 0, 2 );       // Baja E

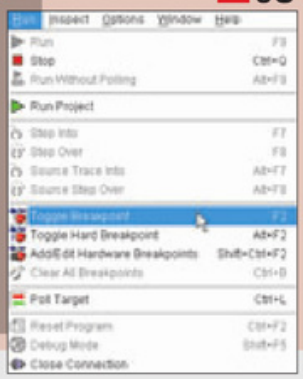
   // ciclo mínimo de E: ~1us (volver a llamar a esta función)
}

int LEERdisplay()
{
```

07



08



04. Para la conexión con el micro, podemos usar un esquema similar al del display con pines, conector hembra y cable plano.
Del lado que conectemos a nuestra placa de desarrollo, usamos la conexión correspondiente. En el caso del RCM5700, disponemos de una hembra de pines doble; podemos usar una tira de pines y soldarles el cable.

05. El código para Dynamic C se encuentra en: **www.redusers.com/cursos/**

electronica digital. El compilador se provee con el kit o podemos bajarlo desde **www.rabbit.com.** Luego, insertamos el módulo RCM5700 en la placa de desarrollo. A continuación, debemos conectar la placa de desarrollo por el conector mini-USB a un puerto USB de la PC.

06. Una vez que la PC reconoce el puerto que provee el chip FTDI de la placa de desarrollo, lo configuramos en Dynamic C, eligiendo en el menú **Options** la opción **Project Options** y, luego, la solapa **Communications**.



Si trabajamos con un micro de 5 V, podemos conectarnos directamente al display.

10



11



07. Abrimos el archivo **demoLCD.c**, lo compilamos y ejecutamos presionando el icono de reproducción o la tecla <**F5**>. Si queremos ejecutar el programa paso a paso, podemos detenerlo con el icono correspondiente y avanzar mediante las teclas <**F7**> y <**F8**>, como en la mayoría de los entornos de desarrollo.

08. Posicionamos el cursor y ponemos un breakpoint en la función **ESCRIBIRDATdisplay()**, mediante la opción **Run:Toggle Breakpoint** o pulsando la tecla <**F2**>; veremos un rectángulo rojo. Cuando la ejecución se detiene en dicha función, observaremos un rectángulo verde.

09. Ejecutamos paso a paso la función mediante la tecla <**F7**>, y podemos ir observando, con un multímetro u osciloscopio, cómo cambia el estado de los pines y el orden de activación del conversor de nivel, el display y el puerto de I/O del micro.

10. Eliminamos el breakpoint y colocamos otro en la función **GRAFICOdisplay()**. Ejecutamos el código y, cuando tenemos el cursor de color verde, ejecutamos paso a paso sin entrar en las demás funciones, con la tecla <**F8**>. Observamos cómo los datos al display van siendo enviados uno a uno.

11. Terminado el proceso, vemos el logo completo. Luego, podemos eliminar el breakpoint y poner uno en el bloque for(), que realiza el dibujo de un carácter en la función **TEXTOENdisplay()**. Ejecutamos paso a paso mediante la tecla < > y observamos cómo se dibuja el carácter en el display.

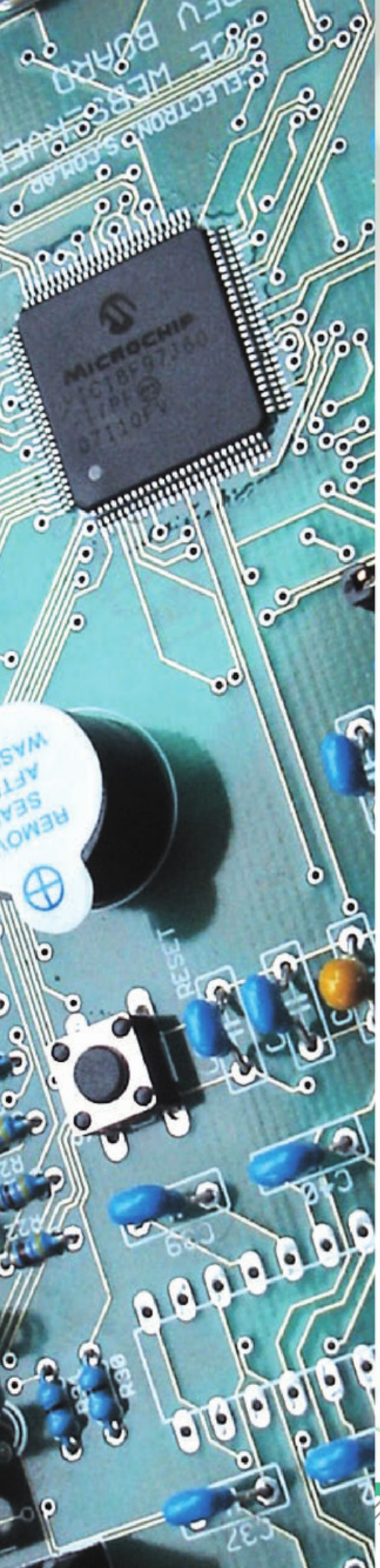
ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué son los displays gráficos LCD?
2. ¿Cuál es la estructura de la memoria de un display gráfico?
3. ¿Cómo se realiza el control de contraste?
4. Caracterice la arquitectura de un display LCD.
5. Describa la forma en que se dibujan letras en un display LCD.
6. ¿Qué características posee un display LCD en estado de reposo?

EJERCICIOS PRÁCTICOS

1. Ejemplifique algunos algoritmos que se utilizan para escribir textos en un display LCD.
2. Dibuje una letra en un display LCD.
3. Dibuje un gráfico en un display LCD.



CAPÍTULO 9

NETWORKING CON TCP/IP

Networking
Modelo OSI
TCP/IP
Resolución de nombres: DNS
TCP y UDP
Conexión a servidores web
Conexión a un web server vía Telnet
El lenguaje HTML
Ethernet con pic
Otras alternativas
Actividades

09

NETWORKING

Nos introduciremos en el mundo de las redes informáticas para conocer las distintas clasificaciones, su alcance y topología.

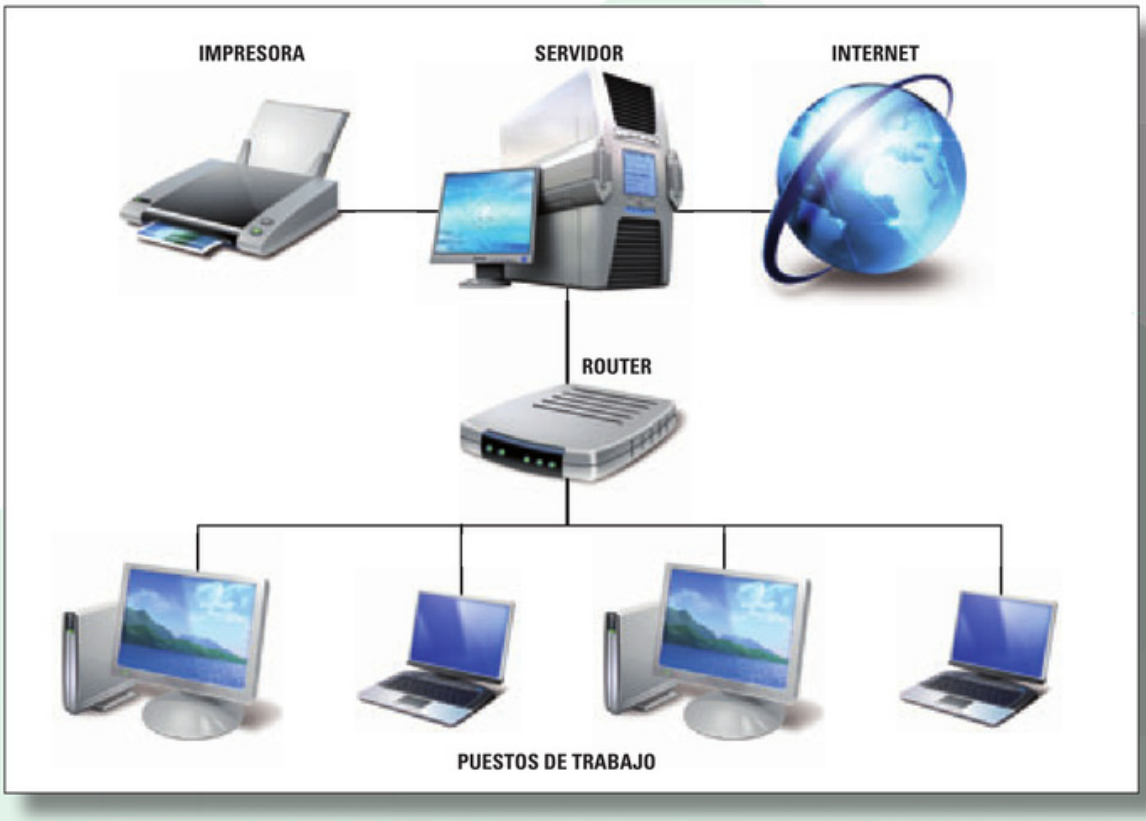
Antes de abordar este tema, surge una pregunta clave: ¿qué son las redes informáticas? Pues bien, se trata de un conjunto de dispositivos interconectados, que permiten el intercambio de información entre ellos. Esta es una definición bastante amplia, pero es la base fundamental del tema que trataremos en este apartado.

Clasificación de las redes

Las redes pueden clasificarse de distintas maneras. Una de ellas es por el alcance: es así que tenemos redes tipo **LAN**, **MAN** y **WAN**.
También podemos agruparlas según su topología, es decir, la disposición física de interconexión de los distintos dispositivos que las integran: **bus, estrella**, **árbol**, **malla**, **anillo**, **anillo doble** y **mixta**.
Otra posibilidad es hacerlo de acuerdo con la forma de transmisión de datos, en cuyo caso pueden ser **simplex** (de transmisión simple), que es unidireccional (un dispositivo transmite y otro recibe); **half duplex** (bidireccionales), donde ambos dispositivos pueden transmitir y recibir, pero no al mismo tiempo; y **full duplex** (bidireccionales), en las que los dos dispositivos transmiten y reciben simultáneamente.

Redes LAN, MAN y WAN

Las redes tipo **LAN** (Local Area Network, o red de área local) son de pequeña extensión (entre 10 metros y 1 kilómetro). Se las emplea generalmente en el ámbito de un edificio u organización. Las de tipo **MAN** (*Metropolitan Area Network,* o red de área metropolitana) tienen una extensión mayor (hasta



> **Figura 1. Conexión de red LAN montada en una oficina. Hay un servidor principal, una base de datos, una impresora y cuatro puestos de trabajo.**

10 kilóme-tros), en tanto que las **WAN** (*Wide Area Network*, o red de área amplia) se extienden aún más que las ante-riores (entre 100 y 1000 kilómetros).

Topología bus

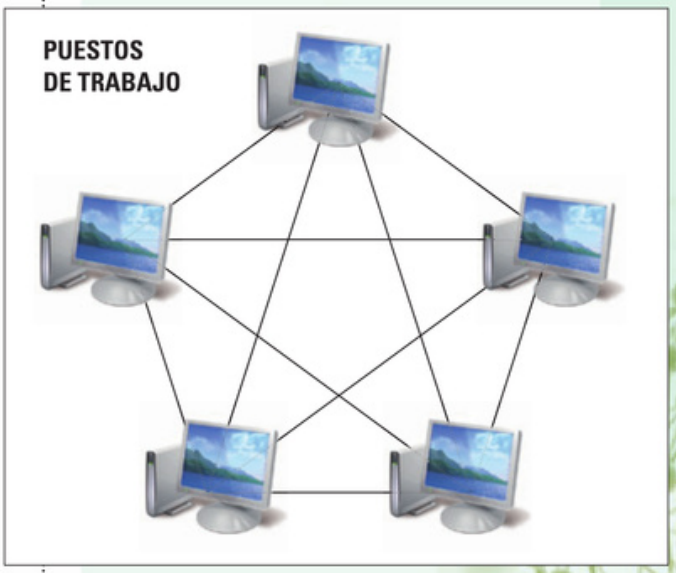
Consta de un bus principal, al cual se conectan todos los dispositivos de la red. El medio de transmisión típico es el **cable coaxil**. Para la construcción de esta red, se requiere que en ambos extremos del cable principal se coloque un terminador, que no es más que una resistencia cuyo fin es adaptar las impedancias. La principal ventaja de esta topología es su facilidad de instalación y posterior ampliación. Como desventaja, podemos mencionar que ante un problema físico en la red, como algún cable cortado, falla la red completa.

Topología malla

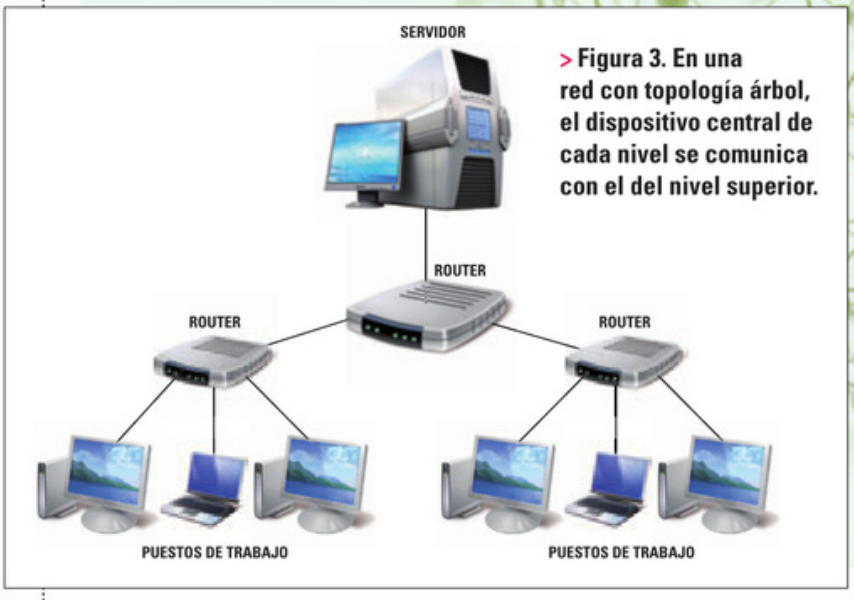
En este caso, cada dispositivo de la red se conecta directamente con los demás. En una malla completa, todos los dispositivos están conecta-dos con todos. Una de las ventajas es que los paquetes de datos pueden viajar por diferentes caminos. La desventaja es el alto costo de imple-mentación, debido a la gran cantidad de conexiones que se requiere.

Topología estrella

En esta modalidad, todos los dispositivos de la red se conectan a un único dispositivo central, que puede ser un hub (concentrador), un switch (conmutador), un router (enrutador) o, directamente, otra computadora. De esta forma, todas las comunicaciones pasan a través de dicho dispositivo central. Uno de los puntos a favor de esta topología es que, ante un problema en alguna de las estaciones de trabajo o en el medio de transmisión entre esta y el dispositivo central (algún cable cortado, por ejemplo), el resto de la red continúa funcionando normalmente. El punto negativo es que una falla en el dispositivo central afecta a toda la red.



> **Figura 2. En una red con topología malla, todos los dispositivos se conectan entre sí.**



> **Figura 3. En una red con topología árbol, el dispositivo central de cada nivel se comunica con el del nivel superior.**

> TOPOLOGÍA ÁRBOL

Puede considerarse como varias topologías estrella, en donde el dispositivo central de cada una se conecta con un dispositivo central de nivel superior, siguiendo la forma de un árbol de conexiones. La ventaja principal de esta modalidad es que resulta sencillo ampliar la red. La desventaja es que si falla alguno de los dispositivos centrales de alguna rama, la rama completa se verá afectada.

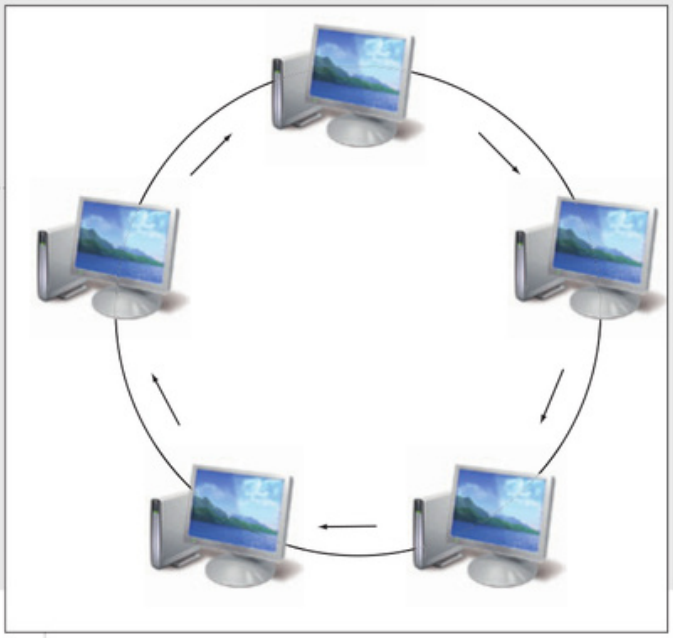


Figura 4. Podemos observar la distribución de una topología anillo, en donde los dispositivos se conectan uno a uno para componer una estructura de esa forma.

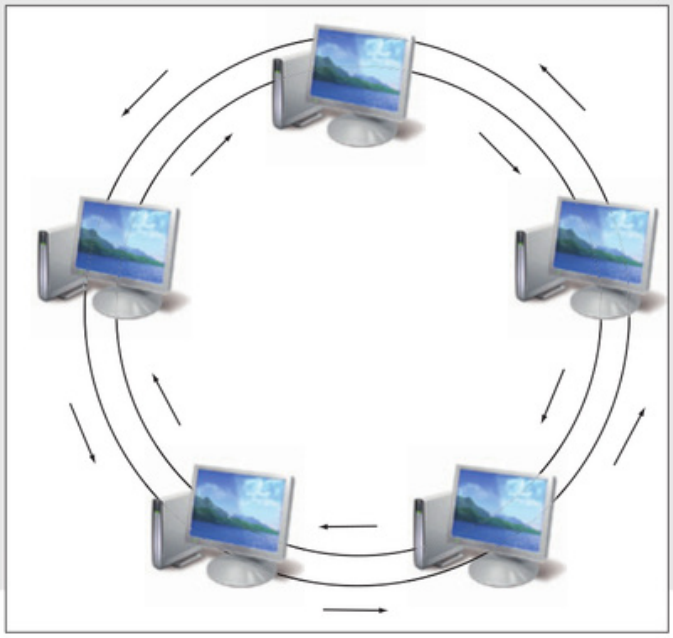


Figura 5. Observamos una topología anillo doble, con la cual logramos redundancia en la transmisión de los datos y, por lo tanto, reducción de fallas.

Topología anillo

En una topología anillo, los dispositivos de la red se conectan uno a uno formando un anillo; es decir, cada uno se conecta con el siguiente, y el último, con el primero. En este caso, los paquetes de datos viajan de un dispositivo a otro en una única dirección a través del medio de transmisión. Cada dispositivo verifica a quién va dirigido ese paquete. En caso de que no sea dirigido a él, lo retransmite al siguiente hasta llegar a destino.

Dispositivos de red

Entre los diferentes dispositivos de red existentes, podemos nombrar el hub o concentrador, el switch o conmutador, y el router o enrutador. Un **hub** es un dispositivo que trabaja en la capa física (capa 1 del modelo OSI, que veremos más adelante) y permite estructurar el cableado de una red de manera de poder expandirla.

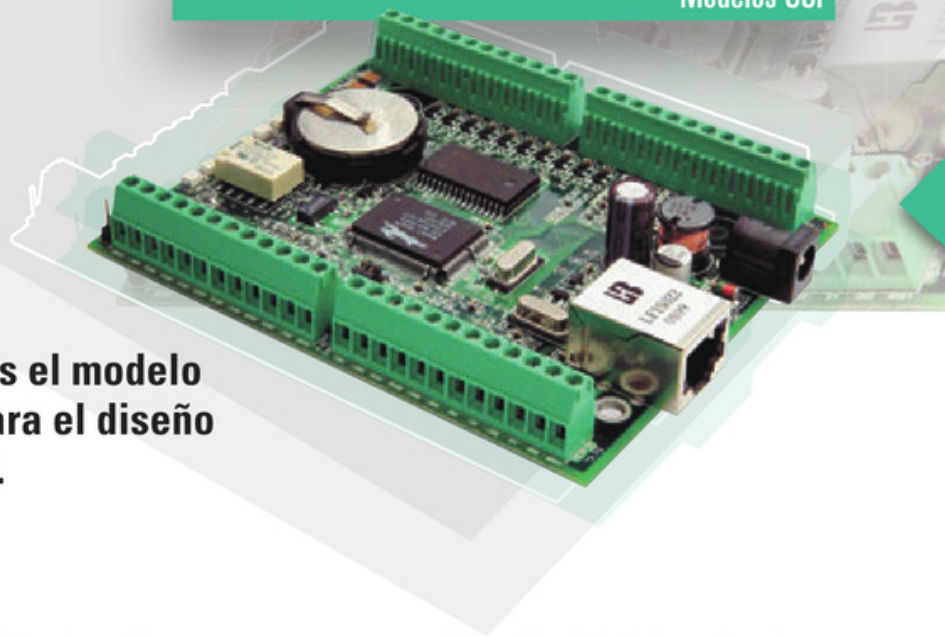
El **switch** trabaja en la capa de enlace de datos (capa 2 del modelo OSI). Se dice que es como un "hub inteligente", ya que elige el mejor camino para enviar los datos.

Por último, el **router** actúa en la capa de red (capa 3 del modelo OSI) y una de sus funciones es brindar conectividad con Internet a los distintos dispositivos que la integran. Al trabajar en esa capa, maneja direcciones IP, las cuales detallaremos más adelante.

Una típica configuración de red hogareña con conexión a Internet de banda ancha se realiza conectando la salida del módem entregado por el proveedor de Internet a la entrada de un router. Las salidas del router se conectarán a las interfaces de red de cada PC o dispositivo de la red. ■

> CONCEPTO DE TOKEN

Para evitar que varios dispositivos envíen datos a la red al mismo tiempo, surge el concepto de token, que no es más que una serie de bits que viajan por la red. El token es capturado por cada dispositivo, y le permite acceder al medio en ese momento. Cuando el dispositivo recibe el token, puede enviar datos por la red; mientras que cuando el dispositivo de destino recibe el dato, le envía una señal al de origen para indicar que la operación se realizó correctamente.



MODELO OSI

En este apartado explicaremos el modelo de referencia OSI, utilizado para el diseño de sistemas de comunicación.

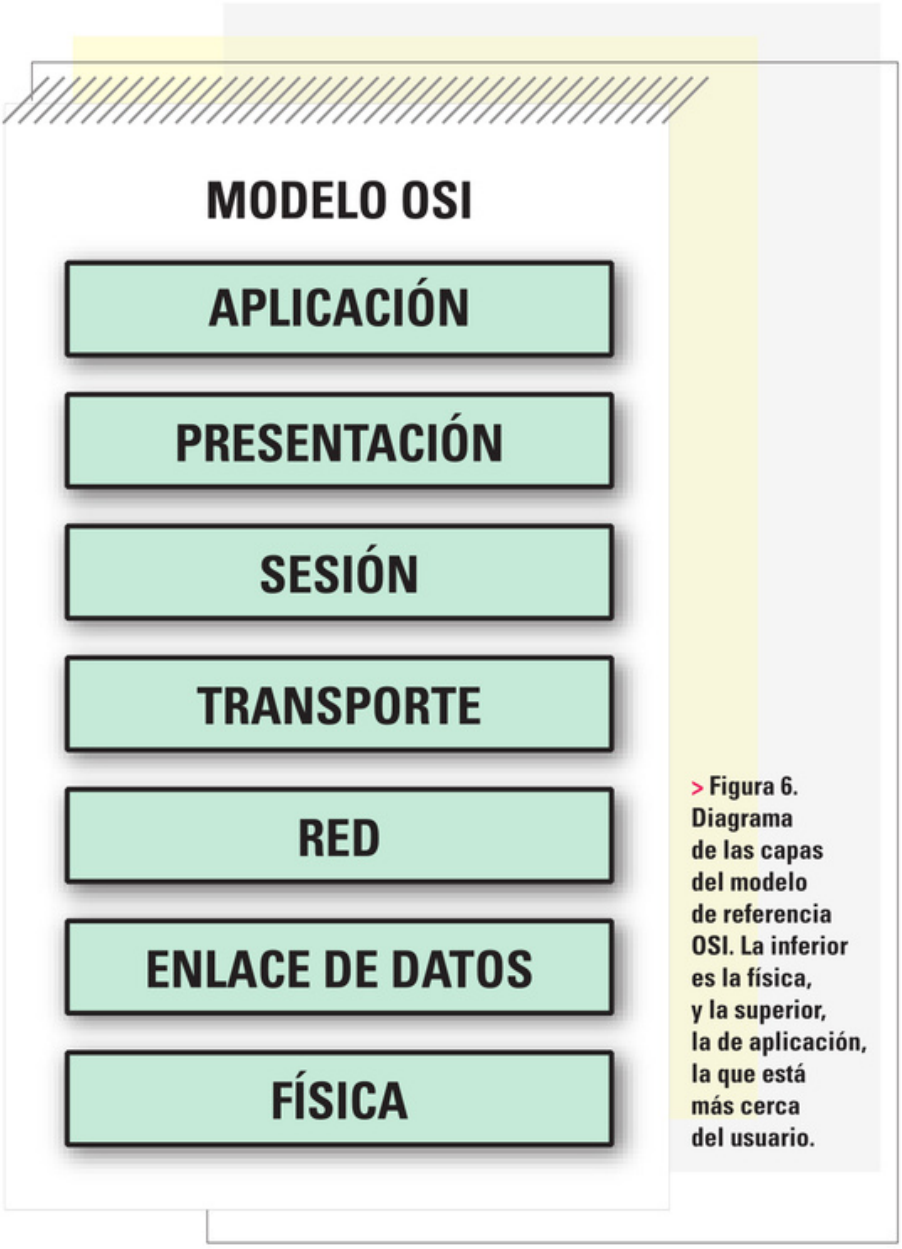
El modelo de referencia OSI (*Open System Interconnection*, o interconexión de sistemas abiertos) fue creado en el año 1984 por la ISO (*International Organization for Standardization*, Organización Internacional para la Estandarización). El objetivo era solucionar los problemas de compatibilidad que existían entre los sistemas provenientes de distintos fabricantes. Este modelo define una estructura de siete capas, en donde cada una es completamente independiente del resto y soluciona un aspecto particular del sistema de comunicación. Veamos cada una de ellas.

Capa 1: Física

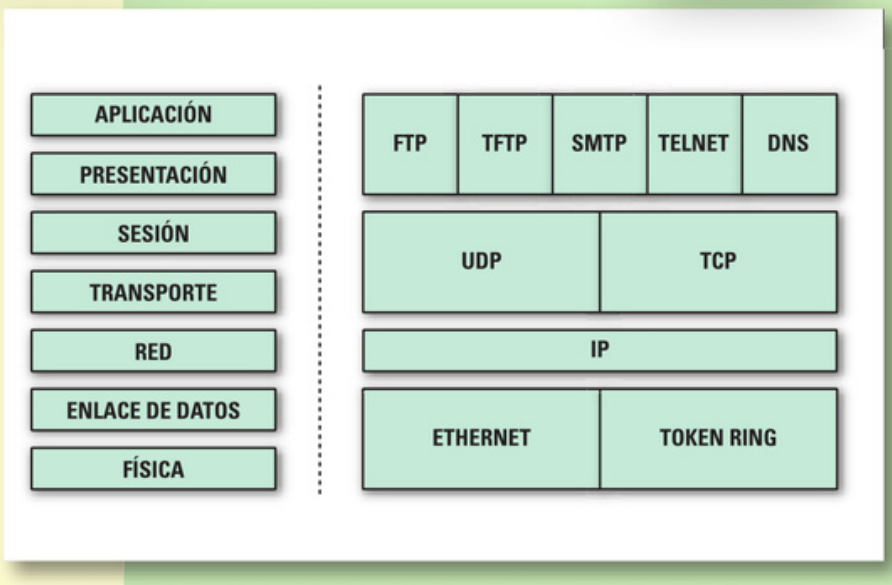
La capa física es la que se encarga de transmitir los bits (unos y ceros), ya sea por un cable (medios guiados) o en forma inalámbrica (medios no guiados). En esta capa se definen, entre otras cosas, los niveles de tensión en los cuales se representan los unos y los ceros, los conectores empleados, el tipo de medio guiado (por ejemplo, coaxil, par trenzado o fibra óptica), el tipo de medio no guiado (como infrarrojo, microondas o radiofrecuencia), la velocidad de transmisión, y otros parámetros. En definitiva, todos los aspectos físicos requeridos para transportar los bits de información.

Capa 2: Enlace de datos

La función principal de esta capa es permitir la comunicación con el dispositivo inmediato, y que esta sea libre de errores. Esta capa delimita la cantidad de bits que maneja la capa física. Por lo tanto, envía los bits en forma de tramas, agregando una codificación al inicio (encabezado o header) y otra al final (cola o tail). De esta manera, se permite la detección y recuperación de errores. Ethernet ocupa tanto esta capa como la 1.



> **Figura 6. Diagrama de las capas del modelo de referencia OSI. La inferior es la física, y la superior, la de aplicación, la que está más cerca del usuario.**



> **Figura 7. Algunos de los protocolos más importantes de TCP/IP y la capa del modelo de referencia OSI en que se encuentran.**

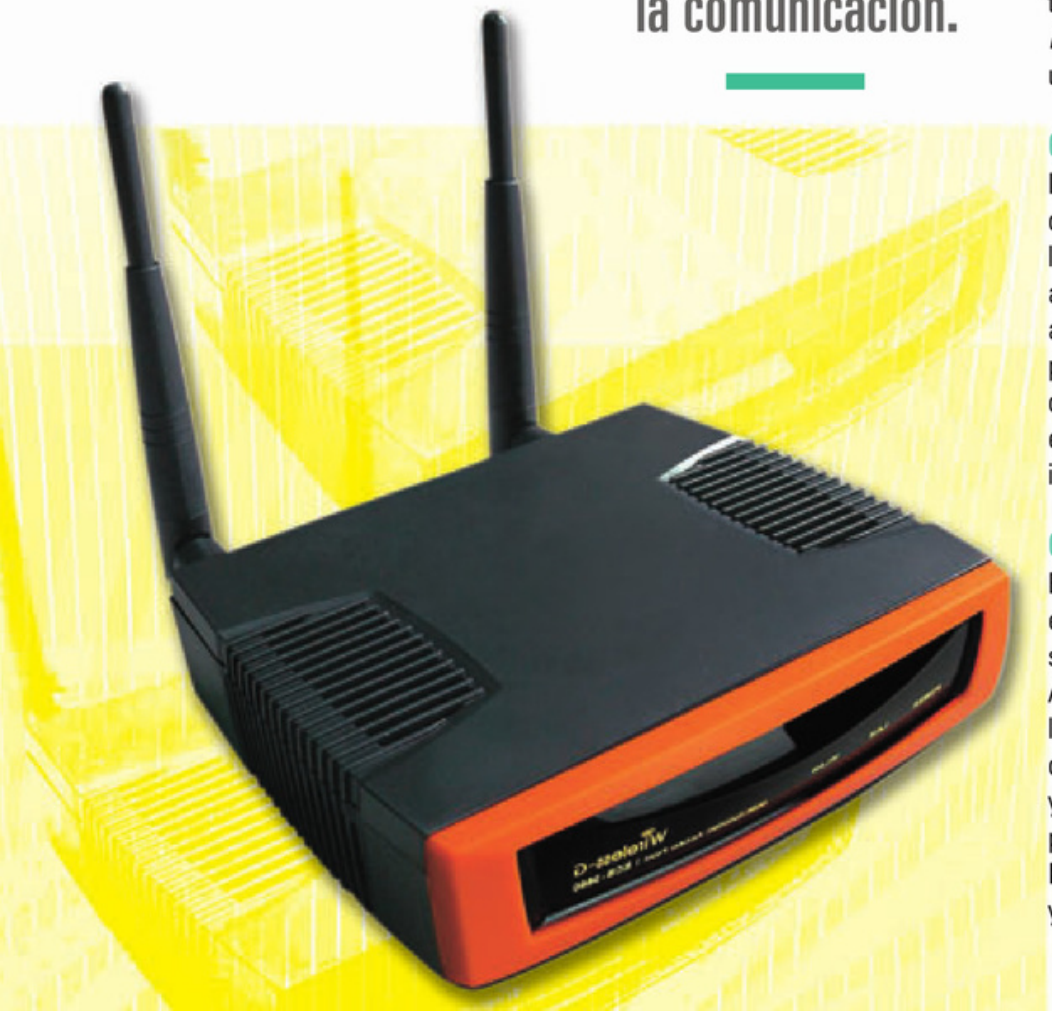
Capa 3: Red

La función de la capa de red es enrutar y enviar los paquetes de información entre las diferentes redes. Define el camino por donde viajarán los datos para llegar a destino. De esta forma, se encarga de controlar la congestión en la red. En esta capa se habla de los **protocolos IP** (*Internet Protocol*, o protocolo de Internet), **NetBIOS** (*Network Basic Input/Output System*) y **NetBEUI** (*NetBIOS Extended User Interface*, o interfaz extendida de usuario de NetBIOS), entre otros.

Capa 4: Transporte

Es la capa **intermedia** del modelo, que actúa como nexo entre los primeros tres niveles (encargados, básicamente, de la comunicación) y los tres últimos (más orientados a la aplicación). En ella se define, por ejemplo, si los paquetes de información se entregarán en el mismo orden en el que fueron enviados o en cualquier orden. Esta capa agrega un **header**, que gestiona las distintas maneras de operación; también garantiza la integridad de la información y el control de flujo. En esta capa se habla de **TCP** (*Transmission Control Protocol*, o protocolo de control de transmisión), **UDP** (*User Datagram Protocol*, o protocolo de datagrama de usuario), y otros.

La capa de sesión se encarga de comenzar, mantener y terminar la comunicación.

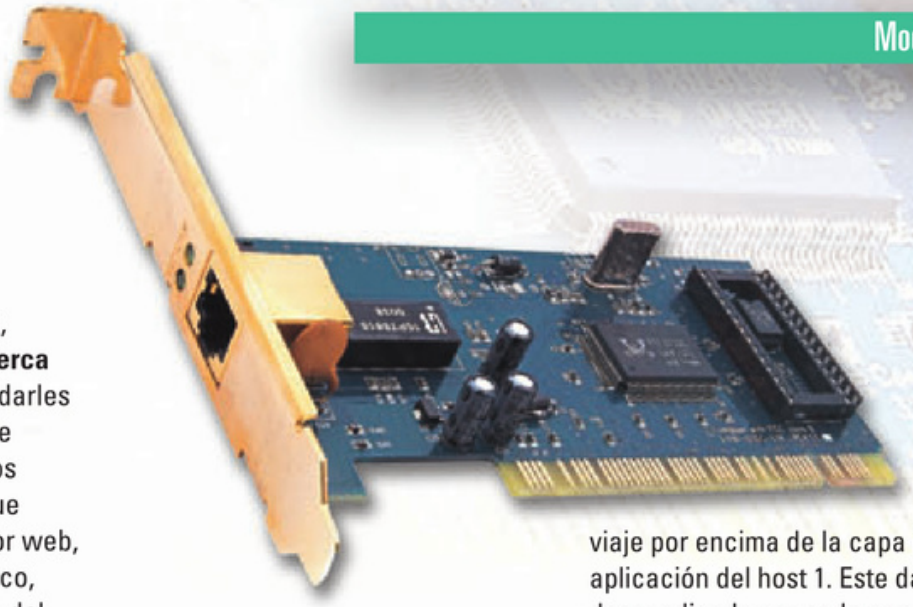


Capa 5: Sesión

La capa de sesión es la encargada de comenzar, mantener y terminar la comunicación. Brinda servicios a la capa de transporte, como el agregado de puntos de sincronismo para la recuperación de transferencias extensas, en donde la comunicación es interrumpida por algún motivo.

Capa 6: Presentación

El objetivo de esta capa es lograr que la información sea representada correctamente. Además, se ocupa de tomar la información proveniente de la capa de aplicación y darle un formato entendible para las demás capas. Permite, además, la encriptación y la compresión de datos.



Capa 7: Aplicación

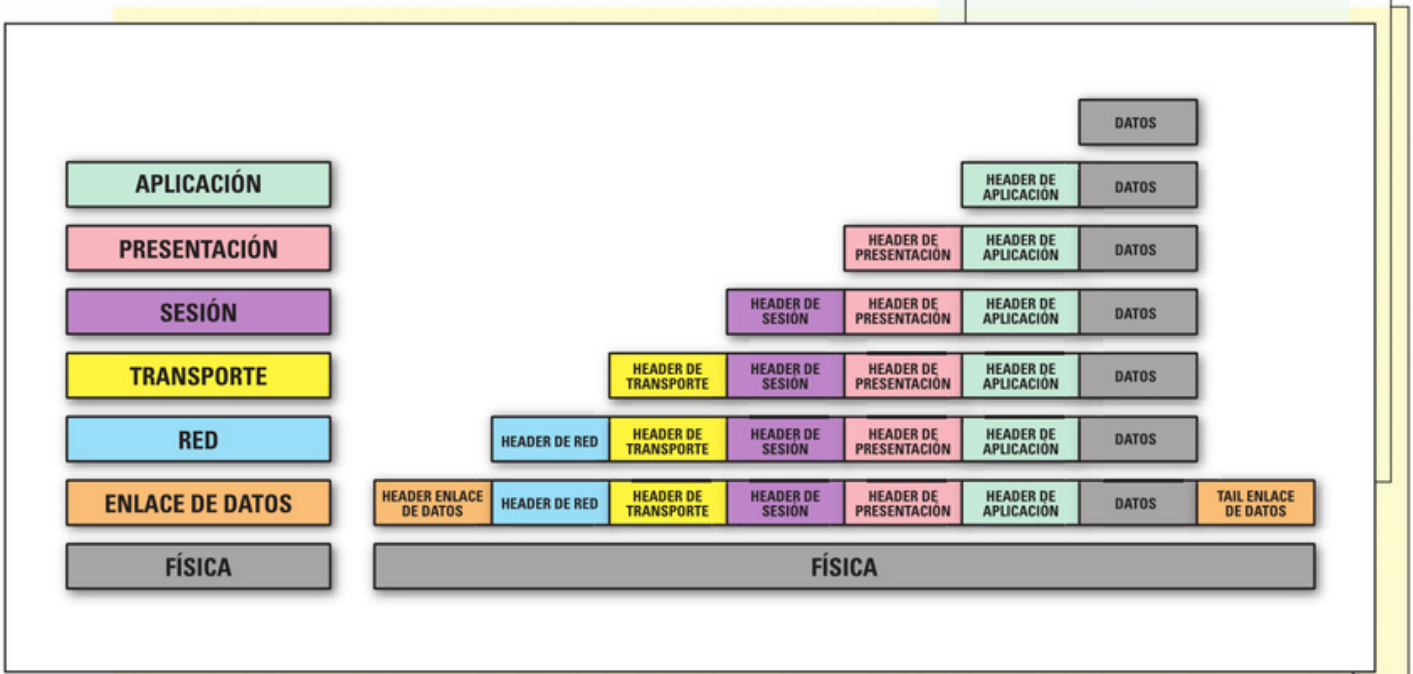
Es la **última** capa del modelo de referencia OSI, con lo cual, es la que se encuentra **más cerca del usuario**. Se ocupa de brindarles servicios a las aplicaciones de los usuarios. Cuando hablamos de aplicaciones, lo primero que se nos ocurre es un navegador web, un cliente de correo electrónico, etc. Sin embargo, en el marco del modelo de referencia OSI, se refiere a los **protocolos de red** que emplean las aplicaciones con las que interactúa una persona. Por ejemplo, cuando navegamos por Internet, no leemos directamente el **código HTML** (*HyperText Markup Language*, o lenguaje de marcado de hipertexto), sino que lo hacemos mediante el uso de un navegador web, que interpreta el código y lo traduce en un formato más amigable para el usuario. En esta capa se habla de **HTTP** (*HyperText Transfer Protocol*, o protocolo de transferencia de hipertexto), **SMTP** (*Simple Mail Transfer Protocol*, o protocolo simple de transferencia de correo) y **FTP** (*File Transfer Protocol*, o protocolo de transferencia de archivos), entre otros.

Cada capa es independiente y soluciona un aspecto particular del sistema de comunicación.

Flujo de información entre capas

Imaginemos que queremos enviar un mensaje desde un **host 1** a un **host 2**, ambos ubicados en la misma red local. El mensaje se interpreta como un dato, que comienza su viaje por encima de la capa de aplicación del host 1. Este dato irá descendiendo por cada capa, y cada una de ellas le agregará una codificación al inicio (encabezado o header), antes de pasarlo a la capa inferior. La capa de enlace de datos agrega, además, una codificación al final (cola o tail). La física toma esta secuencia de bits y la manda a través de la red, hasta que llega a la capa física del host 2. Una vez allí, la información va subiendo por cada capa, cada una de las cuales se encarga de quitarle el encabezado correspondiente, hasta que, finalmente, pasa a la capa de aplicación del host 2. ■

> **Figura 8. Flujo de información entre capas. Vemos cómo se va formando el mensaje en cada una, al agregarle el header/tail (encabezado/cola) correspondiente.**



TCP/IP

En estas páginas explicaremos los conceptos fundamentales del modelo TCP/IP, que es la base de Internet. Veremos también el direccionamiento IP.

El modelo **TCP/IP**, cuya sigla significa *Transmission Control Protocol* (protocolo de control de transmisión) e *Internet Protocol* (protocolo de Internet), nació en los Estados Unidos a mediados de los años 70. Fue desarrollado por **DARPA** (*Defense Advanced Research Projects Agency*, o Agencia de Proyectos de Investigación Avanzada), bajo el proyecto **ARPANET**.

Estructura TCP/IP

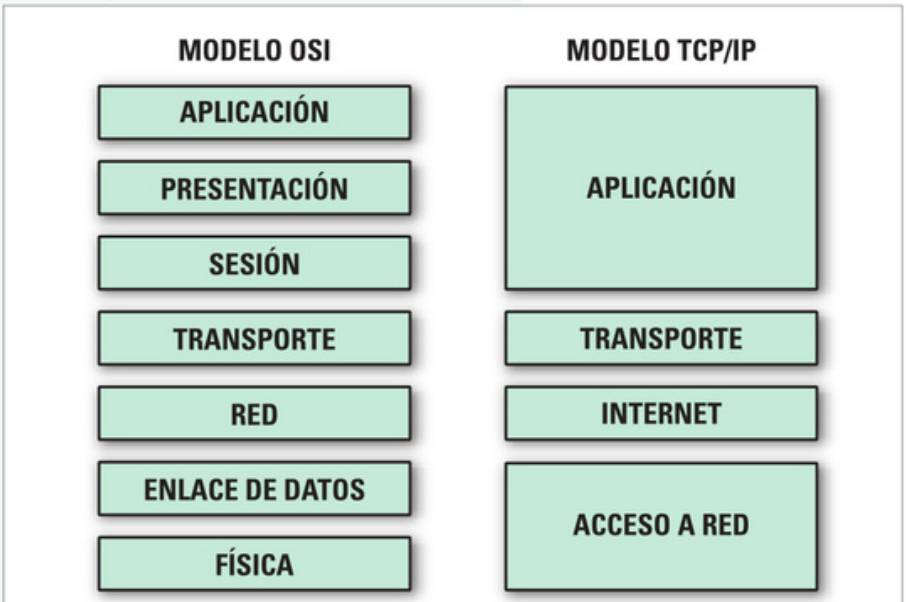
TCP/IP es un conjunto de protocolos que permiten realizar el transporte confiable de datos por las redes. Al igual que el modelo OSI, el TCP/IP está dividido en capas, en este caso, cuatro: acceso a red, Internet, transporte y aplicación. Veamos en detalle cada una de ellas.

Capa 1: Acceso a red

Es la capa inferior del modelo TCP/IP. Si hacemos una analogía entre el modelo OSI y el TCP/IP, corresponde al conjunto de capas 1 y 2 del primero (capa física y de enlace de datos). En esta capa se habla de *Ethernet* y *token ring*, entre otros conceptos.

Capa 2: Internet

Se encarga de enrutar los paquetes para enviarlos a otras redes. Es análoga a la capa 3 del modelo OSI (capa de red). En esta capa se habla de protocolo **IP**, **ARP** (*Address Resolution Protocol*, o protocolo de resolución de direcciones) e **ICMP** (*Internet Control Message Protocol*, o protocolo de mensajes de control de Internet).



> **Figura 9. Comparación entre las siete capas del modelo de referencia OSI y las cuatro del TCP/IP.**

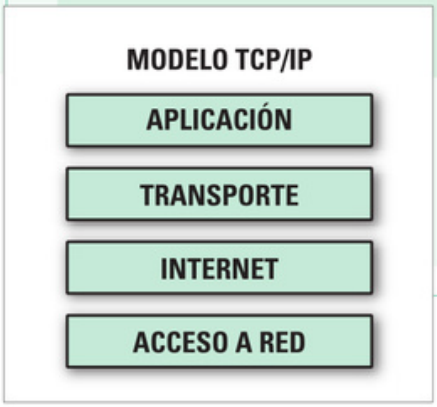


Figura 10. Capas del modelo TCP/IP. La inferior es la de acceso a red, y la superior, la de aplicación, la que está más cerca del usuario.

Capa 3: Transporte

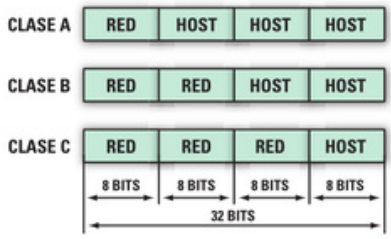
Corresponde a la capa 4 del modelo OSI (capa de transporte). Se ocupa de transportar los mensajes sin importar el tipo de red empleada. En esta capa se habla de TCP y UDP, dos protocolos que veremos más adelante.

CLASES DE REDES

CLASE	BITS	1er BYTE	FORMATO	RANGO
A	0xxxxxxx	0 – 127	red.host.host.host	1.0.0.0 – 126.255.255.255
B	10xxxxxx	128 – 191	red.red.host.host	128.0.0.0 – 191.255.255.255
C	110xxxxx	192 – 223	red.red.red.host	192.0.0.0 – 223.255.255.255
D	1110xxxx	224 – 239	-	224.0.0.0 – 239.255.255.255
E	1111xxxx	240 – 255	-	240.0.0.0 – 255.255.255.255

Tabla 1. Detalle de cada una de las cinco clases de redes.

El modelo TCP/IP está dividido en cuatro capas: acceso a red, Internet, transporte y aplicación.



> **Figura 11. Cada octeto corresponde a la dirección de red o de host, dependiendo de la clase de red.**

Capa 4: Aplicación

Es la capa superior del modelo TCP/IP, correspondiente a las capas 5, 6 y 7 del modelo OSI (sesión, presentación y aplicación). Aquí se habla de SMTP y FTP, entre otros.

Protocolo IP

El protocolo IP está ubicado en la capa 3 del modelo OSI (capa de red). Es el principal de esta capa y permite el intercambio de datos en una red. Se trata de un protocolo **no orientado a la conexión**, ya que cada paquete que se envía puede tomar distintos caminos entre el origen y el destino, con lo cual pueden llegar desordenados. También es un protocolo **no fiable**, debido a que los paquetes de datos pueden dañarse o perderse.

> IP ESPECIAL Y RESERVADA

Existen direcciones con significados especiales. Por ejemplo, la 0.0.0.0 (este host en esta red) es la que posee cada dispositivo cuando aún no se le ha asignado una IP. La dirección **loopback** es la 127.0.0.1, y es la de nuestro propio equipo. Por otro lado, están las direcciones reservadas para redes privadas, que no existen en Internet; ellas son: 10.0.0.0 a 10.255.255.255, 172.16.0.0 a 172.31.255.255 y 192.168.0.0 a 192.168.255.255.

Direcciones IP

Para identificar un dispositivo en una red, a cada uno se le asigna una dirección, llamada **IP**, que si bien es única dentro de la red, puede no ser fija y modificarse de manera dinámica. Las direcciones IP están formadas por una serie de **32 bits**, separados en **cuatro octetos** que se representan en formato decimal, por ejemplo: 192.168.0.3.

Ahora bien, estas direcciones se dividen en dos partes: **red** y **host**. La parte que indica la red define hacia qué red va dirigido el paquete de datos. Por otro lado, la parte de **host** señala hacia qué equipo dentro de esa red se lo dirige.

Se dice que es un direccionamiento jerárquico, ya que existen diferentes niveles. Las redes se dividen en cinco clases: **A**, **B**, **C**, **D** y **E**, identificadas con el primer octeto. Las de clase **A** son redes grandes, que cuentan con muchos dispositivos. Las de clase **B** son medianas; las **C** son redes pequeñas, con pocos dispositivos; en tanto que las **D** son multicast, en donde el envío de datos es recibido por múltiples dispositivos; y las **E** son para experimentación y prueba. ■

RESOLUCIÓN DE NOMBRES: DNS

En las siguientes páginas explicaremos de qué forma se traducen las direcciones de dominio empleadas diariamente, en direcciones IP.

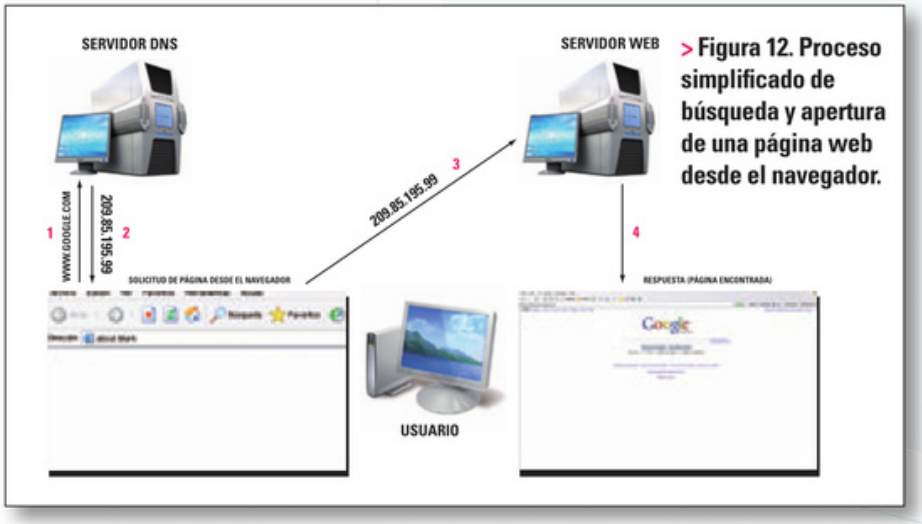
Como mencionamos en secciones anteriores, cada dispositivo conectado a una red debe tener asignada una dirección IP, mediante la cual se podrán direccionar datos entre diferentes dispositivos. Con esta información, podría surgir la pregunta: ¿cómo puedo acceder a una página web sin saber su IP? La respuesta es muy sencilla: mediante el **DNS** (*Domain Name System*, o sistema de nombre de dominio).

Sistema de nombre de dominio

El DNS es una base de datos distribuida y jerárquica en donde se almacena diferente información, entre la que se encuentra la asociación entre un nombre de dominio y una dirección IP.

> La función de los servidores DNS es traducir los nombres de dominio en direcciones IP.

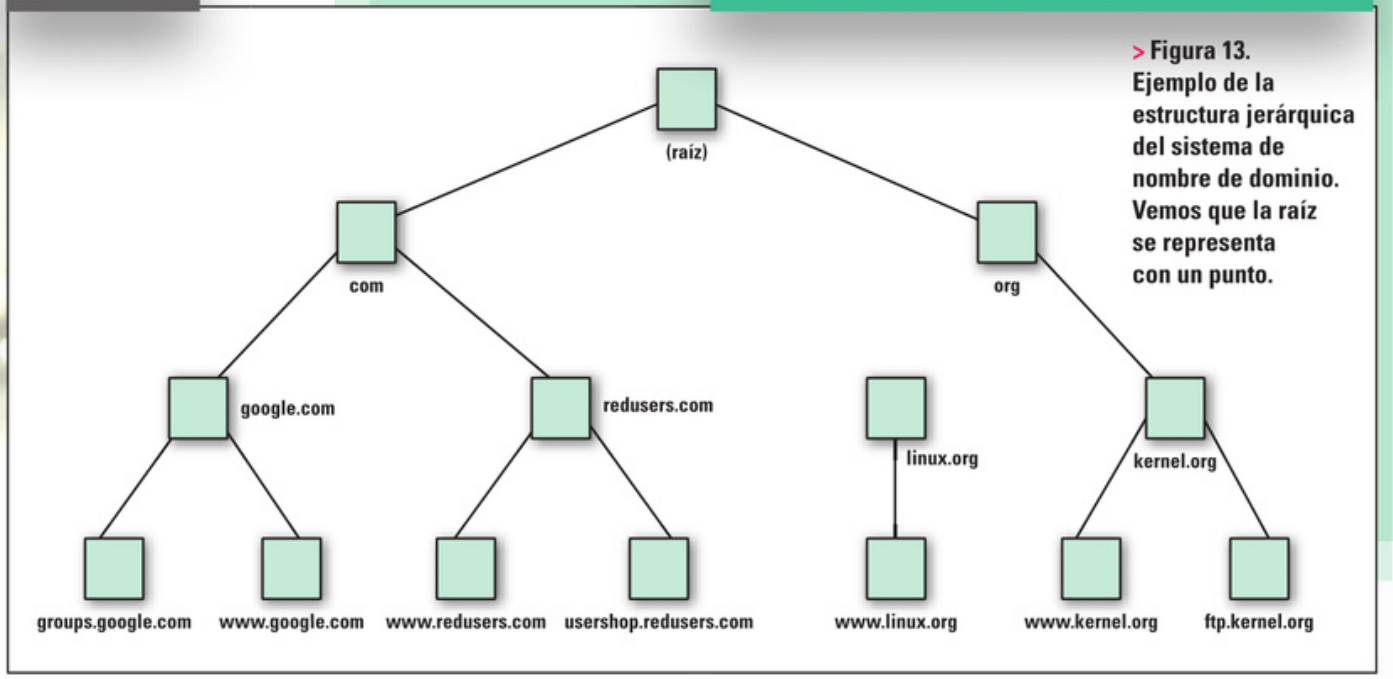
Veamos un ejemplo. Supongamos que queremos acceder al sitio de Google. Para hacerlo, simplemente escribimos **www.google.com** en la barra de dirección del navegador. Ahora bien, también podríamos acceder a la misma página ingresando **209.85.195.99** en la barra, que corresponde a la IP del servidor web de Google.Cuando decimos que el DNS es una base de datos distribuida, nos referimos a que no está ubicada en un servidor en particular, sino que se encuentra dispersa en diversos servidores. Por otra parte, es jerárquica porque el sistema de nombre de dominio tiene una estructura de árbol, dividido en dominios y subdominios, cada uno de ellos, separado por un punto.
En un nombre de dominio completo, la jerarquía debe leerse de derecha a izquierda. Por ejemplo, para el dominio **www.google.com**, el dominio de jerarquía más alta es **.com**. La raíz de este árbol se representa con un punto al final del nombre **www.google.com**.



> Figura 12. Proceso simplificado de búsqueda y apertura de una página web desde el navegador.

Servidores DNS

La función principal de los servidores DNS es traducir los nombres de dominio en direcciones IP. Existen distintos tipos de servidores DNS: primarios y secundarios. Cada dominio y subdominio posee un servidor primario y varios secundarios como respaldo ante una eventual falla. El primario es el que cuenta con la información original, en tanto que los secundarios obtienen la información solicitándola en forma periódica al primario.



> **Figura 13.** Ejemplo de la estructura jerárquica del sistema de nombre de dominio. Vemos que la raíz se representa con un punto.

> PROCESO DE RESOLUCIÓN

Supongamos que deseamos acceder al sitio groups.google.com. En primer lugar, el sistema se contactará con un servidor de DNS local. Este interrogará a un servidor de DNS raíz, preguntando por la dirección indicada. La respuesta será una lista de servidores para el dominio .com. Posteriormente, nuestro servidor interrogará a esos nuevos servidores por el nombre de dominio completo. De esa forma, obtendremos la lista de servidores correspondiente al dominio google.com.



Caché de DNS

Existe un mecanismo de caché mediante el cual se agiliza la búsqueda en servidores de DNS. El funcionamiento es de la siguiente manera. Cuando intentamos acceder a una dirección web, el primer servidor DNS al que se acude es al de **nuestro propio ISP** (*Internet Service Provider*, o proveedor de servicios de Internet). Si este no la localiza en ese servidor, mediante una búsqueda jerárquica intenta ubicarla en otros servidores DNS, hasta hallarla. Ahora, supongamos que otro usuario del mismo ISP quiere acceder a la misma dirección web. En ese caso, el servidor DNS del ISP ya tiene almacenada esa dirección gracias a la búsqueda efectuada previamente; es decir que, a medida que un usuario intenta acceder a distintas direcciones, los servidores DNS van guardando esa información, mecanismo llamado **caché de DNS**.

La información guardada en la caché del servidor DNS no queda allí para siempre, sino que se elimina después de un cierto tiempo, denominado **TTL** (*Time To Live*, o tiempo de vida); este es un parámetro asignado por el administrador a cada servidor. ■

UDP Y TCP



Analizaremos los dos protocolos más utilizados que posee la capa de transporte. Veremos las características de cada uno y sus diferencias.

Los protocolos UDP y TCP están ubicados en la capa 4 del modelo OSI (de transporte). Ambos utilizan el protocolo IP para el transporte de los mensajes. La diferencia fundamental entre ellos es que UDP es un protocolo **no orientado** a la conexión, mientras que TCP está orientado a ella.

Protocolo UDP

Su función es proporcionar comunicación entre las aplicaciones de dos equipos. Emplea el protocolo IP para el transporte de los mensajes y, al igual que este, es **no orientado a la conexión y no fiable**. En el primer caso, debido a que no se establece una conexión previa entre los dos equipos para la transmisión de los mensajes, con lo cual existe la posibilidad de que estos no lleguen ordenados al destino. En el segundo caso, porque los mensajes pueden llegar dañados o perderse. Tampoco existe confirmación de llegada por parte del receptor, con lo cual no hay manera de saber si alcanzaron el destino correctamente. Las aplicaciones que empleen este protocolo deberán prever el modo de asegurar que la información que llegue sea la adecuada.

Datagrama UDP

El datagrama UDP cuenta con cinco campos: cuatro de ellos corresponden a la cabecera, y el quinto, a los datos. La cabecera está compuesta por puerto de origen, puerto de destino, longitud del mensaje y suma de verificación. Los **puertos** son números de **16 bits** que permiten asociar las diferentes aplicaciones con un mismo host. Así, por ejemplo, **TFTP** (*Trivial File Transfer Protocol*, o protocolo trivial de transferencia de archivos) emplea el **puerto 69** de **UDP**. La longitud del mensaje se mide en bytes, incluyendo la cabecera. Este protocolo es muy utilizado para la transmisión de video y sonido, en donde se necesita una alta velocidad de respuesta, sin importar que se pierdan algunos datos.

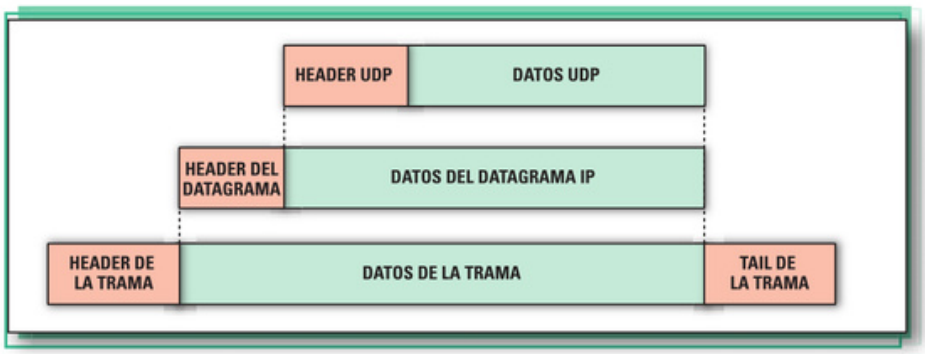


Figura 14. En este esquema ejemplificamos la manera en que un mensaje UDP viaja sobre el datagrama IP.

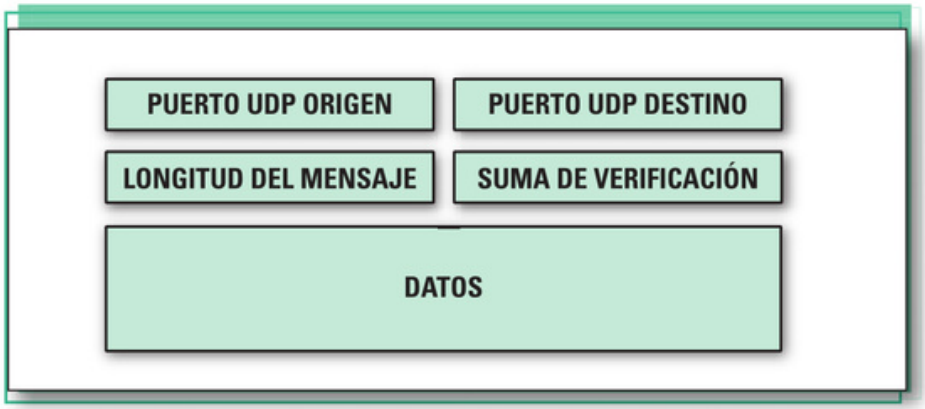
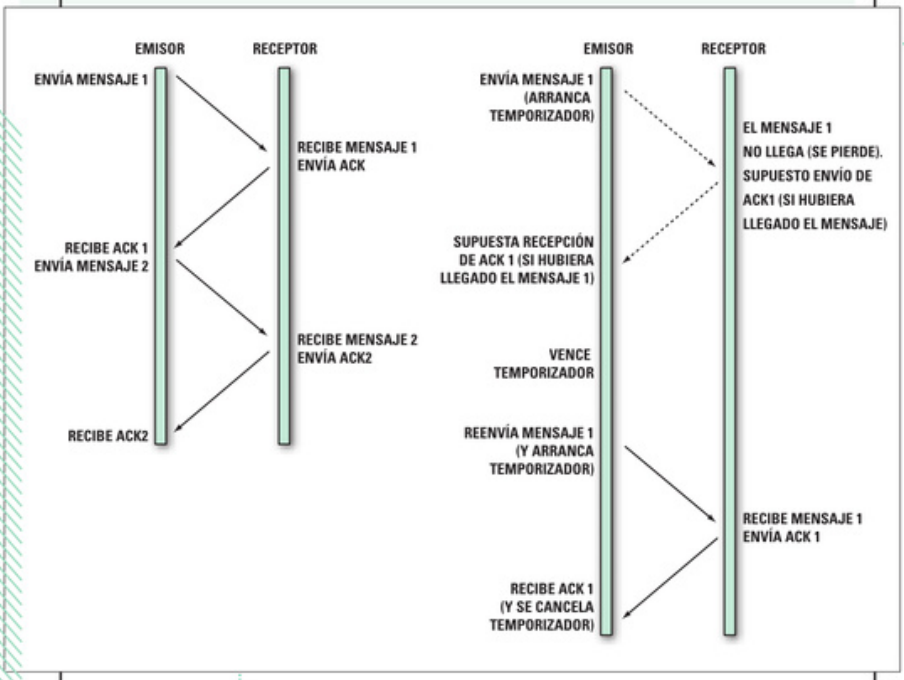
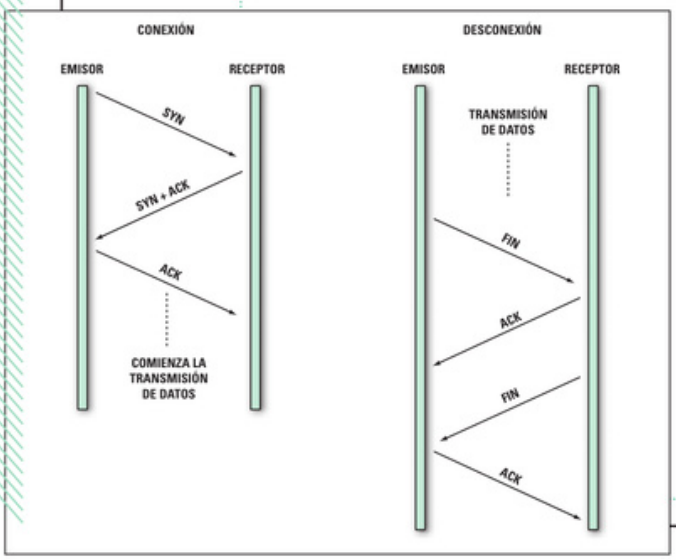


Figura 15. El datagrama UDP está compuesto por el header y los datos. El header consta de cuatro campos: puerto UDP de origen, puerto UDP de destino, longitud del mensaje y suma de verificación.

> Figura 16. En el diagrama de la izquierda, vemos el envío de mensajes sin errores. En el de la derecha, vemos qué ocurre en caso de que el mensaje no llegue correctamente a destino.





> Figura 17. En el esquema de la izquierda, observamos, paso a paso, el proceso para realizar una conexión en TCP (*three way handshake*). A la derecha, el proceso de desconexión (*four way handshake*).

Protocolo TCP

Este protocolo, al igual que el UDP, emplea IP para el transporte de los mensajes, pero a diferencia de aquel, está **orientado a la conexión** y **es fiable.** En el primer caso, porque se necesita establecer una conexión previa entre los dos equipos para la transmisión de los mensajes, con lo cual se garantiza que estos lleguen ordenados al destino; luego de la comunicación, se debe cerrar la conexión. En el segundo caso, debido a que garantiza que los mensajes lleguen correctamente al destino. De esta forma, las aplicaciones que empleen este protocolo no deberán preocuparse por la integridad de los datos, porque de eso se encarga el propio protocolo TCP.

Los protocolos UDP y TCP están ubicados en la capa 4 de transporte del modelo OSI.

Fiabilidad en TCP

Dijimos anteriormente que el protocolo TCP es fiable. Quizá nos preguntemos cómo es posible esto, considerando que TCP viaja sobre el protocolo IP, que es no fiable. Pues bien, la forma de lograr esto es muy simple, y es que el receptor envíe un mensaje de confirmación (**ACK** o **acknowledgement**) al emisor cada vez que reciba un mensaje de él.
En caso de que el emisor no reciba esta confirmación pasado un cierto tiempo, volverá a enviar el mismo mensaje.



Conexión y desconexión en TCP

El proceso para una conexión se realiza en tres pasos. En primer lugar, el emisor manda un mensaje **SYN** (configuración dentro de la cabecera TCP), y si el receptor está en condiciones de establecer la comunicación, le envía un mensaje **SYN + ACK** al emisor. Luego, el emisor manda un mensaje **ACK** al receptor. Recién en ese momento se establece la comunicación. Por otra parte, el proceso para realizar una desconexión se efectúa en cuatro pasos, en donde cada uno deberá enviar un mensaje **FIN** y recibir el **ACK** correspondiente.

CONEXIÓN A SERVIDORES WEB



Veremos cómo se realiza la comunicación entre nuestro navegador web y un servidor web, explicando los conceptos básicos del protocolo HTTP y del lenguaje HTML.

Cada vez que ingresamos en una página web desde nuestro navegador, estamos accediendo a un servidor web; se trata de programas que se ejecutan en una computadora y están permanentemente a la espera de que algún cliente realice una conexión. Estos clientes suelen ser los navegadores web que utilizamos a diario.

El protocolo HTTP

El **protocolo HTTP** está ubicado en la capa 7 del modelo OSI (capa de aplicación). Es un protocolo simple, basado en texto, utilizado por los navegadores y los servidores web para comunicarse.
Cuando desde nuestro navegador intentamos acceder, por ejemplo, al sitio **www.google.com**, estamos realizando una petición HTTP al servidor de Google, que responderá enviando la página solicitada.

El lenguaje HTML

El **lenguaje HTML** es el más empleado para la realización de páginas web. Con él es posible darle formato a un texto: modificar tipo de fuente, color y tamaño; así como también variar su estructura, como párrafos, tablas, etcétera. Para lograrlo, este lenguaje emplea etiquetas, que deben agregarse al texto cuyo formato deseamos cambiar. De este modo, el navegador web las interpreta, y permite mostrar en la pantalla un texto con un formato determinado. Por ejemplo, para que el navegador vea un determinado texto en **negrita**, la página web debe contener ese texto dentro de la etiqueta **** de la siguiente manera: ****texto en negrita****. El navegador web, entonces, mostrará: **texto en negrita**.

Cabecera HTTP

Veamos un ejemplo de una cabecera HTTP correspondiente a la respuesta de un servidor web ante una petición de nuestro navegador (**código 01**). ■

> **Desde Internet Explorer, podemos ver el código fuente (HTML) de una página web haciendo clic en Página/Ver código fuente. En este ejemplo, vemos el de Nic Argentina (www.nic.ar).**



> **Código 01**

```
HTTP/1.1 200 OK
Date: Fri, 02 Apr 2010 02:55:57 GMT
Server: Apache
Last-Modified: Tue, 28 Apr 2009 16:03:52 GMT
Accept-Ranges: bytes
Content-Length: 1386
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

> **Código 01. La respuesta HTTP/1.1 200 OK indica que la petición fue correcta.**

PASO A PASO

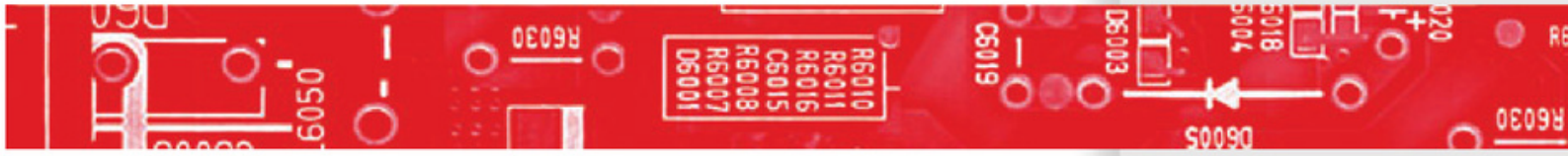
CONEXIÓN A UN WEB SERVER VÍA TELNET

Explicaremos cómo conectarnos a un servidor web mediante la aplicación Telnet, para obtener el código HTML que interpreta nuestro navegador.

Telnet es un protocolo de la capa 7 del modelo OSI (capa de aplicación), que permite conectar un cliente con un servidor, y enviarle comandos a este último a través del teclado. Utiliza una conexión TCP para mandar información. A continuación, realizaremos una conexión con el servidor web de Nic Argentina (www.nic.ar). ■

01



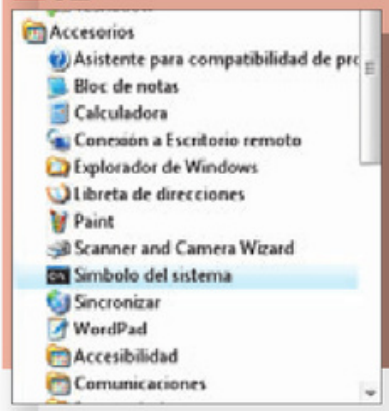


01. Debemos verificar que esté habilitada la aplicación Telnet. Para hacerlo, vamos a **Inicio/Panel de control** y hacemos clic en **Desinstalar un programa**. Posteriormente, presionamos **Activar o desactivar las características de Windows**. Aparecerá una ventana en la cual debemos asegurarnos de que estén marcadas las opciones **Cliente Telnet** y **Servidor Telnet**.
Por último, pulsamos **Aceptar**.

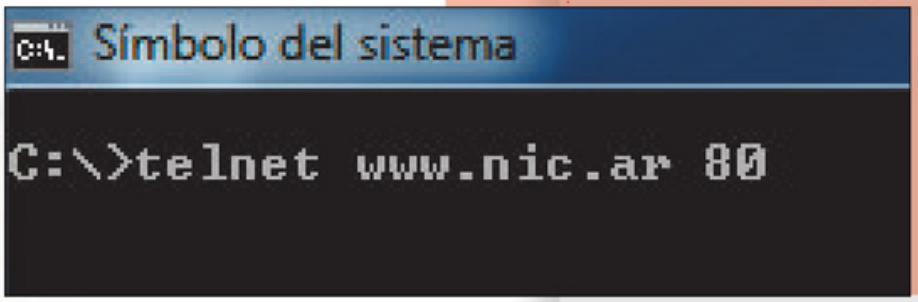
02. A continuación, vamos a abrir una terminal, desde **Inicio/Todos los programas/Accesorios/ Símbolo del sistema**.
Para ver mejor el resultado del Telnet, es conveniente maximizar la ventana.

03. Estamos en condiciones de realizar la conexión al servidor web mediante Telnet.
Para esto, escribimos lo siguiente: telnet www.nic.ar 80 y presionamos **<Enter>**. Lo que estamos haciendo es abriendo una conexión TCP con el puerto 80 (HTTP) del servidor web de Nic Argentina.

02

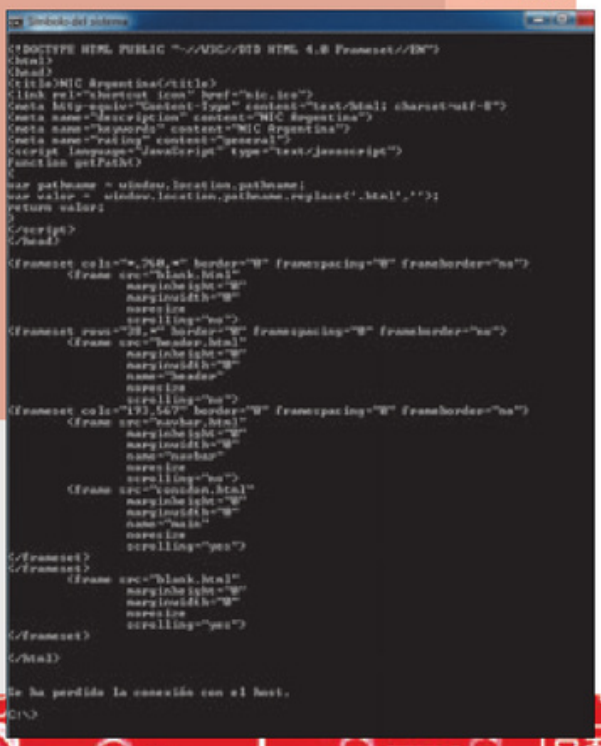


03



http://

05

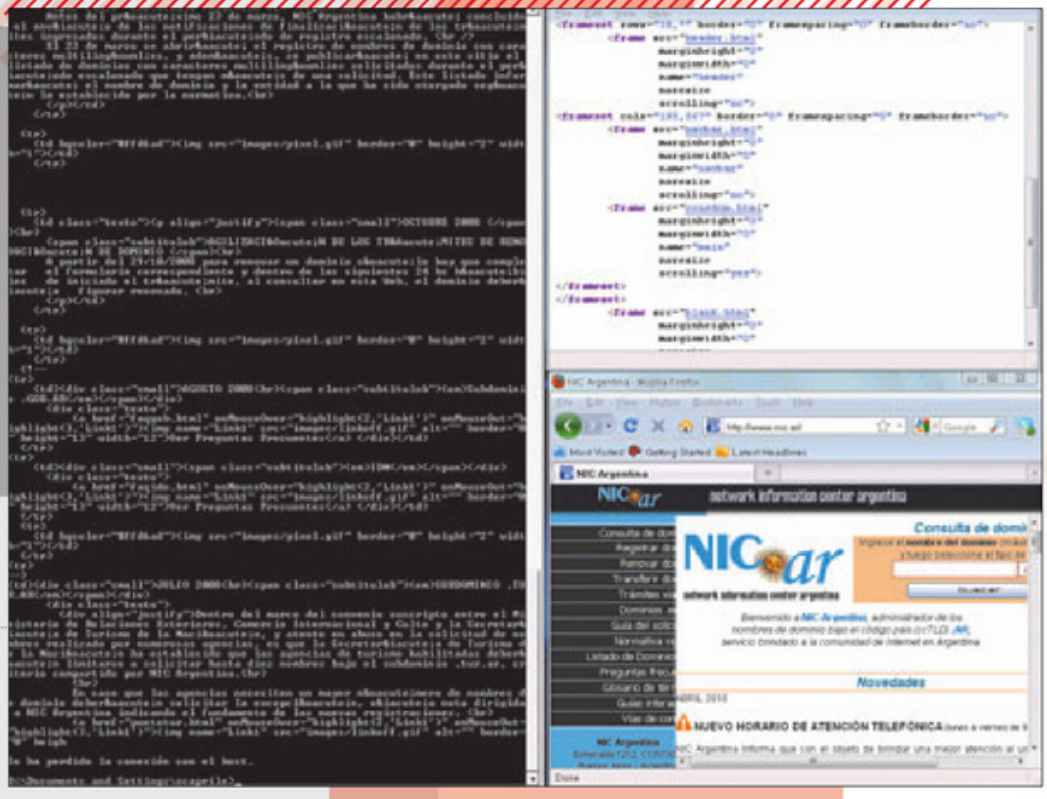


04. Quedará la terminal vacía, sin ningún texto. Es el momento de darle la instrucción al servidor web para que nos envíe la página correspondiente. Entonces, escribimos: GET / y presionamos la tecla **<Enter>**. Debemos tener cuidado al hacerlo, ya que probablemente no aparezca en pantalla el texto ingresado. Esto se debe a que, por defecto, el eco local está desactivado.

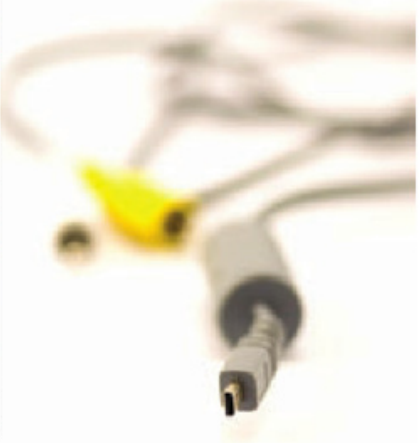
05. Finalmente, veremos en pantalla el código HTML crudo que entrega el servidor de Nic Argentina. Los navegadores se encargan de interpretarlo y mostrarlo en pantalla con el formato adecuado. Por ejemplo, vemos que hay varios framesets que llaman a otros archivos HTML.

Telnet permite conectar un cliente con un servidor, y enviarle comandos desde el teclado.

06



06. Volvemos a conectarnos y, esta vez, pedimos, por ejemplo, uno de los framesets: GET /consdom.htm y pulsamos **<Enter>**. Obtenemos más código HTML, de donde extraeremos información sobre imágenes y texto.



EL LENGUAJE HTML

HTML es un lenguaje que nos permite incluir marcas en el texto para darle formato y embeber otros objetos, como imágenes y pequeños programas en JavaScript.

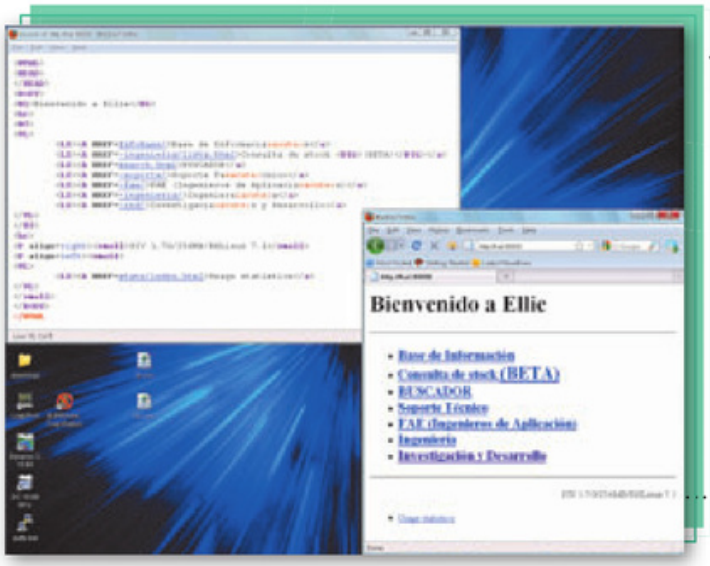


En la actividad realizada nteriormente tuvimos la oportunidad de comportarnos como un navegador web; es decir, nos conectamos a un servidor **HTTP** iniciando una conexión **TCP al puerto 80** y le pedimos una página web.
Procedimos a analizarla sintácticamente y descubrimos que no podíamos mostrar nada porque necesitábamos otros objetos. Finalmente, comenzamos un proceso iterativo y recursivo para traer uno por uno los objetos, analizarlos y seguir así hasta obtener todos.
Pudimos observar que los objetos se indican dentro de **tags**, encerrados de esta forma: **< tag >**. El código HTML está encerrado entre los tags **<HTML>** y **</HTML>**. Algunos de los objetos que podemos encontrar son:

- Encabezado de página: entre **<HEAD>** y **</HEAD>**; contiene, entre otras cosas, el título de la página, encerrado entre los tags **<TITLE>** y **</TITLE>**
- Cuerpo de la página: entre **<BODY>** y **</BODY>**.

Por otro lado, en el cuerpo de la página podemos hallar:

- framesets: entre los tags **<FRAMESET>** y **</FRAMESET>**; contienen otras páginas HTML incrustadas en porciones de la página actual.



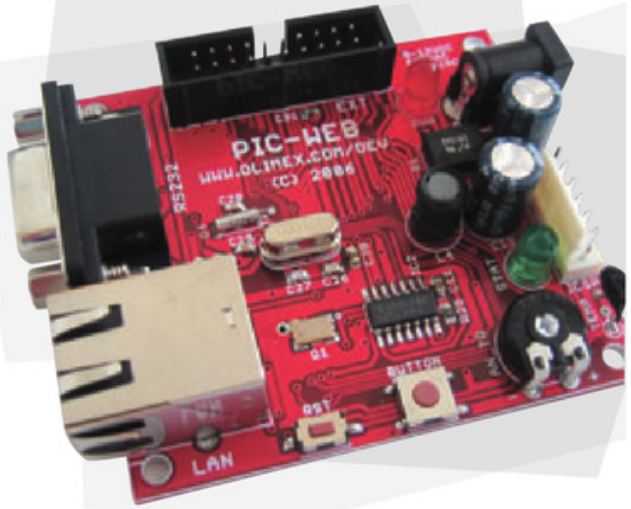
> Una página HTML muy simple, con listas y links, para servir de acceso en una intranet (red interna de una empresa).

> ESPECIFICACIONES HTML

La especificación corriente de HTML puede obtenerse de la página de estándares de W3C: **www.w3.org/TR**.
Mientras se trabaja en la versión 5, el estándar vigente es HTML 4.01. Existen muchos más objetos, como botones, selectores y listas. Poco a poco, iremos descubriendo algunos de ellos y veremos la manera de utilizarlos en nuestros proyectos con microcontroladores.

- imágenes, identificadas por el tag **<IMG SRC = "nombre.ext>**
- encabezados: identificados entre los tags **<Hx> y </Hx>**, donde la x es un número del 1 al 6 que corresponde al orden de encabezado, según su importancia y tamaño de fuente
- tablas: se definen entre los tags **<TABLE>** y **</TABLE>**. En el cuerpo de la tabla, cada fila se define entre **<TR>** y **</TR>**, y cada columna, entre **<TD>** y **</TD>**.
- párrafos: entre los tags **<P>** y **</P>**.
- separadores horizontales: indicados como **<HR>**.
- saltos de línea: indicados como **
**.
- links a otras páginas u objetos. El link se indica dentro del tag **<A HREF = "url">**, mientras que el texto comprendido entre este tag y **</A>** se muestra como un link.

ETHERNET CON PIC



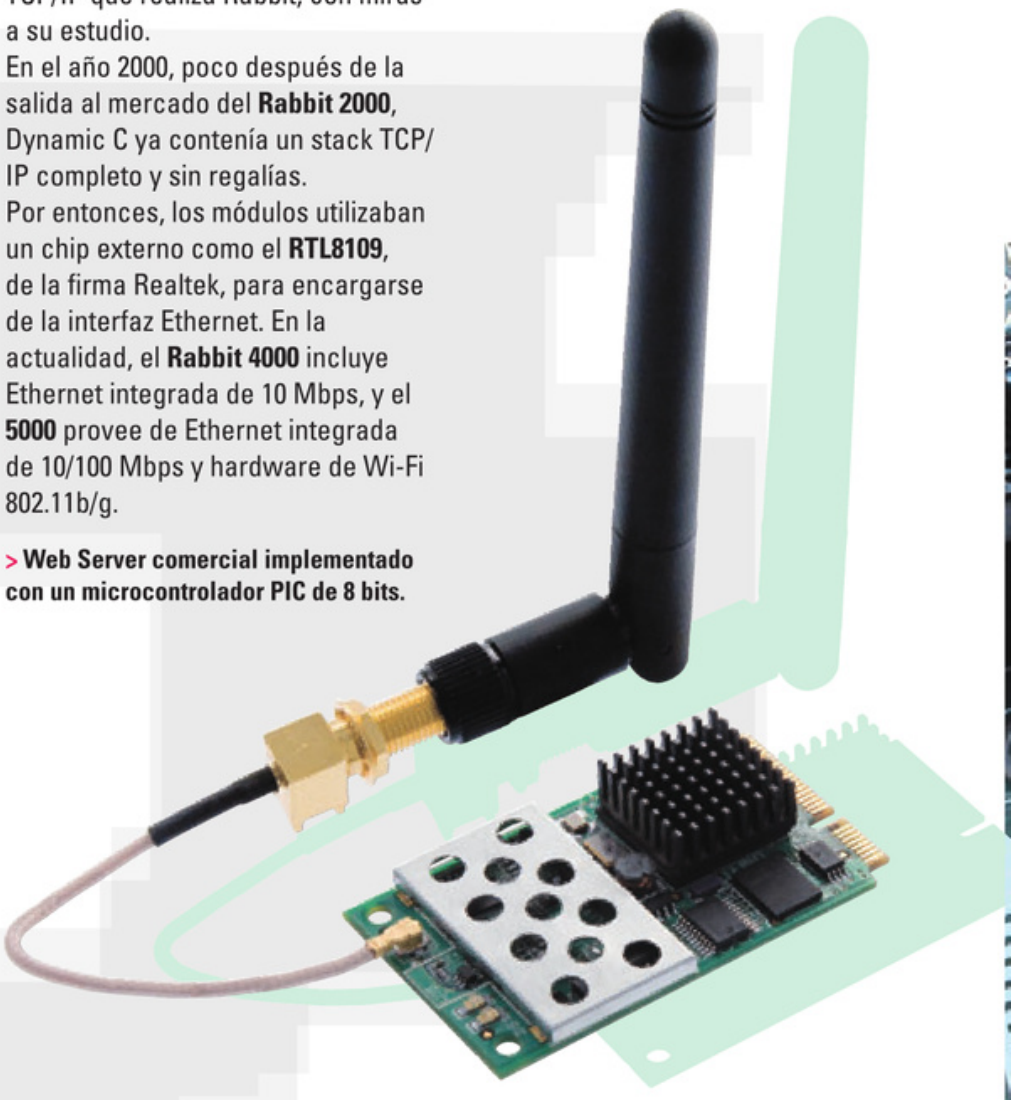
Microchip también ofrece varias alternativas para realizar una conexión Ethernet embebida. Debemos destacar su bajo costo y sencillez de implementación.

Los microcontroladores PIC diseñados para Ethernet incorporan un módulo dedicado para esta función. En la mayoría de los casos, estos micros poseen gran cantidad de pines (64, 80 o 100) y son de montaje superficial. Sin embargo, podemos migrar una aplicación que originalmente no tenía conectividad Ethernet a través de módulos de Microchip que se conectan por SPI al microcontrolador.
Por ejemplo, podríamos conectar un módulo Ethernet al PIC18LF4620 que utilizamos en la primera parte de la colección y, de este modo, tener el sistema conectado a una red.
Para realizar una conexión Ethernet con PIC es posible recurrir al Microchip TCP/IP Stack, que se descarga de forma gratuita desde el sitio oficial del fabricante: **www.microchip.com/tcpip**.
A diferencia de la capa física, este conjunto de protocolos establece la conexión lógica entre el otro host y el microcontrolador. Además, incluye varios ejemplos, lo que permite optimizar el tiempo de desarrollo. Una ventaja de estos micros es su reducido tamaño y el hecho de que, al incorporar el módulo Ethernet, los costos se reducen considerablemente. Además, veremos de qué forma añadir conectividad Wi-Fi, y realizar telemetría y control a grandes distancias por medio de Internet.

Los microcontroladores PIC con módulo Ethernet se destacan por su robustez y facilidad de uso. Además, las librerías gratuitas de Microchip permiten una rápida implementación.

TCP/IP en Rabbit

Vamos a explicar brevemente la implementación del modelo TCP/IP que realiza Rabbit, con miras a su estudio.
En el año 2000, poco después de la salida al mercado del **Rabbit 2000**, Dynamic C ya contenía un stack TCP/IP completo y sin regalías.
Por entonces, los módulos utilizaban un chip externo como el **RTL8109**, de la firma Realtek, para encargarse de la interfaz Ethernet. En la actualidad, el **Rabbit 4000** incluye Ethernet integrada de 10 Mbps, y el **5000** provee de Ethernet integrada de 10/100 Mbps y hardware de Wi-Fi 802.11b/g.



> **Web Server comercial implementado con un microcontrolador PIC de 8 bits.**

RabbitWeb es un lenguaje script para armar páginas web dinámicas.

La interfaz de programación

La interfaz que provee **Dynamic C** guarda una cierta relación con la de **Berkeley sockets**, lo suficiente como para que el programador no deba aprender desde cero un nuevo entorno. Si bien muchas llamadas son diferentes, la arquitectura es muy similar y se domina con rapidez. Todo el stack corre cuando el usuario cede el control mediante una llamada a la función **tcp_tick()**.

La ocupación de memoria se administra al momento de compilar mediante macros que definen, por ejemplo, la cantidad de sockets y servidores.

> EL STACK TCP/IP

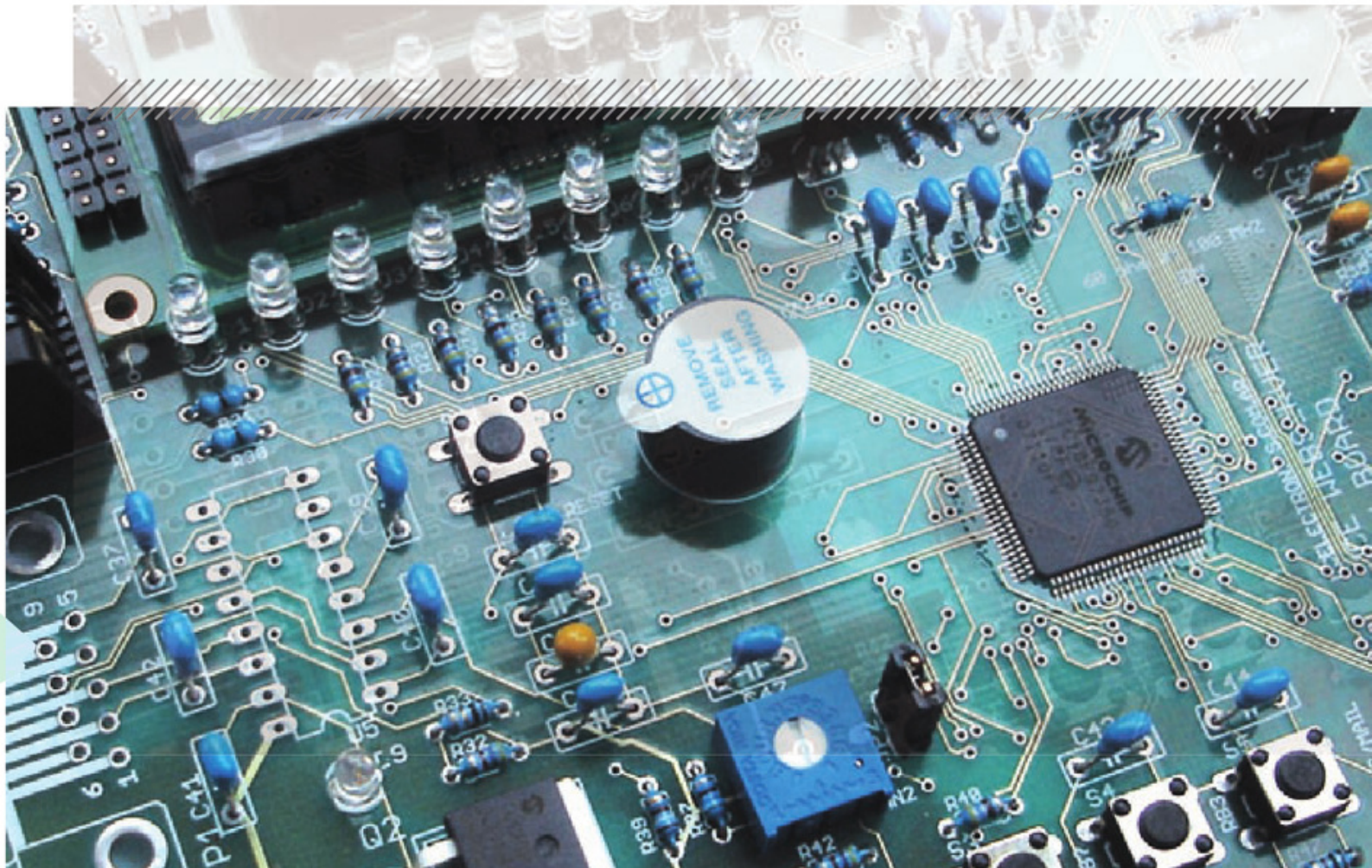
Un stack completo requiere mucha RAM, particularmente, para **packet buffers**, recuperar datagramas IP fragmentados y ensamblar el máximo de un segmento TCP. Dada su capacidad de memoria, los módulos Rabbit siempre han incorporado lo suficiente como para que quepa el **stack completo** y el usuario disponga de memoria para su aplicación.

Los servicios

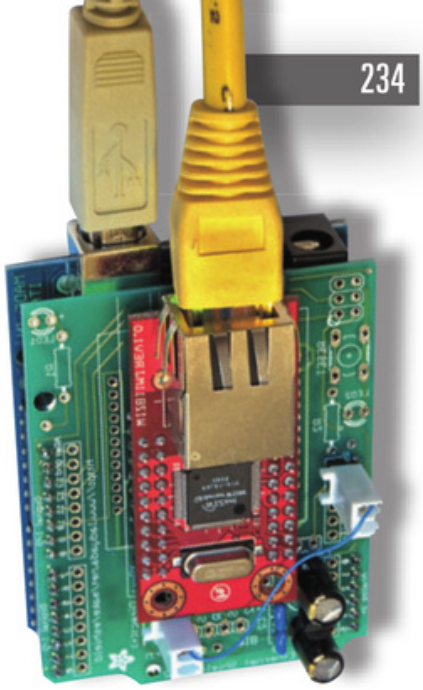
Las bibliotecas de funciones de Dynamic C incorporan código fuente para muchas aplicaciones, veamos cuáles son:

- **Servidor HTTP con SSI** (*Server Side Includes*) y un lenguaje script con sintaxis similar a C, llamado **RabbitWeb**.
- **Cliente POP y SMTP** para recibir y enviar correo, respectivamente.
- **Cliente DHCP**.
- **Cliente y servidor FTP** para transferir archivos.
- Resolución de nombres (**DNS**).
- Soporte para **DDNS** (DNS dinámico, mediante un **parser HTML**).
- **SSL** y **AES** para conexiones seguras entre dispositivos.

Estos son solo algunos de los **códigos fuente** que incorporan las bibliotecas de Dynamic C.



OTRAS ALTERNATIVAS



En estas páginas analizaremos las opciones de networking disponibles para otros fabricantes de microcontroladores. Veamos qué hay en el mercado.

En las páginas anteriores analizamos la propuesta de **Microchip** y la de **Rabbit**, pero estos no son los únicos fabricantes de microcontroladores. Afortunadamente, la oferta es muy amplia y hay productos para todos los gustos. Si bien algunos desarrolladores lideran en ventas, otros lo hacen en innovación, y hay quienes marcan tendencias en los inicios, por lo que todos tienen algo de lo que podemos aprender.

Freescale

El coloso heredero de Motorola, por el momento, no parece interesarse en networking para 8 o 16 bits. Sus soluciones están orientadas a su línea de **32 bits Coldfire**, mediante un stack que provee sin cargo y que fue desarrollado por la firma **InterNiche**.

NetBurner es una línea de módulos con Ethernet basados en micros Coldfire de Freescale.

Texas Instruments

Este gran innovador tiene el foco de networking en sus líneas de 32 bits, entre ellas, **OMAP**, basada en núcleos **ARM** de alta gama; y **Stellaris**, la línea **ARM Cortex-M3**. Sin embargo, provee una nota de aplicación que plasma un hardware y stack de código abierto basado en el controlador **CS8900A** de Crystal Semiconductor, para ingresar en el networking de la mano del **MSP430**. La firma Rowley Associates, fabricante de un excelente entorno de desarrollo y compilador **C/C++**, brinda, además, el hardware y un stack basado en **uIP** (stack simplificado para microcontroladores).

Atmel AVR

Este micro, caracterizado por su fuerte impacto en los aficionados –que terminan proveyendo gran parte del soporte adicional–, no posee controlador Ethernet. Existe en la Web una gran cantidad de proyectos, muchos basados en el controlador **RTL8019** de Realtek, en los que el stack TCP/IP está basado otra vez en **uIP**. El foco de Atmel en networking está puesto en la línea **AVR32**, una versión extendida del core AVR a 32 bits, y sus micros con núcleo **ARM**.



> ALTERNATIVAS

Existen algunos fabricantes que han desarrollado alternativas basadas en clones del 8051; entre ellos, el **TINI**, de Dallas Semiconductor (hoy **Maxim**). Por su parte, **Silicon Labs** ofrece cores 8051 compatibles de hasta 100 MIPS, y kits para agregar networking.

WIZnet

Hay una opción adicional para quienes no quieren abandonar su micro, y es utilizar módulos de networking. Hemos visto lo que todos tienen para ofrecer y, seguramente, tenemos el nuestro preferido. Si ninguna de las opciones anteriores nos satisface, todavía podemos ingresar en el mundo del networking, dado que el fabricante coreano **WIZnet** ofrece una serie de chips y módulos prearmados que no solo se encargan de la interfaz Ethernet, sino que, además, nos permiten delegarles el **stack TCP/IP**.

Los chips

El servicio provisto por los chips consta de una interfaz **socket level** a TCP o UDP para algunos sockets (**4** en el **W3150A+** y **8** en el **W5300**). No existen servicios como Web Server o cliente DNS, solo comunicación por un socket. El fabricante ofrece notas de aplicación y **código C** para, por ejemplo, ATmega128, con el objetivo de implementar algunas de estas funcionalidades. La implementación de **ICMP** es bastante reducida, y los chips no parecen soportar fragmentación **IP**.

> MÓDULOS

Algunos de los módulos disponibles para conexión inmediata son el **WIZ811MJ**, basado en el chip **W5100**; y el **WIZ830MJ**, basado en el chip **W5300**. El primero provee de una interfaz **SPI**, mientras que el segundo ofrece un bus de 8 o 16 bits. Ambos contienen el conector RJ-45 Ethernet y una tira de pines para conexión a un proyecto determinado.

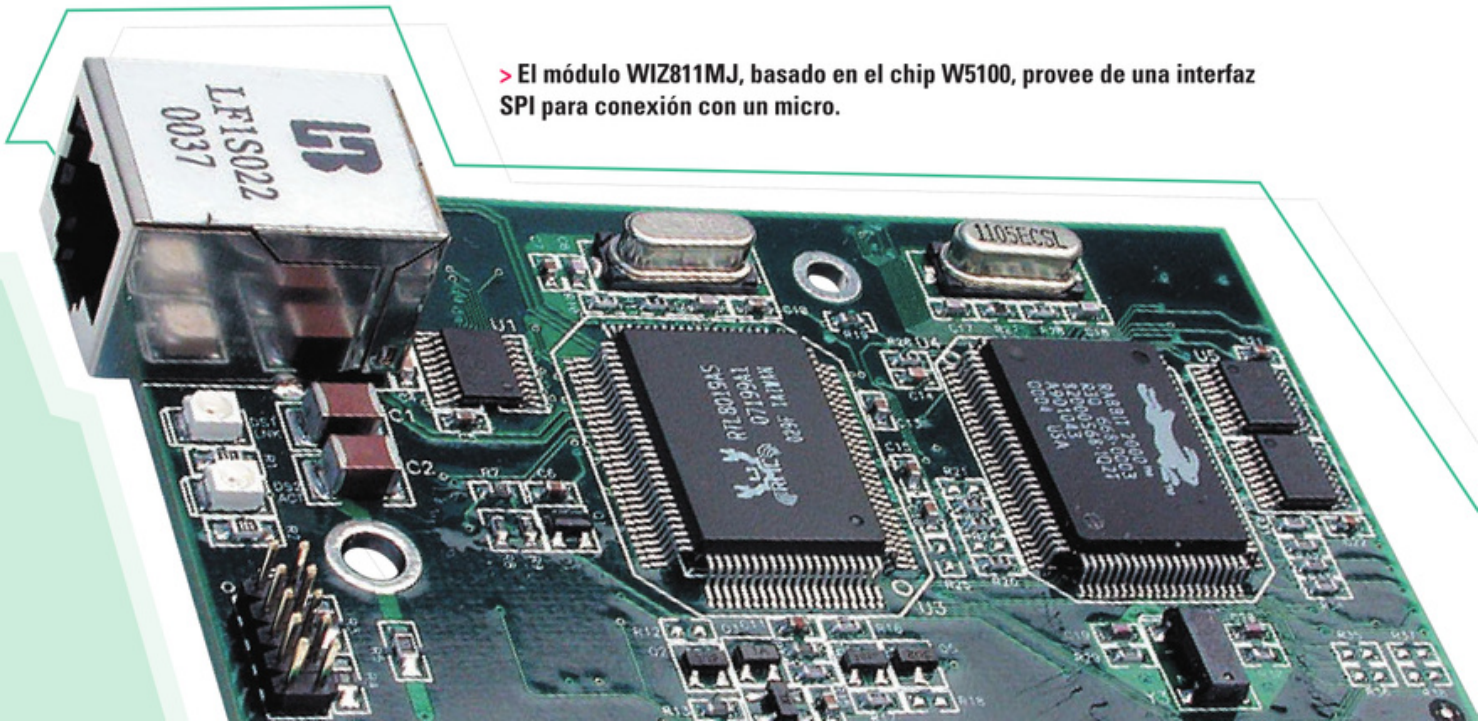
El chip W5300 soporta Ethernet 10/100, hasta ocho sockets y PPoE para conectarse a Internet.

La forma de configurar los chips es lo que esperaría un developer acostumbrado a **SPI** y periféricos, pero sin demasiados conocimientos sobre networking. Esto resulta más atrayente para quien piensa en su conexión TCP/IP como una forma de reemplazar un cable, en oposición a programar una aplicación utilizando una interfaz similar a **Berkeley sockets** como **Rabbit**.

Los módulos

Existen, fundamentalmente, dos tipos de módulos: unos proveen una interfaz **RS-232**, y los otros, acceso al chip:

- **Módulos con acceso al chip:** presentan un conector **RJ-45** y tiras de pines para su conexión a la PCB de control. No existe I/O, solo una interfaz para ser controlada por un micro. El módulo por sí mismo solo sirve como interfaz de networking para un microprocesador.
- **Módulos con interfaz RS-232:** tienen un conector **DB-9** y otro para la fuente de alimentación, además del **RJ-45**. Incorporan un micro para brindar cierta funcionalidad de configuración de servicio. ■



> El módulo WIZ811MJ, basado en el chip W5100, provee de una interfaz SPI para conexión con un micro.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué son las redes informáticas?

2. Caracterice una red LAN, MAN y WAN.

3. Describa las diferentes topologías de red existentes.

4. ¿Qué es el modelo OSI?

5. ¿Qué es TCP/IP?

6. ¿Cuáles son las capas del modelo TCP/IP?

7. ¿Qué función cumple un servidor DNS?

8. ¿Qué es el protocolo UDP?

9. ¿Cuáles son las características que poseen los microcontroladores PIC?

10. ¿Para qué sirve NetBurner?

EJERCICIOS PRÁCTICOS

1. Genere un esquema de una red LAN.

2. Realice la conexión a un servidor web explicando cada paso del proceso.

3. Verifique que Telnet se encuentre habilitado en el sistema.

4. Utilice Telnet para conectarse a un servidor web.

5. Acceda al código HTML de una página web e identifique cada uno de sus bloques.

CAPÍTULO 10

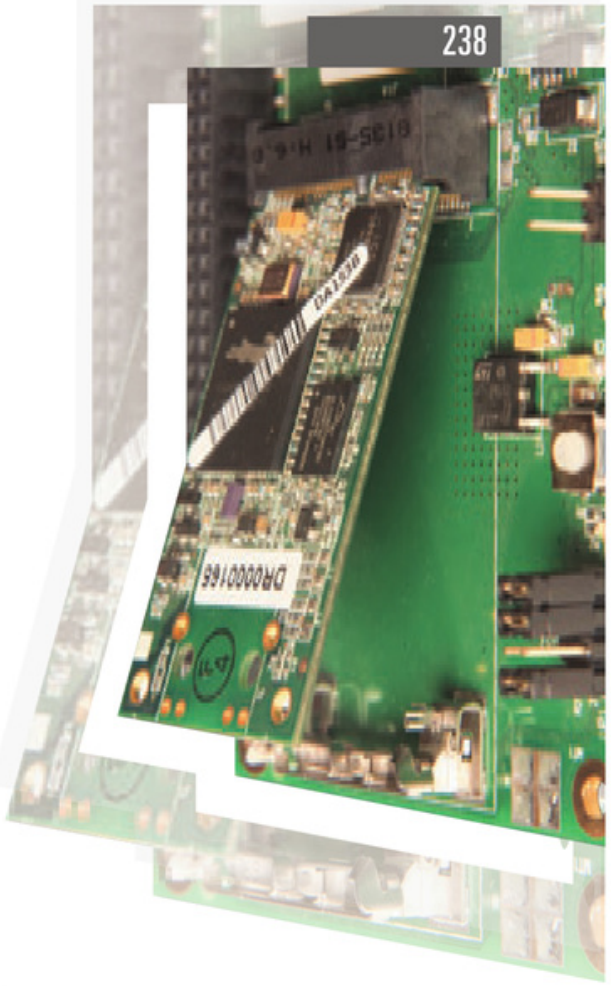
NETWORKING CON RABBIT

Las funciones de networking en Rabbit
Servidores web SSI y CGI
RabbitWeb
Servidor web con Rabbit
Conceptos integrados
WI-FI con Rabbit
Multitarea cooperativo
Multitarea cooperativo en Rabbit
Actividades

10

LAS FUNCIONES DE NETWORKING EN RABBIT

Rabbit provee el código fuente de todo el stack TCP/IP, con recursos como servidores HTTP y FTP, clientes para enviar y recibir correo, y acceso a sockets.



Todo el proceso de inicialización y operación del hardware de networking se maneja a través de una serie de libraries (librerías), de modo que el usuario no deba interactuar con el hardware. La principal ventaja de este conjunto es **DCRTCP.LIB**, que contiene todas las funciones esenciales del **stack TCP/IP.** Incluiremos esta library mediante una directiva, pero antes de hacerlo, deberemos definir nuestra dirección IP, máscara, router y DNS. Por supuesto que es posible recibir toda la configuración mediante **DHCP** (*Dynamic Host Configuration Protocol*), pero para mantener las cosas simples y bajo control, iremos paso por paso.
Un ejemplo se ve en el **Código 01**.

Inicialización de la interfaz

Para inicializar la interfaz –en este caso, Ethernet–, debemos realizar un llamado a la función **sock_init()**.
Esta vuelve inmediatamente y, como asignamos una dirección fija, ya podemos comenzar a trabajar. En el caso de DHCP o de Wi-Fi, como veremos más adelante, es necesario esperar a que termine el proceso de inicialización.

> Todo el stack corre cuando el usuario cede el control mediante una llamada a la función tcp_tick().

Manejo del stack

Como adelantamos el capítulo anterior, todo el stack corre cuando el usuario cede el control mediante una llamada a la función **tcp_tick()**. Esta función deberá ser llamada periódicamente para poder atender al buen funcionamiento del stack TCP/IP. Para todas las otras tareas que vayamos a realizar, disponemos de funciones específicas.
Por ejemplo, existe una que nos permite demorar la aceptación o rechazo de una conexión TCP a la espera de sockets libres, **tcp_reserveport()**, que se emplea para evitar que las conexiones sean rechazadas en el tiempo entre que se cierra una y se está abriendo otra.

> Código 01

```
#define TCPCONFIG        0
#define USE_ETHERNET     1

#define MY_IP_ADDRESS  "192.168.1.54"
#define MY_NETMASK     "255.255.255.0"

#define MY_GATEWAY     "192.168.1.1"
#define MY_NAMESERVER          "200.42.0.108"

#use "dcrtcp.lib"
```

Servicios

En el capítulo anterior, vimos que **Dynamic C** provee de código fuente para muchos servicios, como los siguientes:

- **Servidor HTTP con SSI** (*Server Side Includes*) y un lenguaje de script con sintaxis similar a C llamado **RabbitWeb**.
- **Cliente POP y SMTP** para recibir y enviar correo, respectivamente.
- **Cliente DCP** (*Dynamic Host Configuration Protocol*).
- **Cliente y servidor FTP** para transferir archivos.
- Resolución de nombres (**DNS**).
- Soporte para **DDNS** (DNS dinámico, mediante un parser **HTML**).
- **SSL** y **AES** para conexiones seguras.

Sockets

Para cualquier otra actividad que no sea resuelta por estos servicios, o para UDP, se brinda acceso directo a los sockets:

SSL y AES son operaciones que requieren gran poder de procesamiento.

- Podemos abrir sockets TCP llamando a la función **tcp_open()** y sockets UDP mediante **udp_open()**.
- Podemos crear servidores que esperen una conexión TCP en determinado port llamando a **tcp_listen()**.
- Establecido el socket TCP, enviamos y recibimos información mediante **sock_read()** y **sock_write()**.
- En UDP, utilizamos **udp_send()** y **udp_recv()**.

El entorno, en general, es similar al que encuentra alguien familiarizado con programar aplicaciones de networking (Berkeley sockets). La cantidad de conexiones y el tamaño de los buffers se limitan operando sobre algunas macros, previamente a la inclusión de **DCRTCP**. Por ejemplo, para setear el número máximo de sockets a seis hacemos lo que muestra el **Código 02**.

Resolución por DNS

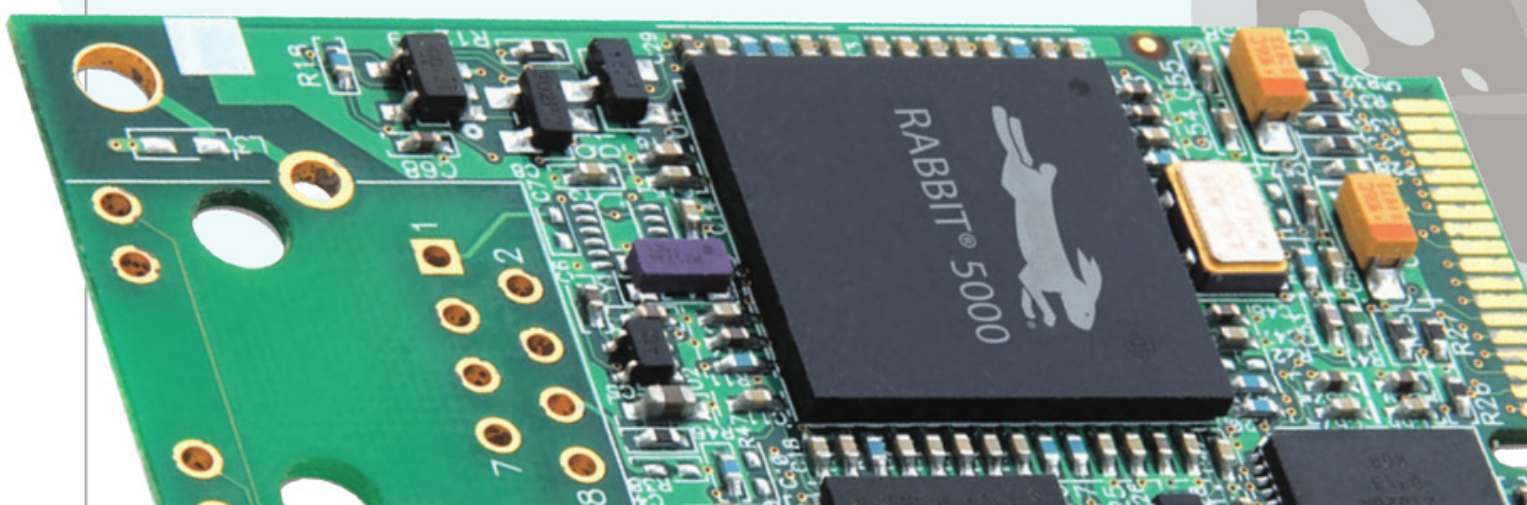
Si queremos iniciar una comunicación y solo disponemos del nombre de host, deberemos **resolver** este nombre a una dirección IP. El proceso de resolución es una búsqueda iterativa, como vimos en el capítulo anterior, pero en este caso todo lo que debemos hacer es utilizar la función **resolve()**. Esta devuelve una dirección IP en un formato que puede ser utilizado por las demás funciones del stack TCP/IP. Un ejemplo de esto se ve en el **Código 03**.

> **Código 02**

```
#define MAX_TCP_SOCKET_BUFFERS 6
```

> **Código 03**

```
resolve(" HYPERLINK ""www.ldir.com.ar");
```



> El minicore RCM5700 no posee conector RJ-45, pero provee Ethernet de 10/100 Mbps en sus pines.

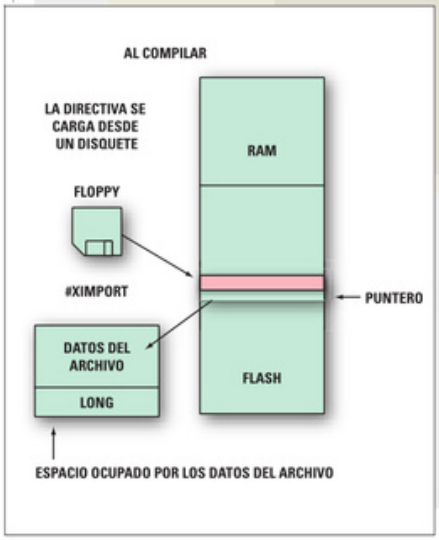
Carga de archivos en memoria

Para algunos de los servicios, como servidor FTP o web (HTTP), necesitaremos referirnos a archivos. Como estamos programando en un micro sin un sistema operativo que nos respalde, el concepto de file system o sistema de archivos no existe. Si bien podemos implementarlo, hay maneras más simples y productivas de trabajar. Lo que los servicios en realidad necesitan es una referencia a un bloque de memoria donde esté determinado un contenido específico. Después, asociaremos este contenido con un nombre mediante un directorio.
Así, al momento de compilar, Dynamic C toma un archivo del disco de la PC, lo aloja en un espacio de la memoria flash de Rabbit, y genera un puntero a esa posición. Esto se realiza mediante una directiva (**Código 04**).

El primer elemento es el nombre del archivo en la PC; y el segundo, el del puntero. Así, **index_html** apunta al contenido de **index.html**, copiado a la memoria flash de la PC al momento de compilar.

Servidor HTTP

El servidor HTTP es el encargado de servir los elementos de una página web. Como tal, requerirá de un directorio donde tendrá referencias para traducir los nombres de los archivos (URLs) a su posición en la flash. De modo similar, precisará una tabla para indicar al navegador el tipo de archivo (tipo MIME) que va a servir, cuando el navegador lo interrogue; y, en algunos casos, un tercer elemento que le señale al servidor mismo cómo procesar ese archivo (qué **handler** utilizar). Esto se resuelve mediante dos estructuras, en las cuales se cargan los valores necesarios. Para facilitar la operatoria, Dynamic C incluye una serie de macros que simplifican el proceso en cuestión. Por ejemplo, para definir la tabla de tipos **MIME**, procedemos del modo que muestra el **Código 05**. Vemos que **SSPEC_MIME_FUNC** permite definir el **handler** por utilizar para este tipo de archivos, mientras que **SSPEC_MIME** es una entrada normal de la lista de tipos MIME.



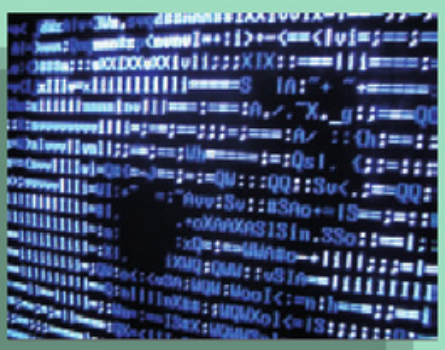
La directiva #ximport copia el contenido del archivo en la PC a la flash de Rabbit.

> **Código 04**

```
#ximport "index.html"   index_html
```

> **Código 05**

```
SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".shtml", "text/html", shtml_handler),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
        SSPEC_MIME(".cgi", "")
SSPEC_MIMETABLE_END
```



> SERVIDOR HTTP

Un servidor HTTP (*HyperText Transfer Protocol)* es una tarea que abre un socket TCP en el puerto 80 y espera una conexión. Cuando la recibe, la acepta y aguarda un tiempo por un pedido. Si no recibe nada, cierra la conexión; y si recibe un pedido, lo cumplimenta y luego la cierra.
En ambos casos, queda en espera de una nueva conexión. Como vimos en el capítulo anterior, los pedidos son texto ASCII que indican un archivo por servir.

Para definir el directorio de un servidor HTTP, realizamos lo que se muestra en el **Código 06**.

Aquí, **SSPEC_RESOURCE_XMEMFILE** es un "archivo" en xmem, **SSPEC_RESOURCE _ROOTVAR** identifica a una variable, y **SSPEC_RESOURCE_FUNCTION** identifica a una función **SSI** o **CGI**. Ambas se aplican en un contexto que veremos más adelante.

También es posible agregar archivos en tiempo de ejecución y servirlos, por ejemplo, desde una tarjeta mini-SD. La cantidad de servidores HTTP dispuestos a atender pedidos es dos; podemos modificar este dato agregando una macro antes de incluir la library de HTTP. Por ejemplo, para cuatro servidores veamos el **Código 07**.

El servidor se inicializa llamando a la función **http_init()**. El manejo del servidor lo lleva adelante la función **http_handler()**, que deberemos llamar periódicamente. Esta, a su vez, llama a **tcp_tick()**, manteniendo activo el stack. ■

> **Código 06**

```
SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/", index_html),
        SSPEC_RESOURCE_XMEMFILE("/muestra.shtml", muestra_html),

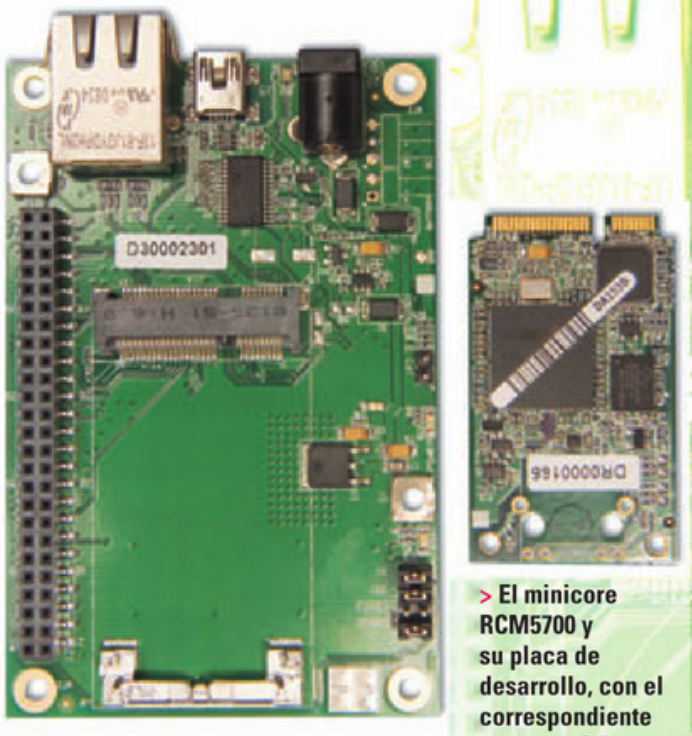
        SSPEC_RESOURCE_ROOTVAR("mivar",&mivar, INT16,"%d"),

        SSPEC_RESOURCE_FUNCTION("/mifun.cgi", setup)
SSPEC_RESOURCETABLE_END
```

> **Código 07**

```
#define HTTP_MAXSERVERS 4
#use "http.lib"
```

El directorio del servidor y los tipos MIME se arman como estructuras.



> **El minicore RCM5700 y su placa de desarrollo, con el correspondiente conector RJ-45.**

SERVIDORES WEB SSI Y CGI

Los servidores web son capaces de mostrar variables y ejecutar funciones al momento de servir una página. Veremos cómo hacerlo.

SSI (*Server Side Includes*) y **CGI** (*Common Gateway Interface*) son dos formas estándares de generar páginas dinámicas. La primera permite que el servidor identifique **tags** en el **SHMTL** (*HTML con SSI*) y los reemplace por el valor de una variable o la ejecución de una función. La segunda asigna una **URL** (*Uniform Resource Locator*, dirección de una página web) para que se ejecute una función a pedido. Esto es soportado por todo servidor web que se precie de tal; solo debemos utilizar la extensión **shtml** para el archivo y, en Rabbit, indicar el **handler** como vimos en la **página 7**. Los **tags** se introducen como comentarios HTML; por ejemplo, para mostrar una variable y ejecutar una función.

La forma en que el servidor web vincula los nombres dentro del **SHTML** con el resto del sistema depende de cada implementación en particular. Para la ejecución de funciones **CGI**, un servidor muy utilizado, **Apache**, permite definir un directorio en el cual se alojarán los archivos que podrán ser ejecutados. Estos, a su vez, pueden ser cualquier tipo de ejecutable que maneje el sistema donde reside el servidor. Este directorio es, generalmente, **/**

> AUTENTICACIÓN

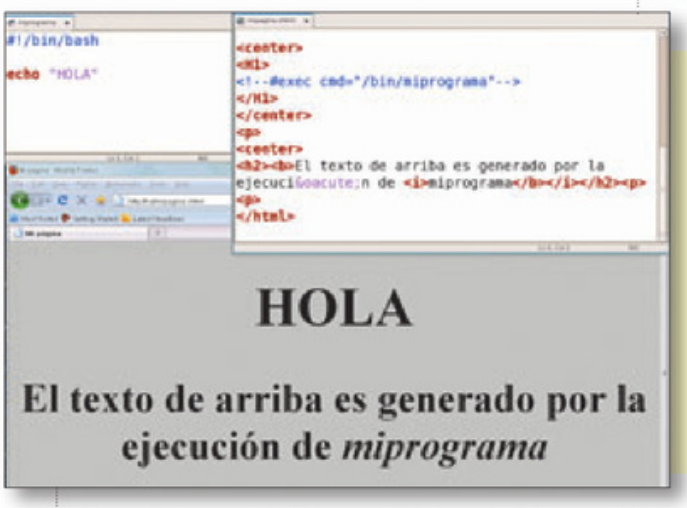
El proceso de autenticación consiste en solicitar un nombre de usuario y una clave. Uno de los métodos para hacerlo, **basic**, se basa en el envío en limpio de la información pedida. Otro, más avanzado y seguro, se denomina **digest**, e intercambia información generada a partir de la que se quiere autenticar, sin que esta se envíe realmente por la red. Esto evita que sea leída en tránsito.

Un script CGI es un programa que se ejecuta en respuesta a un pedido de página.

> Código 08

```
<!--#echo var="mivar"-->

<!--#exec cmd="mifun"-->
```



> Ejecución de un programa mediante SSI en Apache. A la izquierda, el programa; a la derecha, el SHTML.

En Rabbit

En este entorno, los nombres de la **función** y la **variable** deben existir en el directorio del servidor y apuntar al objeto respectivo.

Autenticación

Cuando queremos proteger el acceso a una página o limitar la ejecución de una función, necesitamos identificar a quién intenta acceder, para establecer de algún modo cómo se relaciona esto con los archivos.
Una forma utilizada en el servidor Apache es leer un archivo de nombre **.htaccess**, que reside en el mismo directorio que los archivos protegidos. En él se identifican los usuarios o grupos de usuarios que tienen permiso para acceder a los archivos de ese directorio. Usuarios y grupos pueden, a su vez, definirse en el mismo **.htaccess**, en otro archivo de configuración del servidor o, incluso, en uno del sistema. Cuando haya un intento de acceso a estos archivos, el servidor solicitará autenticación al navegador, y este mostrará una ventana que pide identificarse e indica el **realm** (entorno) correspondiente.

Entorno Rabbit

En este entorno necesitamos definir usuarios y grupos. Los archivos se asocian a grupos, y los usuarios pertenecen a uno o varios de ellos.

> UTILIDAD

Desde el punto de vista de un sistema dedicado, podemos mostrar en una página web las variables de proceso de nuestra aplicación y ejecutar procesos a pedido. Nuestro sistema puede no solo tener una interfaz de control, sino que también es capaz de mostrar su estado de operación. El usuario obtiene, entonces, una presentación más amigable y profesional.

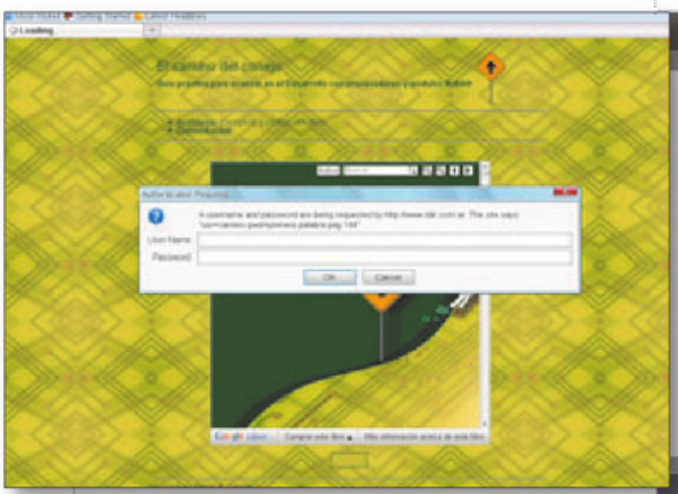
Cuando un archivo tiene permiso, ya sea de lectura o de escritura, asociado a uno o más grupos, solo quienes pertenezcan a alguno de ellos podrán acceder a esos archivos y realizar dicha acción. Para indicar que un elemento requiere autenticación, contamos con una entrada especial en el directorio del servidor.
El valor **1** (bit 0 seteado) indica que los usuarios del grupo **0** podrán leer **/secret.shtml** o ejecutar **/mifun.cgi**. El valor **0** señala que ningún grupo tiene permiso de escritura. El **realm**, en este caso, es **Secreto**. Luego, creamos el usuario **cone** con clave **jito**, y lo asociamos al grupo **0**. ■

> Código 09

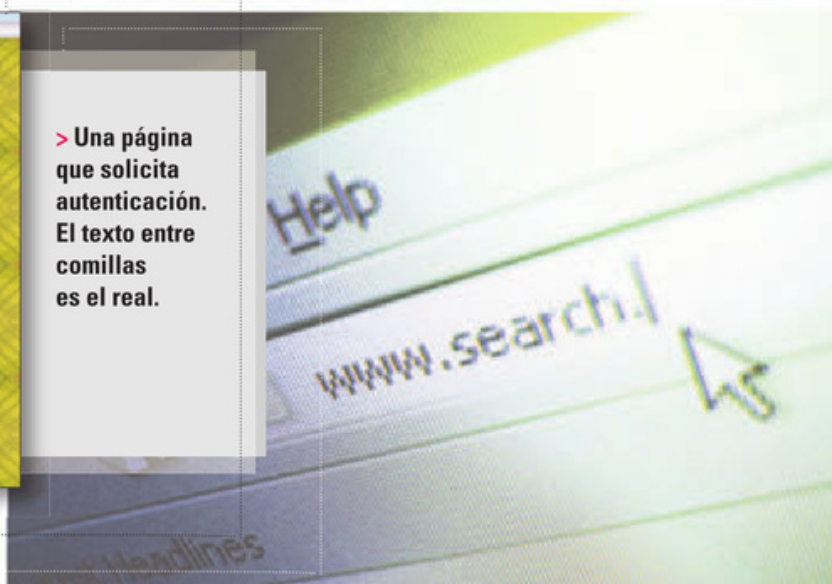
```
SSPEC_RESOURCE_P_XMEMFILE("/secret.shtml",secret_html,"Secreto",1,0
                ,SERVER_HTTP, SERVER_AUTH_ANY),
SSPEC_RESOURCE_P_FUNCTION("/mifun.cgi",mifun,"Secreto",1,0,
                SERVER_HTTP,SERVER_AUTH_ANY)
```

> Código 10

```
uid=sauth_adduser("cone", "jito", SERVER_HTTP);
                sauth_setusermask(uid, 1, NULL);
```



> Una página que solicita autenticación. El texto entre comillas es el real.



RABBITWEB

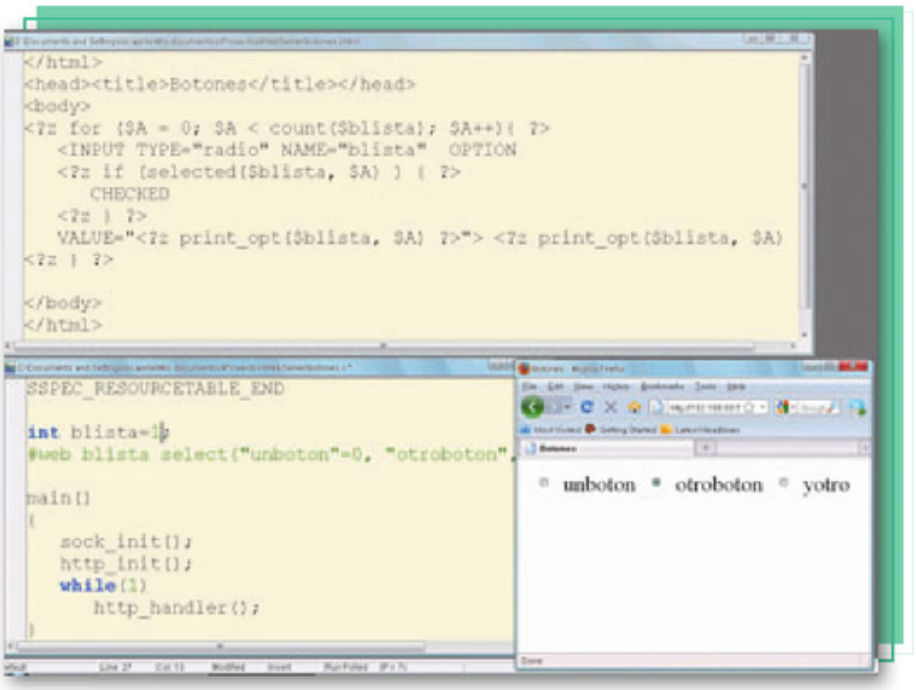


El lenguaje script, interpretado por el servidor HTTP al servir la página web, incluye extensiones a C para simplificar el trabajo con variables. Veamos de qué se trata.

El lenguaje de RabbitWeb, **ZHTML**, se encuentra embebido dentro de **tags**, de modo similar a **PHP** o **SHTML**, y permite realizar páginas dinámicas. El código se aloja en varias líneas dentro del envoltorio <?z ?>. Cada variable referida en el script está precedida por los signos **$** o @: el primero indica que estamos haciendo referencia al valor que se intenta introducir al enviar la página; en tanto que el segundo señala que nos referimos al valor que tiene la variable en el programa principal. La sintaxis es reducida y similar a C:

- ejecución condicional mediante **if()**
- bloques iterativos mediante **for(; ;)**
- llamadas a funciones internas de RabbitWeb.

Por ejemplo, el código que se presenta a continuación hace que el servidor muestre el valor de la variable **mivar:** en rojo si esta posee **1**, y en azul si tiene otro valor (**Código 11 de la página 245**).



Generación de botones con RabbitWeb.
En la esquina inferior derecha, el navegador muestra el resultado.

Selectores y botones

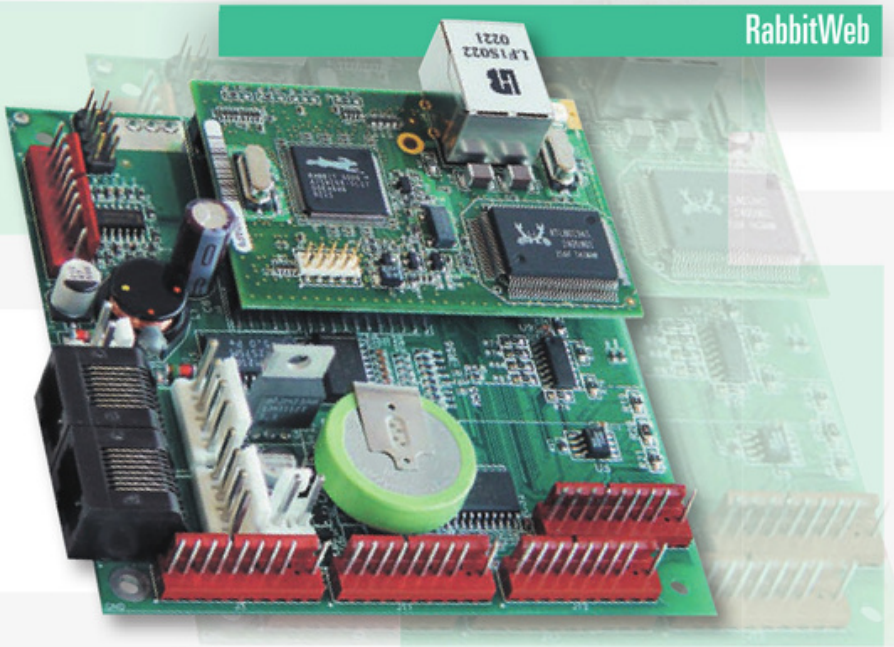
Existen funciones propias para soportar la generación de selectores y botones, y también podemos marcar la opción corriente. El siguiente código genera un selector desplegable y una lista de botones (**Código 12 de la página 245**). Vemos aquí que la función **count($blista)** determina la cantidad de botones (opciones) en **blista**. Valiéndonos de **for(; ;)** armamos un bloque iterativo. Las variables de estos son internas, y sus nombres son de una sola letra.

Funciones

También existen funciones para determinar si la página está siendo mostrada como resultado de un **GET** (pedido de página) o de un **POST** (envío de datos), y en este caso, si cometimos un error en la información enviada (**Código 13**). Los rangos de los valores se especifican al declarar las variables, como veremos más adelante. Estas son solo algunas de las prestaciones; vayamos ahora a las extensiones al lenguaje C.

Las extensiones a C

Dynamic C incorpora una serie de extensiones al lenguaje C para soportar la comunicación con el script y los servicios adicionales. Por ejemplo, podemos definir grupos de usuarios en RabbitWeb, cuyos nombres usaremos en las funciones de autenticación vistas en la **página 9**.



> Código 11

```
<?z if($mivar==1){ ?>
        <FONT COLOR="#FF0000">
<?z } ?>
<?z if($mivar!=1){ ?>
        <FONT COLOR="#0000FF">
<?z } ?>
        <?z print($mivar) ?>
```

> Código 12

```
<SELECT NAME="milista">
<?z print_select($milista) ?>
</SELECT><br>
<?z for ($A = 0; $A < count($blista); $A++){ ?>
        <INPUT TYPE="radio" NAME="blista" OPTION
        <?z if (selected($blista, $A) ){ ?>
                CHECKED
        <?z } ?>
        VALUE="<?z print_opt($blista, $A) ?>"> <?z print_opt($blista, $A) ?>
<?z } ?>
```

> Código 13

```
<?z if(update()){ ?>
        estoy respondiendo a un POST
        <?z if(error()){ ?>
                hay un error en alguno de los valores
        <?z } ?>
        <?z if(error($mivar)){ ?>
                hay un error en el valor de mivar en particular
        <?z } ?>
<?z } ?>
<?z if(!update()){ ?>
        estoy mostrando la página por un GET
<?z } ?>
```

> Código 14

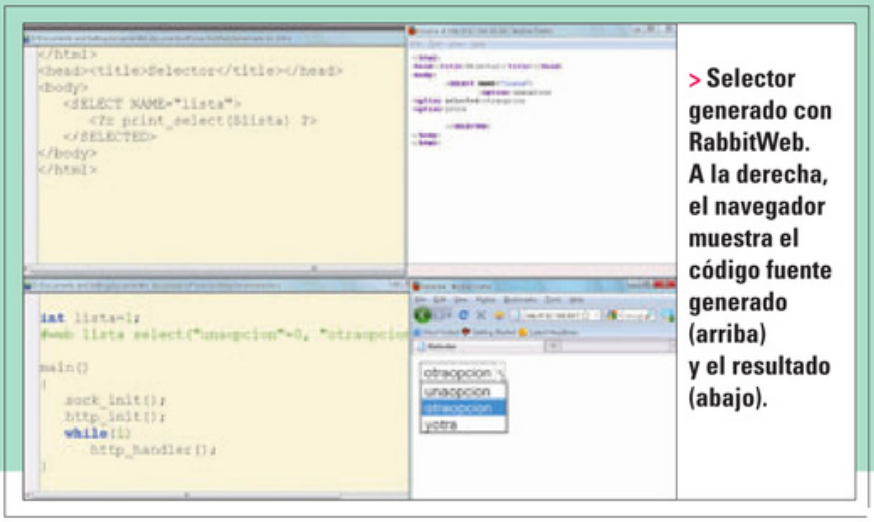
```
#define USE_RABBITWEB 1
#web_groups GRUPO1, GRUPO2

#web mivar (($mivar >= -5) && ($mivar <= 5)) auth=basic,digest \
groups=GRUPO1(rw),GRUPO2(ro)
```

> RABBITWEB

RabbitWeb es introducido por Rabbit en el año 2005. Su intención es minimizar e integrar el trabajo de implementar interfaces de usuario. Con RabbitWeb, el usuario puede ver todo como una unidad, repartiendo la tarea entre un script y un programa principal, en vez de tener un programa en C por un lado, y una colección de scripts CGI y páginas web por el otro.

Las variables que vamos a mostrar o cambiar vía web se registrarán después de declararlas. De manera opcional, al mismo tiempo o luego de hacerlo, podemos definir los grupos de usuarios y los permisos de acceso asociados a ellos. También podemos asociar una función de evaluación para comparar los valores que se quieren modificar con rangos permitidos (**Código 14**). Solo los usuarios pertenecientes a **GRUPO2** y **GRUPO1** pueden acceder a **mivar**, y únicamente los de este último pueden modificarla. El valor por ingresar debe estar entre **-5** y **5**. El signo **$** por delante del nombre hace referencia al valor que se quiere ingresar desde el navegador, en vez del que tiene la variable en el sistema.



> Selector generado con RabbitWeb. A la derecha, el navegador muestra el código fuente generado (arriba) y el resultado (abajo).

Para trabajar con selectores y botones, asignamos los valores posibles en un entero, y los textos del selector como cuando trabajamos con un enumerador de C. Cuando el navegador envía datos, estos son diferentes de los actuales, y cumplen con los requisitos especificados de acceso y rango de validez, podemos ejecutar una función que haga algo con todo eso. Lo llevamos a cabo de la forma que se ve en el **Código 15**.

Utilización

Cuando empleamos RabbitWeb, no necesitamos declarar variables en la estructura de directorio; solo precisamos el tipo **MIME** para **ZHTML** con su **handler** **(Código 16)**.
Las variables declaradas mediante **#web** son accedidas dentro del script **ZHTML** en tiempo de ejecución de este, y no es necesario incluirlas en el directorio del servidor web ni hacer nada adicional.

Soporte AJAX

AJAX (*Asynchronous JavaScript And XML*) es una tecnología que permite actualizar elementos de una página web sin tener que recargarla. Un programa JavaScript se ejecuta en el navegador web y, periódicamente, solicita al servidor los elementos de interés.
Por lo general, se trata de archivos XML que contienen los valores de las variables por actualizar.
Como vemos, no se trata de algo relativo al servidor, sino que todo el trabajo está en el navegador; el servidor solo debe ser capaz de actualizar y entregar dichos objetos.

Utilizando RabbitWeb, podemos generar el archivo XML fácilmente mediante un script, empleando **<?z print (@mivar) ?>**.
Un ejemplo de esto se ve en el **Código 17.**

Ajax es una aplicación que actualiza las páginas de forma dinámica.

Servidor web con Rabbit

A continuación, observamos el código C, que incorpora todo lo visto hasta el momento, en el siguiente orden: configuración de la dirección IP, carga de archivos, tabla de tipos MIME, directorio del servidor, declaración de la variable que usaremos, función que se ejecuta ante un cambio y programa principal **(código 18 de la página 247).**

> Código 15

```
int milista;
#web milista select( “optuno” = 1, “optdos”, “opttres”, “optcuatro” )

void cambio(void)
{
        // tarea a ejecutar si se modifica mivar
}

#web_update mivar cambio
```

> Código 16

```
SSPEC_MIME_FUNC(“.zhtml”, “text/html”, zhtml_handler)
```

> Código 17

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<values>
        <analog>
                <ch> <?z print(@entradaAD) ?> </ch>
        </analog>
</values>
```

> Código 18

```
#define TCPCONFIG          0
#define USE_ETHERNET       1

#define MY_IP_ADDRESS  "192.168.69.54"
#define MY_NETMASK     "255.255.255.0"

#define MY_GATEWAY     "192.168.69.1"

#use "dcrtcp.lib"
#define USE_RABBITWEB 1
#use "http.lib"

#ximport "index.zhtml" index_zhtml
#ximport "users.gif"       users_gif

SSPEC_MIMETABLE_START
        SSPEC_MIME_FUNC(".zhtml", "text/html", zhtml_handler),
        SSPEC_MIME(".html", "text/html"),
        SSPEC_MIME(".gif", "image/gif"),
SSPEC_MIMETABLE_END

SSPEC_RESOURCETABLE_START
        SSPEC_RESOURCE_XMEMFILE("/", index_zhtml),
        SSPEC_RESOURCE_XMEMFILE("/index.zhtml", index_html),
        SSPEC_RESOURCE_XMEMFILE("/users.gif", users_gif),
SSPEC_RESOURCETABLE_END

int mivar;
#web mivar ($mivar > mivar)

void cambio(void)
{
        printf("Cambio de valor: %d\n", mivar);
}

#web_update mivar cambio

main()
{
        sock_init();
        http_init();
        tcp_reserveport(80);
        while(1)
                http_handler();
}
```



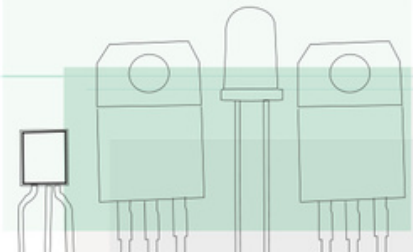
> DETALLE DE OPERACIÓN

El servidor web se ejecuta cada vez que llamamos a **http_handler()**. Es internamente una máquina de estados que va realizando paso a paso lo que necesita. El script **ZHTML** se ejecuta dentro de este tiempo, cuando el usuario solicita (mediante el navegador) la página web que lo contiene.

> Código 19

```
</html>
<head><title>Mi web server</title></head>
<body>
<center><IMG SRC="users.gif" ></center>
<hr>
<H1>Felicitaciones, ha iniciado su primer web server</H1>
<hr>
<form ACTION="index.zhtml" METHOD="POST">
<?z if(updating()) { ?>
        <?z if(!error()) { ?>
                <H2><FONT COLOR="#0000FF">OK</FONT></H2><hr>
        <?z } ?>
<?z } ?>
<table>
<tr><td>Variable:
<td><input TYPE="TEXT" NAME="mivar" SIZE=5 VALUE=<?z print(@
mivar) ?>>
</table>
<input TYPE="SUBMIT" VALUE="Cambiar"></form>
</body>
</html>
```

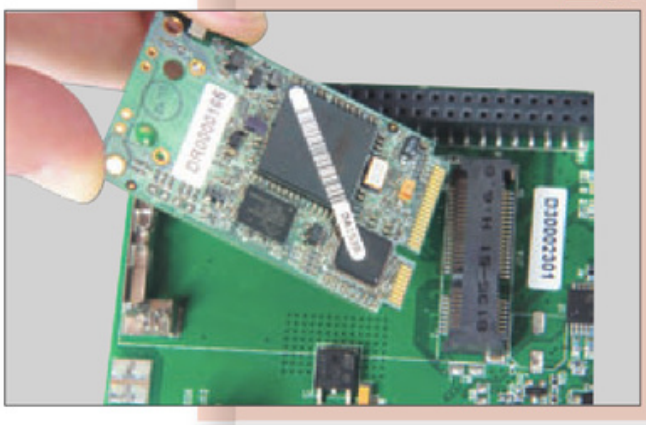
Luego, vemos el código HTML con el script embebido. Mostramos una imagen, la variable y un botón para cambiar el valor. Si el valor introducido cumple la condición que se le pide (mayor que el anterior), al aceptarlo, se mostrará OK en azul (**Código 19**). Esto es todo lo necesario para realizar un servidor web con Rabbit: el código del programa y la página en sí, sin conversiones ni pasos intermedios. En el próximo apartado lo veremos en acción. ■



SERVIDOR WEB CON RABBIT

Llegó el momento de llevar a la práctica los conceptos de networking –en general– que hemos aprendido en el capítulo anterior, y los de Rabbit –en particular– vistos en este capítulo.

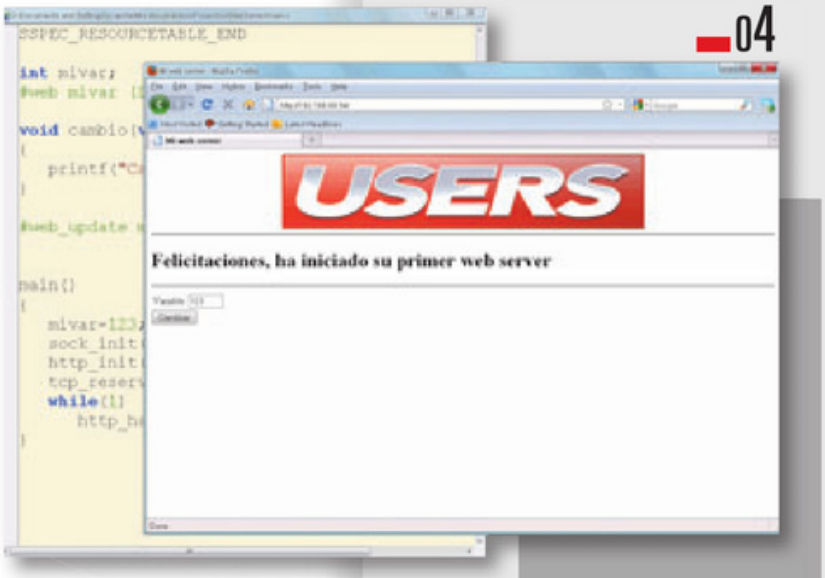
01

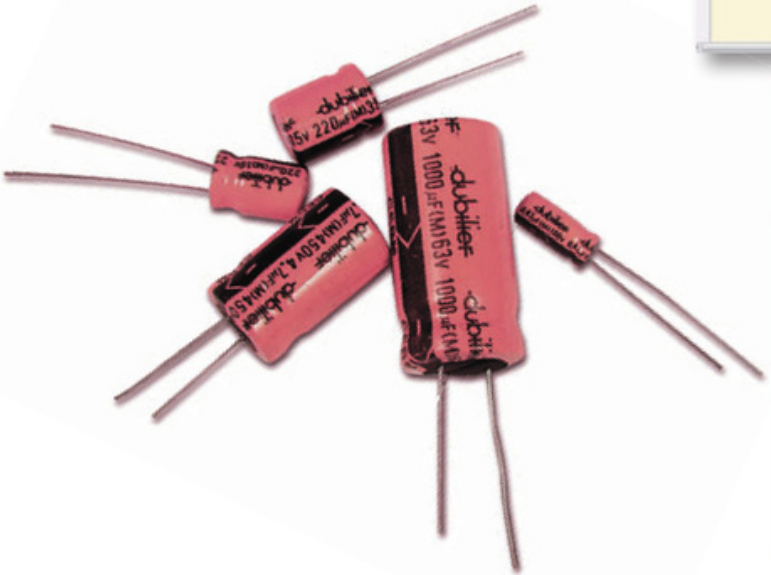




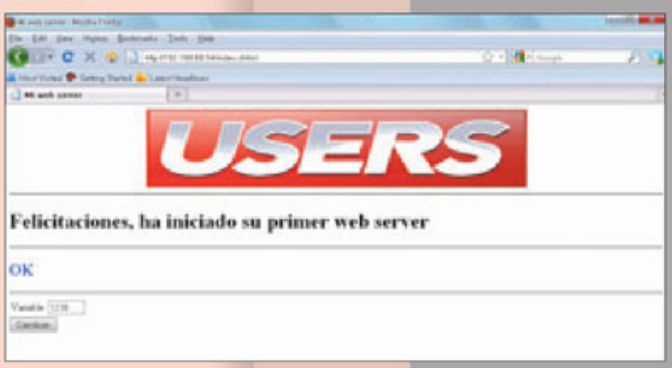
Para conectarnos con el módulo Rabbit, podemos usar un switch. Si disponemos de una red o podemos consultar a alguien, esta es la opción más aconsejable. Si no es así, en estas páginas sugerimos un cable cruzado para unir la PC y Rabbit directamente, de modo de poder realizar la práctica en forma sencilla, aun para quienes no tienen conocimientos avanzados sobre networking. ■

04





05

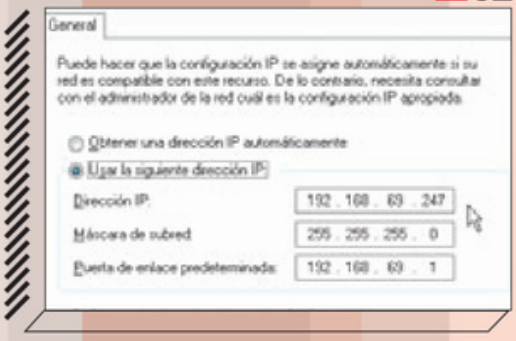


PASO A PASO

01. Mediante un patch-cord cruzado, unimos el conector de red de la PC (RJ-45) con el del módulo Rabbit. Si el módulo es un minicore, como el RCM5700 utilizado en clase la 17, lo insertamos cuidadosamente en la placa de desarrollo.

02. Debemos configurar la dirección IP de la PC dentro de la misma red que Rabbit. Utilizaremos la IP 192.168.69.247, con máscara 255.255.255.0. Para esto, seleccionamos **Conexiones de red** y, luego, el port Ethernet correspondiente en el **Panel de control**.

02

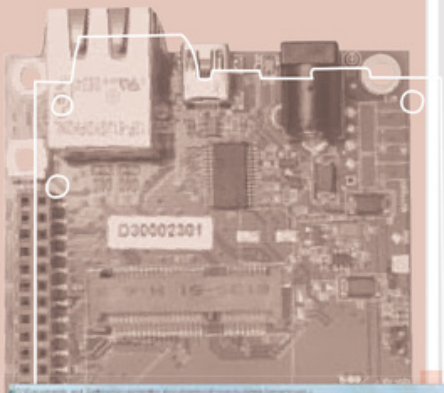


03





La IP de Rabbit se encuentra al principio del código del programa.



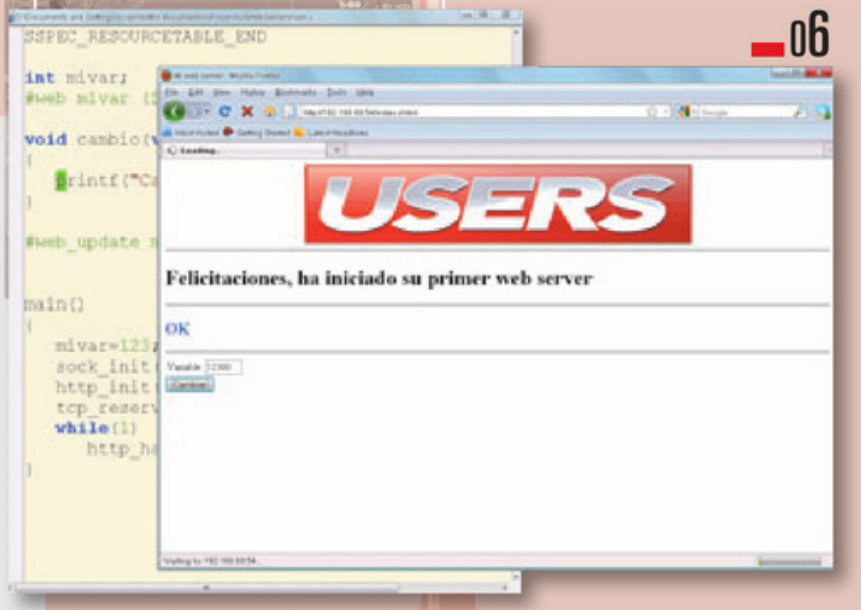
03. La dirección IP de Rabbit está al principio del código del programa. Usaremos 192.168.69.54, máscara 255.255.255.0. Al hacer correr el programa, el módulo Rabbit estará en la red y podremos conectarnos a él.

04. Abrimos el navegador web utilizado normalmente y colocamos la URL de Rabbit: http://192.168.1.54/. Observaremos la página servida por él; de no ser así, tendremos que revisar los pasos anteriores.

05. Modificamos el valor mostrado, ingresando uno mayor, y presionamos el botón **Cambiar**. El valor es aceptado y recibimos la página nuevamente, esta vez, con el texto **OK** en azul, para señalar que todo salió bien. Vemos, además, una indicación en la ventana "stdio" de Dynamic C.

06. La acción anterior provoca que Rabbit ejecute una función al recibir un valor aceptable. Ponemos un breakpoint en ella y repetimos la acción anterior. Veremos que la ejecución se detiene en esa función, y el navegador queda esperando.

06



CONCEPTOS INTEGRADOS

Hasta aquí hemos visto cómo funcionan algunos conceptos por separado; ahora nos detendremos en todo lo que involucra el acceso a una página web.



Un servidor web es una máquina de estados (veremos este tema más adelante). Vimos que el servidor espera una conexión y que, cuando la recibe, la atiende y aguarda a que se le haga un pedido relacionado con su tarea principal, que es servir páginas web.

Cada objeto se pide por separado. Esto quiere decir que si el contenido de una página HTML hace referencia a cuatro imágenes, el servidor recibirá cinco pedidos: el primero por la página, y uno por cada imagen. Podemos decir entonces, resumiendo, que un servidor web es un servidor de objetos. Internamente, mantiene una relación entre los nombres de los objetos en la Web (URL) y su ubicación en el soporte físico en que se encuentren, como un disco duro, una memoria flash, etc. Como tareas más avanzadas, el servidor puede, además, interpretar ciertas secuencias de escape en la página, como **SSI** en **SHTML**, tal como vimos en la **página 8**. También es posible enviar estas secuencias de escape a un programa externo para que las procese. De esta forma, existe la posibilidad de embeber programas dentro de una página web, como sucede con **PHP** o con **RabbitWeb**, como ya explicamos.

Sobre el navegador

Un navegador web es un programa encargado de convertir las definiciones



> ¿QUÉ OCURRE AL PEDIR UNA PÁGINA WEB?

Cuando el usuario ingresa una URL en el navegador, este resuelve el nombre del servidor a una IP mediante DNS. A continuación, se conecta con el servidor y solicita la URL, luego de lo cual este le entrega el archivo (generalmente, HTML). El navegador lo procesa, identifica los objetos externos y repite la operación con cada uno de ellos.

que recibe en formato HTML, en algo que el usuario pueda ver. El código HTML define tablas, listas e imágenes. El navegador interpreta este código y realiza el dibujo en pantalla. Cuando interpreta que debe mostrar una imagen –por ejemplo, al encontrarse con **<IMG SRC="miimagen.jpg">**–, efectúa el pedido correspondiente al servidor. Para cada elemento externo que localiza, hace un pedido. Una vez mostrada la página, termina su operación.
Para realizar tareas más avanzadas, es posible instruir al navegador mediante directivas embebidas en el código HTML, para que, periódicamente, vuelva a pedir la página web, con lo cual se produce el redibujo completo de ella.
También es posible que una página web pueda traer embebido un **script Java**, que el navegador ejecutará. Este script puede utilizarse para refrescar en segundo plano ciertos elementos de la página, y es el principio en que se basa **AJAX**.
Para que el navegador obtenga y muestre en forma periódica un objeto (una variable o una imagen), debe ejecutarse un script en el que realice esta tarea en segundo plano.

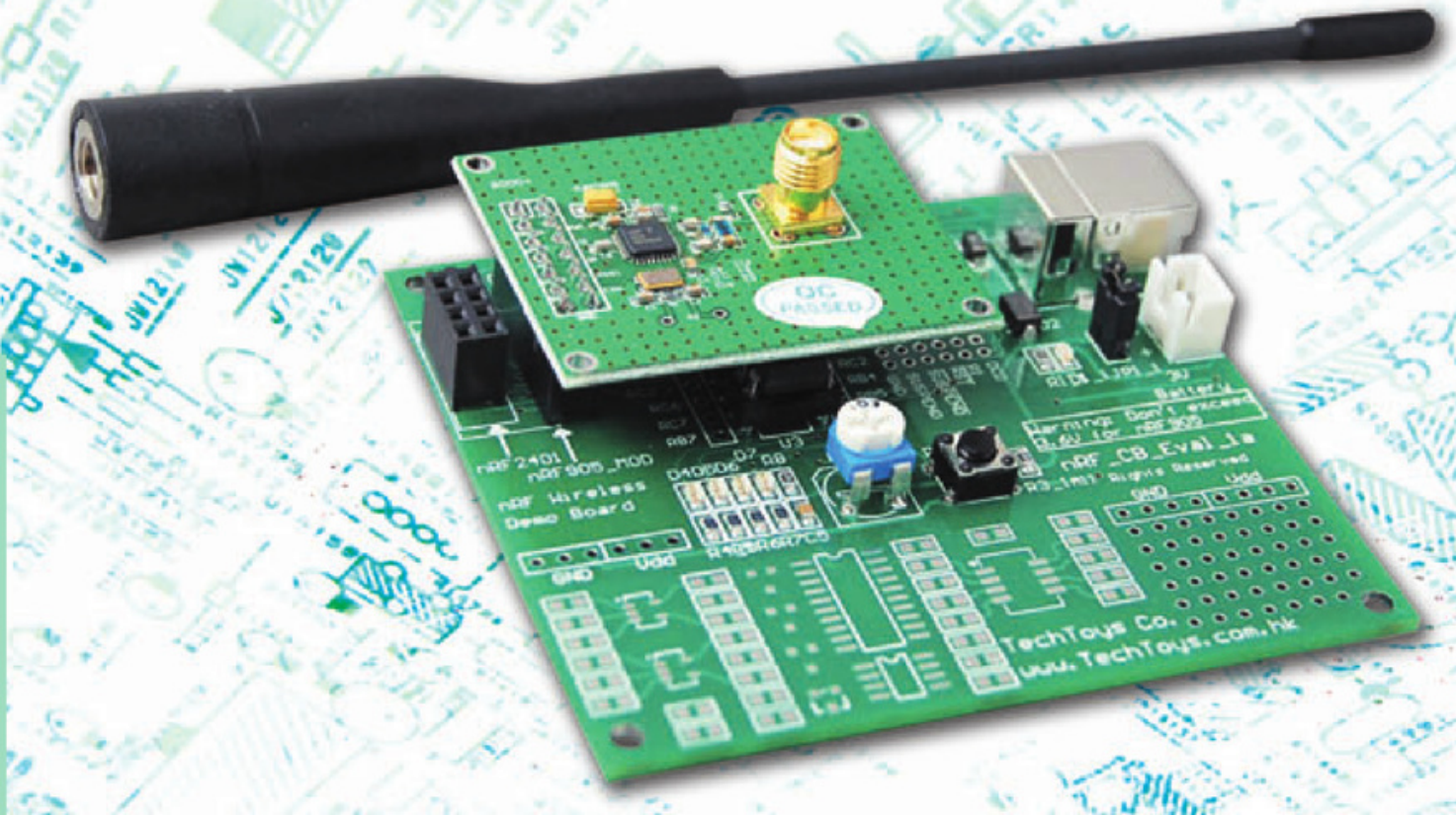
El servidor web de Rabbit opera de modo similar a cualquier otro, como Apache.

> SERVIR UN ELEMENTO

Como Rabbit es Von Neumann, no es necesario diferenciar si la memoria es flash o RAM; con solo tener la dirección de esta, se ubica un elemento. En dicha posición, existe un entero largo, que contiene la longitud del objeto y, a continuación, están los bytes que lo componen. Por este motivo, para enviarlo, simplemente tomamos los bytes uno a uno y los colocamos en el buffer de TCP del socket correspondiente.

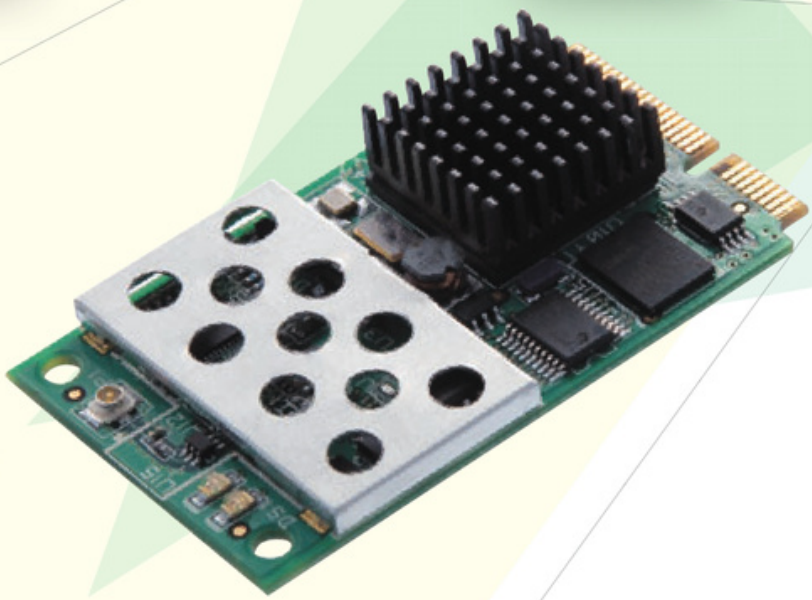
Servidor web en Rabbit

El servidor web de **Rabbit** opera de modo similar a cualquier otro, como Apache. Sin embargo, dado que no existe un sistema operativo que lo respalde, realiza ciertas operaciones de manera simplificada.
Al recibir el pedido de un elemento (su URL), el código del servidor verifica la existencia de este en la estructura que forma el directorio, la cual contiene un **puntero a la ubicación en memoria del elemento** en cuestión.
De la URL, el servidor extrae la extensión del elemento (**JPG, GIF**, etc.) y, mediante la estructura que contiene los tipos **MIME**, indica al navegador el contenido por servir.
En dicha estructura también figura el **handler**, que interpreta el contenido, por lo que si se trata de **SHTML** o **ZHTML**, el servidor puede extraer las secuencias de escape y entregárselas al procesador correspondiente. Este las interpretará y procederá a ubicar las variables para efectuar las operaciones que se le pidan en el script.
Todo esto ocurre durante el tiempo en que se llama a **http_handler()**, como vimos en la **página 13**.



WI-FI CON RABBIT

Algunos módulos Rabbit incluyen hardware para conectarse a una red Wi-Fi 802.11 b/g, sin agregados externos, solo colocando la antena.



Wi-Fi es un nombre que engloba los estándares **IEEE 802.11**. Mediante esta tecnología, dos dispositivos pueden comunicarse entre sí **ad-hoc** o a través de un **access point** (punto de acceso) que provee la interconexión, en el modo infraestructura.
Una red **802.11** tiene un identificador **SSID** (*Service Set Identifier*), que se anuncia periódicamente, permitiendo que otros dispositivos puedan asociarse a la red. Para que los datos no sean vistos por aquellos que no deben hacerlo, se suele **encriptar** y autenticar a quienes quieren acceder a la red.
Los métodos más utilizados son:

- **WPA/TKIP**: acceso protegido a Wi-Fi usando el protocolo de integridad temporaria de claves.
- **WPA2/CCMP**: acceso mediante un protocolo basado en encriptación AES, el estándar avanzado.

Los métodos de encriptación dependen de una clave que puede ser **PSK** (clave precompartida) o adquirida mediante el esquema **EAP** (protocolo de autenticación extensible). El primer caso se conoce como esquema residencial y, el último, como enterprise.

Rabbit 5000 soporta Wi-Fi en el chip, WPA2 y autenticación enterprise.

Por lo general, los equipos inalámbricos no utilizan una dirección IP fija, sino que la adquieren al momento de ingresar a la red de un servidor **DHCP**, que además les informa sobre la dirección del router y el o los servidores DNS. Es común que este servidor esté en el mismo access point.
El **Rabbit 5000** incorpora soporte para Wi-Fi dentro del chip, incluyendo **WPA2** y autenticación de tipo enterprise. La forma de trabajar con un módulo Wi-Fi es exactamente igual a lo ya visto para los módulos Ethernet.
A continuación, examinaremos las pequeñas diferencias que existen. En general, debemos especificar el uso de la interfaz Wi-Fi (y si empleamos DHCP), de la manera que se muestra en el **Código 20.**

Conexión y encriptación

Podemos conectarnos sin usar encriptación ni restringir el acceso, por ejemplo, en una red de desarrollo. De lo contrario, tanto para **WPA/TKIP** como para **WPA2/CCMP**, necesitamos compilar el código de soporte de **WPA** y especificar el tipo de encriptación. En el último caso, además, especificamos la necesidad de compilar el código de soporte de **AES (Código 21 de la página 253).**
Como vimos, la clave **PSK** puede colocarse en ASCII. Sin embargo, esto provoca una demora importante (decenas de segundos) para la generación, por lo que es conveniente hacerlo solo la primera vez y, luego, cambiarla por el ingreso directamente en hexadecimal.

> **Código 20**

```
#define TCPCONFIG 5                    // si utilizamos Wi-Fi

#define USE_WIFI 1
   #define IFC_WIFI_SSID "mired"
```

> Código 21

```
// red abierta
   #define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_NONE

// red WPA
#define WIFI_USE_WPA
        #define IFC_WIFI_ENCRYPTION      IFPARAM_WIFI_ENCR_TKIP
#define WIFI_VERBOSE_PASSPHRASE

// red WPA2
#define WIFI_USE_WPA
#define WIFI_AES_ENABLED
        #define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_CCMP
// clave
        #define IFC_WIFI_WPA_PSK_PASSPHRASE "miclave"
        #define IFC_WIFI_WPA_PSK_HEXSTR \
"25BB6E6DE7937997D978A17932DD9ABC4C4F99E84FE95F8F5989964D8AC97004"
```

> Código 22

```
#define TCPCONFIG 5
#define USE_WIFI                    1
   #define IFC_WIFI_SSID   "mired"
   #define IFC_WIFI_ENCRYPTION IFPARAM_WIFI_ENCR_NONE

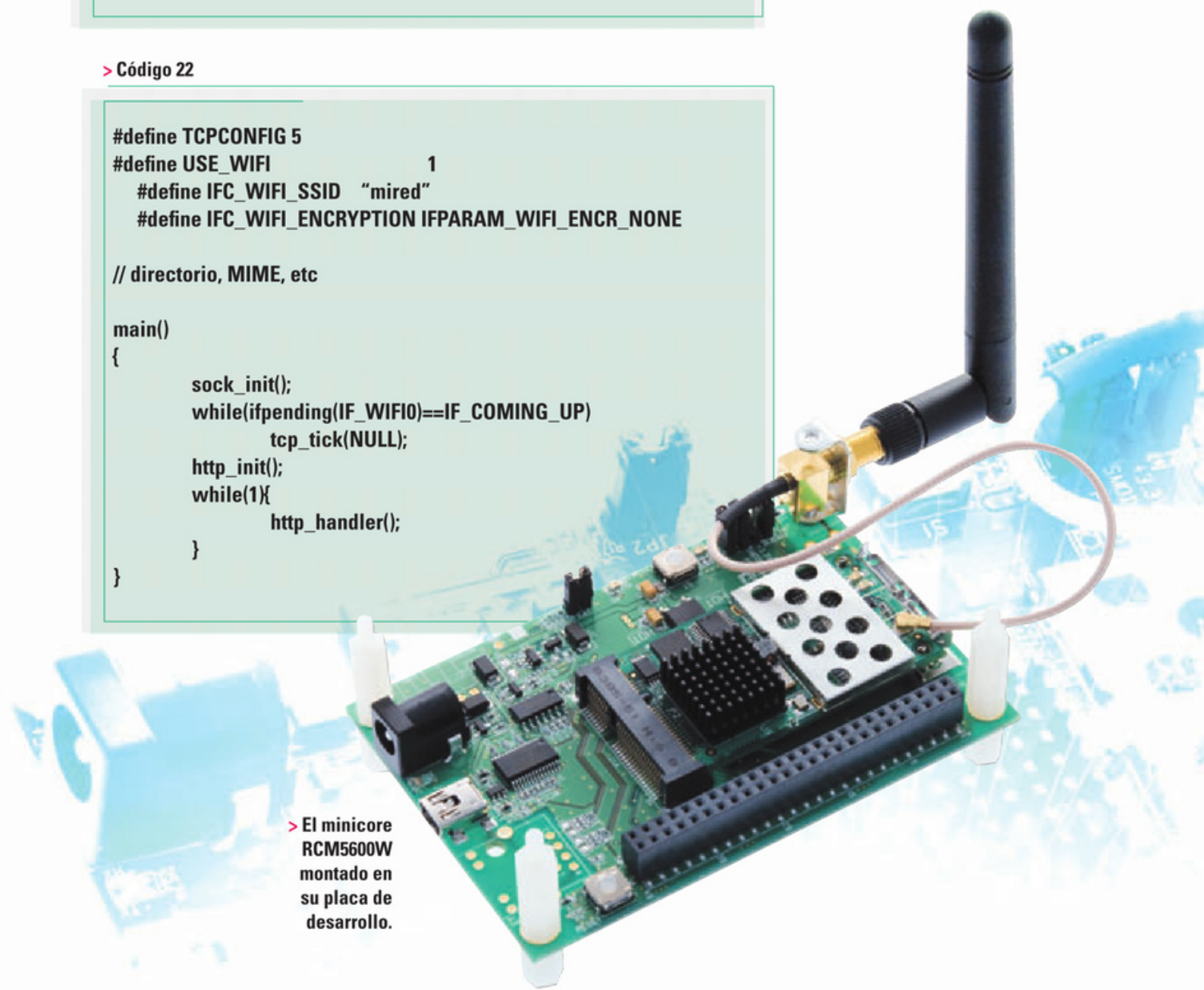
// directorio, MIME, etc

main()
{
        sock_init();
        while(ifpending(IF_WIFI0)==IF_COMING_UP)
                tcp_tick(NULL);
        http_init();
        while(1){
                http_handler();
        }
}
```

Inicio del servidor web

En el programa principal, luego de la llamada a **sock_init()**, debemos revisar el estado de la interfaz en espera de la conexión con el access point y la posterior asignación de la dirección mediante DHCP. Esto se realiza, por ejemplo, a través de un lazo de espera, a menos que debamos efectuar otra tarea (**Código 22**).

Como vemos, el código del servidor web, modificado para un módulo Wi-Fi, difiere del utilizado en Ethernet solo en lo que hemos visto en estas dos páginas.



> El minicore RCM5600W montado en su placa de desarrollo.

MULTITAREA COOPERATIVO

Cuando debemos atender más de una tarea a la vez, necesitamos un esquema que nos permita simplificar la operación de desarrollo del código involucrado.

Hasta ahora, nos enfocamos en realizar solamente una tarea. Pero si debemos ocuparnos de más de una operación al mismo tiempo, resultaría engorroso escribir el código que manejen ellas, porque sería tan complejo que solo lo entendería quien lo hubiera hecho (y a veces ni él mismo), y resultaría muy difícil de mantener y modificar. Ni hablar cuando la cantidad de tareas se incrementa.

> Entendemos por multitarea a la aparente ejecución simultánea de varias acciones.

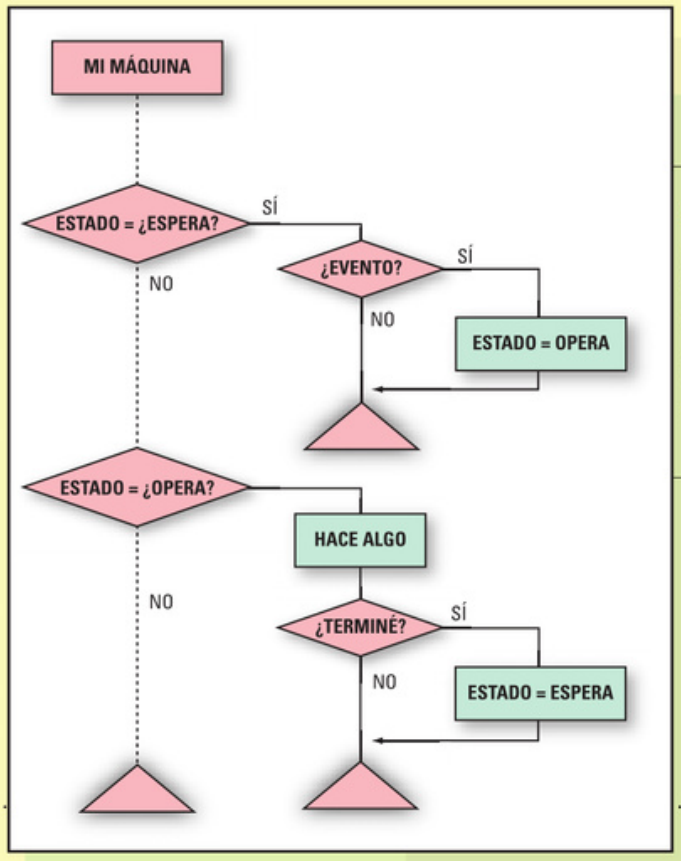
Multitarea

Entendemos por **multitarea (multitasking)** a la **aparente ejecución simultánea de varias acciones**. Decimos que es "aparente" porque sabemos que un procesador solamente puede ejecutar una instrucción (tarea) a la vez. Para que varias se ejecuten de verdad al mismo tiempo, precisamos varias unidades de ejecución, en uno o más procesadores. Cada cambio de tarea (**task switching**) requiere un intercambio del contexto de operación (**context switching**). La forma en que esto se realiza define al tipo de entorno. Los dos más comunes son el **preemptive**, en el que un administrador **interrumpe y conmuta a la fuerza**; y el **cooperativo**, que detallaremos a lo largo de estas páginas.

Multitarea cooperativo

En este esquema, cada tarea es responsable de **ceder el procesador** a las demás cuando no tiene **nada útil que hacer**. El hecho de tener que estar esperando a que ocurra un evento no se considera algo útil, por lo que el esquema se basa en la **detección de los ciclos de espera** y en la **cesión de control** en ellos. Entre las estructuras más utilizadas, encontramos:

- **Handlers** (manejadores), rutinas que atienden un evento y retornan inmediatamente.

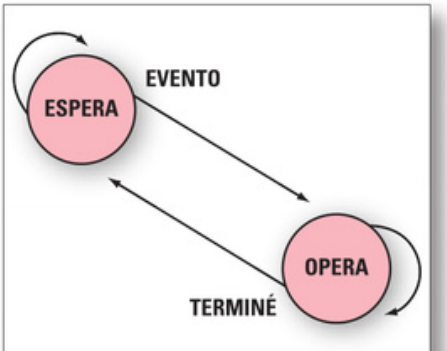


> Diagrama de flujo de una máquina de estados como la descripta en la página siguiente.

■ **Máquinas de estados finitos**, comúnmente llamadas solo máquinas de estados; son rutinas que mantienen un estado interno, que cambia en función de determinados eventos u operaciones, y define la operación por realizar.

Máquinas de estados finitos

En estas máquinas lógicas, durante los ciclos de espera existe un estado que verifica la ocurrencia del evento esperado y cede el control. La máquina solo saldrá de ese estado cuando dicho evento se produzca.
El cambio de contexto es simple, y solo hay un elemento que debe salvarse: el estado. Cada tarea cede el control voluntariamente en los estados de espera, y sabe que las variables compartidas con otras tareas (excepto interrupciones) solo podrán ser alteradas durante los tiempos en que ella lo permite. En caso de no existir ciclos de espera, el código tomará la forma de handlers.
Las tareas serán llamadas de a una por un loop principal. La única manera de que todas las tareas se ejecuten es que todas cedan el control periódicamente, como vemos en el **Código 24.**
Por el contrario, si varias tareas operan sobre el mismo elemento, tendremos que implementar un esquema que permita tener uso compartido de él. Una alternativa común es usar un **semáforo**.



> Diagrama de estados de la máquina ejemplificada. Observemos cómo se indican los cambios.

> Código 23

```
void mimaquina(void)
{
enum { ESPERA=0, OPERA};
static int estado=ESPERA;

        switch(estado){
        case ESPERA:
                if(evento)
                        estado=OPERA;
                break;
        case OPERA:
                // hace algo
                if(termine)
                        estado=ESPERA;
                break;
        }
}

void mihandler(void)
{
                if(evento) {
                        // hace algo
                }
}

main()
{
        while(1){
                miamaquina();
                mihandler();
        }
}
```

> Código 24

```
typedef struct {
        int estado;
        void *elemento;
} mistruct;

void mimaquina(mistruct *ptr)
{
        switch(ptr->estado){
                ...
                // hace algo con *ptr->elemento
        }
}

main()
{
mistruct maq1,maq2;

        // incializa estructuras
        while(1){
                mimaquina(&maq1);
                mimaquina(&maq2);
        }
}
```

MULTITAREA COOPERATIVO EN RABBIT



Además de lo que ya hemos visto, Dynamic C incluye un esquema de soporte que permite simplificar la operación de múltiples tareas en forma cooperativa.

Dynamic C incorpora una serie de extensiones al lenguaje C que simplifican el proceso de generación y el mantenimiento de código. Si bien es posible utilizar las técnicas de programación comunes –y, de hecho, las funciones del stack TCP/IP de Rabbit lo hacen–, y nada obliga a utilizar estas extensiones propietarias, seguramente el lector las encontrará interesantes.

Costates (co-sentencias)

Mediante la palabra clave **costate**, identificamos un bloque que se asocia con una tarea. Internamente, el sistema maneja cada **costate** como si fuera una máquina de estados. Cada **costate** tiene su propio puntero de programa (*statement pointer*) para determinar qué ítem en la lista debe ejecutarse cuando le corresponde. La cesión de ejecución ocurre en los puntos donde existen **sentencias de control** a tal fin, las cuales veremos más adelante. El esquema se ve en el **Código 25**.

> Mediante la palabra clave costate identificamos un bloque que se asocia con una tarea.

Cofunctions (cofunciones)

Se trata de funciones que pueden tener dentro sentencias de control. Así como en un programa en C tomamos una porción de código y la convertimos en una función, en Dynamic C podemos hacer lo mismo con un **costate** y convertir una parte en una **cofunction**. Es posible pasarles argumentos, y devuelven un resultado (**Código 26**).
Las **cofunctions** se llaman desde un **costate** mediante una sentencia de control especial denominada **waitfordone** (**Código 27**).

Sentencias de control

Como vimos, el esquema se basa en el uso de sentencias de control para ceder el procesador a otras tareas. Las sentencias de control son:

> Código 25

```
while(1){
        costate {                                   // tarea 1
                ...
                sentencia de control                        // cede el control a otros costates
                ...
        }
        costate {                                   // tarea 2
                ...
                sentencia de control                        // cede el control a otros costates
                ...
        }
}
```

> Código 26

```
cofunc int micof(int miparam)
{
        ...
        sentencia de control        // cede el control a otros costates
        ...
        return(miresult);
}
```

> Código 27

```
waitfordone {micof}                                    // puede abreviarse
wfd micof;
```

> Código 29

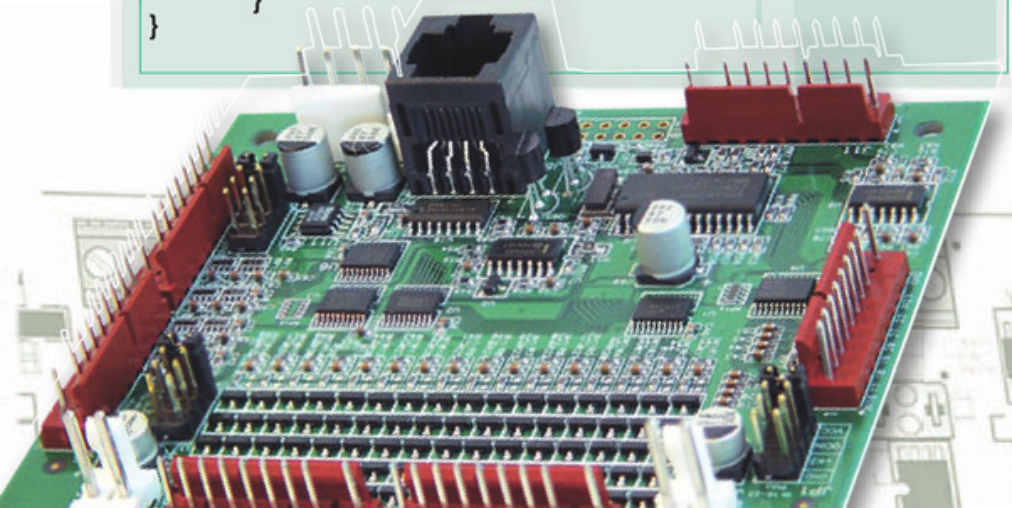
```
cofunc void mimaquina[2](void *elemento)
{
        // hacer algo con *elemento y ceder el control con waitfor y yield
}

main()
{
        while(1){
                mimaquina[0](&elemento1);
                mimaquina[1](&elemento2);
        }
}
```

> Código 30

```
scofunc void miscof(int par)
{
        // hacer algo con un elemento y ceder el control con waitfor y yield
}

main()
{
        loopinit();
        while(1){
                loophead();
                costate{
                        wfd miscof(par);
                }
                costate{
                        wfd miscof(par);
                }
        }
}
```



> Código 28

```
{
        waitfor(evento);
        do {
                //hace algo
                yield;
        } while (!termine);
}
```

- **waitfor(expr):** cede el control hasta tanto expr sea verdadero (1). Al retomar la ejecución, se reevalúa la expresión.
- **yield:** cede el control. Al retomarse la ejecución, se realiza desde la sentencia siguiente.

Por ejemplo, el código de la máquina de estados de la página 21 se muestra en el **Código 29.**

Reutilización de código

En este caso también podemos reutilizar el código, pero como los "estados" son manejados internamente por Dynamic C, disponemos de recursos adicionales:

- El equivalente a pasar el estado como parámetro es una **indexed cofunction**, que se declara de manera similar a un array. El segundo ejemplo de la **página 21** se se muestra en el **Código 29.**
- El análogo al uso de semáforos son las **single user cofunctions**, que se declaran utilizando **scofunc** en vez de **cofunc**. En este caso, existen dos funciones adicionales: una se emplea para inicializar el esquema interno de semáforos, y la otra se llama al inicio del loop (**Código 30**). ■

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Mencione las funciones de networking en Rabbit.

2. Enumere algunos servicios y sockets de Rabbit.

3. ¿De qué forma se realiza la carga de archivos en memoria?

4. ¿Qué es un script CGI?

5. ¿Qué es y para qué sirve Rabbitweb?

6. Mencione algunas de las características de AJAX?

7. ¿Cómo funciona un servidor web?

8. ¿Cuáles son las características de Rabbit 5000?

9. ¿Para qué sirve la multitarea?

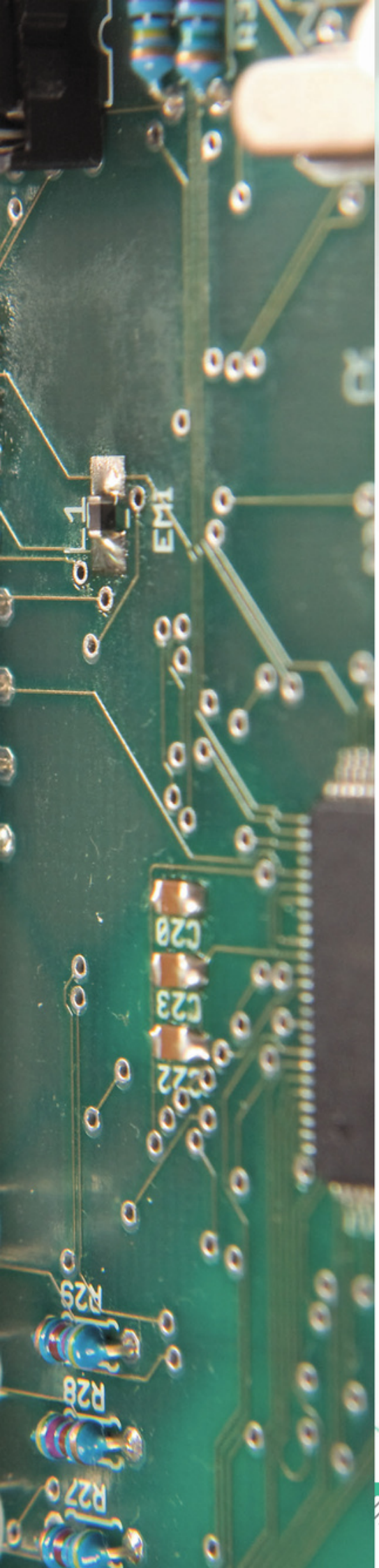
10. ¿Qué es la multitarea cooperativo?

EJERCICIOS PRÁCTICOS

1. Inicialice la interfaz Ethernet llamando a la función sock_init().

2. Cree un servidor web con Rabbit.

3. Implemente el código para trabajar con WiFi y Rabbit 5000.



CAPÍTULO 11

NETWORKING CON PIC

Introducción a networking
Agregar servicios al stack
Configuración del web server
Envío de e-mails
Puente serie-Ethernet
Acceso desde Internet
Arquitectura del PIC18F97J60
Telemetría
Monitoreo de la temperatura por Ethernet
Servidores web dedicados
Encender un led y un relay por Ethernet
Módulo ZeroG
Configuración del stack TCP/IP para Wi-Fi
Actividades

11

INTRODUCCIÓN A NETWORKING

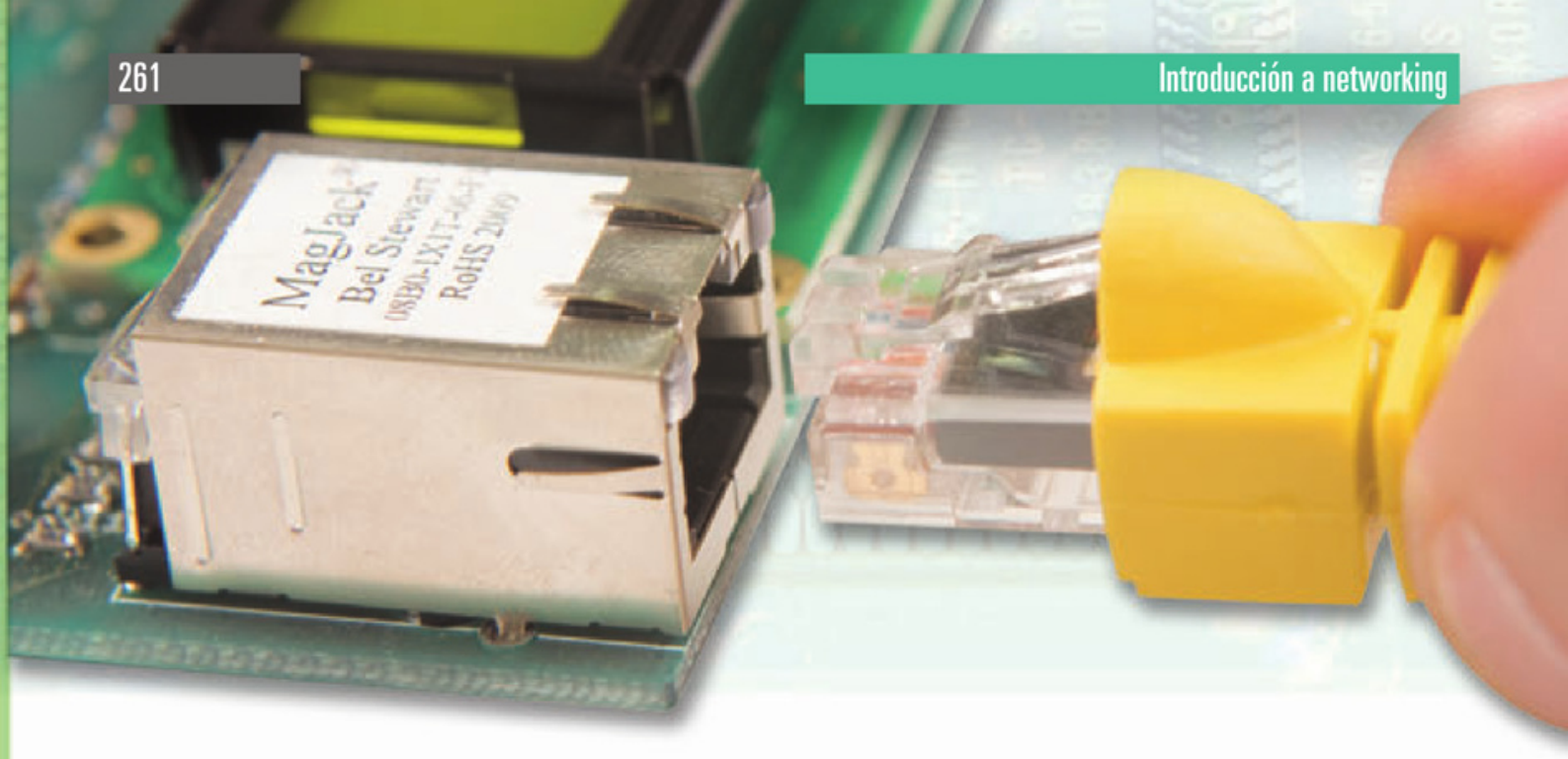
Echemos un vistazo a las herramientas de Microchip, como TCP/IP Configuration Wizard y Microchip Web Preview, para dar los primeros pasos y simplificar nuestro trabajo.

Para comenzar a trabajar, necesitamos tener instalado el programa **MPLAB**, el compilador C18 y Microchip TCP/IP Stack. Tanto MPLAB como C18 fueron explicados en la **clase 12**; por eso, ahora vamos a estudiar Microchip TCP/IP Stack (incluido en el CD 1). Siempre es recomendable consultar la web del fabricante para descargar la última versión: **www.microchip.com/tcpip**.
Para Microchip **TCP/IP Stack**, basta con ejecutar el programa **Microchip Libraries**. En verdad, al hacerlo estaremos instalando una serie de librerías que Microchip provee para el manejo de USB, **mTouch** (sensor capacitivo), pantallas gráficas y Ethernet. Todas son libres, siempre y cuando se utilicen en microcontroladores de la misma firma.

> Microchip TCP/IP Stack está formado por un conjunto de capas que dan acceso a distintos protocolos.



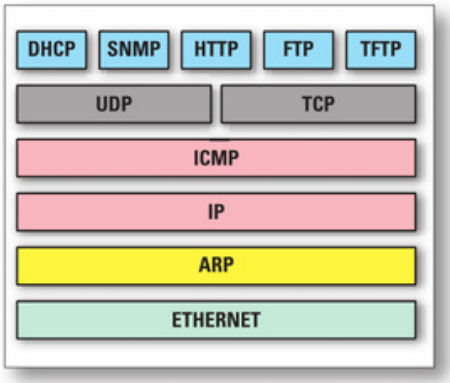
Microchip TCP/IP Stack está formado por un conjunto de capas que dan acceso a distintos protocolos: ARP, IP, ICMP, UDP, TCP, DHCP, SNMP, HTTP, FTP y TFTP. Está programado en C, requiere aproximadamente **32 K** en memoria de programa y es portable a través de las distintas familias de Microchip. Esto quiere decir que funciona en micros de 8, 16 y 32 bits. Además, tiene soporte para encriptación de datos, transmisión de audio y conexión Wi-Fi.
Junto con TCP/IP Stack, se instalan dos aplicaciones sumamente útiles al momento de desarrollar una interfaz Ethernet con PIC: Microchip **TCP/IP Configuration Wizard** y Microchip **Web Preview**. La primera nos permite modificar los parámetros del stack en forma gráfica (sin tener que programar en C), mientras que con la segunda podemos visualizar la página web antes de cargarla en la memoria del PIC. Esta página es



nuestra interfaz con el usuario final, por lo que es muy importante darle el aspecto adecuado: simple, liviana y con gran preponderancia de las variables por monitorear/controlar.

Microchip TCP/IP Configuration Wizard

Podemos darnos una buena idea sobre las posibilidades del stack si estudiamos en detalle el Microchip TCP/IP Configuration Wizard. Si bien esta aplicación nos permite modificar parámetros de la red –como dirección IP, máscara de enlace y default gateway– y seleccionar los protocolos que vamos a utilizar en el sistema, no da la posibilidad de que el usuario agregue su propio código. Al momento de implementar un sistema, necesariamente debemos programar las funciones en C con el MPLAB o cualquier otro entorno de desarrollo.
Lo primero que debemos hacer para empezar a trabajar con el Microchip TCP/IP Configuration Wizard es seleccionar la carpeta desde la cual vamos a operar, es decir, donde está instalado el stack. El proyecto básico se encuentra en **C:\Microchip Solutions library\TCPIP Demo App**. Si bien hay varios ejemplos realizados con el stack, es conveniente empezar con este. Hay otros con soporte para Wi-Fi y librerías de encriptación SSL.
Lo que estamos haciendo es decirle al asistente qué archivos vamos a configurar. Hay que tener presente que el TCP/IP Configuration Wizard agrega o quita funciones del stack comentando código. Sabemos que el C18, como cualquier compilador, no compila el código comentado ni las funciones asociadas, para aho-rrar memoria de programa en el PIC. Por lo tanto, solo debemos seleccionar en el asistente los servicios que realmente necesitamos del stack, como la capacidad para enviar e-mails o servidor seguro.
Es importante tener presente que la memoria de programa del PIC18F97J60 se ocupa en un 55% con las opciones básicas de configuración que veremos más adelante; de esta forma, nos queda un 45% para el código. ■



> Figura 1. El stack está formado por varia capas, cada una brinda servicios a la superior.



> TCP/IP CONFIGURATION WIZARD

Se trata de una aplicación gráfica que permite modificar los parámetros del stack. Sin embargo, para realizar telemetría y control a través de Ethernet, debemos interactuar con el stack mediante funciones callback programadas en C. Estas hacen de nexo entre el firmware del microcontrolador y la página web que ve el usuario final.

AGREGAR SERVICIOS AL STACK



Podemos agregar soporte para distintos servicios, como servidor web, cliente de e-mail, Telnet, FTP y SSL, además de soporte para el módulo Wi-Fi de ZeroG.

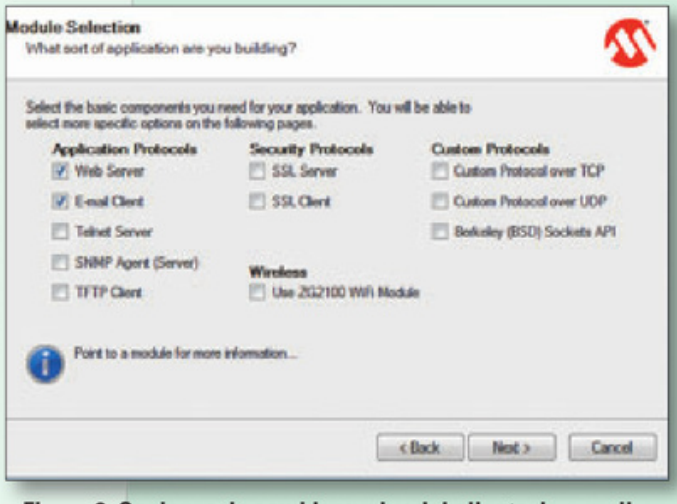
En esta instancia, vamos a agregar el servicio de **servidor web**, que permite acceder a una página por medio de una dirección IP; y el de **cliente de e-mail**, mediante el cual podremos enviar un mensaje automáticamente ante un determinado evento. Esto es útil, por ejemplo, si queremos alertar sobre el aumento de temperatura por encima de un umbral preestablecido. Muchas veces, en aplicaciones industriales es necesario monitorear las revoluciones de un motor; de esta forma, podremos hacerlo con el servidor web y mandar las alertas correspondientes por mail. Como mencionamos anteriormente,

> **Código 01**

```
//#define STACK_USE_DNS
```

Se puede configurar el cliente de e-mail para que el stack envíe un mensaje ante cierto evento.

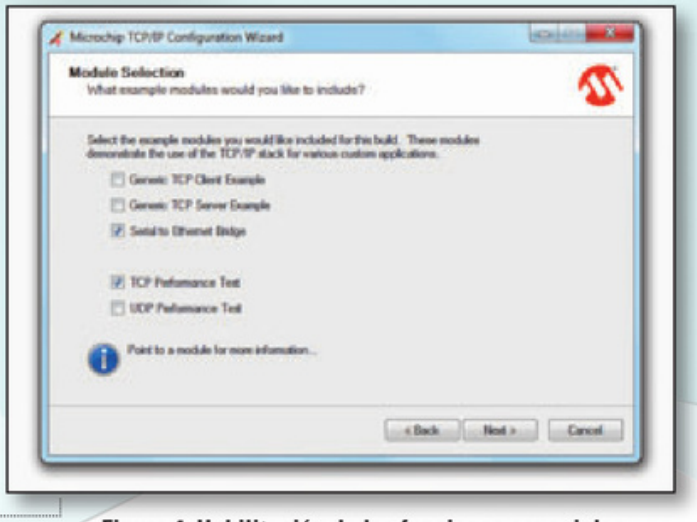
cada servicio que deshabilitamos por medio del asistente es comentado en el stack para que el C18 no lo compile. La **Figura 3** muestra que podemos comentar manualmente cada uno. En esta instancia, cabe mencionar que el **stack de Microchip** está organizado en varios archivos; los encabezados que habilitan los servicios se encuentran en **TCPIPConfig.h.** No todas las funciones del stack están disponibles a través del TCP/IP Configuration Wizard, por lo que es importante acceder al archivo TCPIPConfig.h y comentar los servicios que no precisamos. De esta forma, estaremos optimizando el uso de la



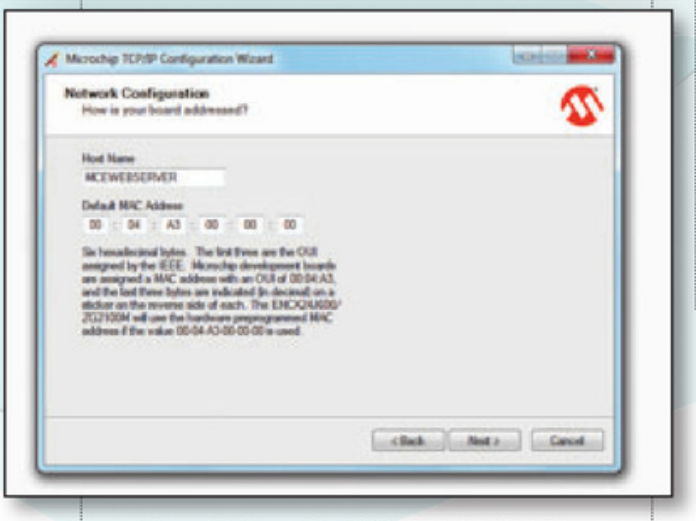
> **Figura 2. Opciones de servidor web y del cliente de e-mail seleccionadas en el TCP/IP Configuration Wizard.**



> **Figura 3. Cómo habilitar o deshabilitar los servicios manualmente.**



> **Figura 4. Habilitación de las funciones especiales Serial to Ethernet Bridge para realizar la conversión serie-Ethernet, y TCP Performance Test para medir el tiempo de viaje de los paquetes.**



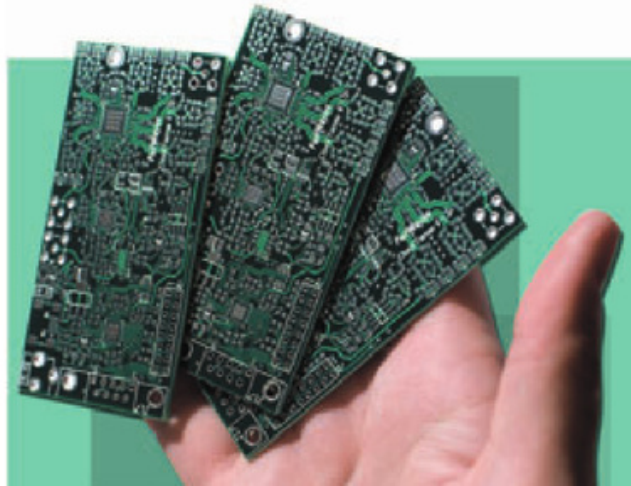
> **Figura 5. Dirección MAC predefinida del PIC y nombre de host.**

memoria de programa. Por ejemplo, para deshabilitar el DNS, la instrucción debe quedar comentada por medio del operador **// (ver Código 01 de la página 262).** El archivo **TCPIPConfig.h** está en la ruta **C:/Microchip Solutions Library/TCPIP Demo App/TCPIPConfig.h**.
En un primer contacto, es interesante recorrer este archivo, y notar las definiciones de dirección IP, MAC y demás parámetros asociados a la conexión del Web Server.

Funciones especiales del stack

Por medio de TCP/IP Configuration Wizard, podemos agregar funciones especiales. Una opción sumamente útil es hacer un **conversor RS232 - Ethernet.** De este modo, los paquetes que ingresen por el conector de red serán retransmitidos por la **UART** del **PIC**, y viceversa. Así, podremos darle conectividad **Ethernet** a sistemas que, originalmente, solo tenían comunicación **RS232**.
Vale destacar que la mayoría de las aplicaciones con microcontoladores utilizan el protocolo RS232 para tener comunicación con una PC.
Otra función interesante es medir el tiempo de viaje de un paquete entre el cliente y el host, lo que comúnmente denominamos **PING**. El stack nos devuelve el tiempo de viaje en milisegundos. Para activar este servicio, debemos marcar **[TCP Performance Test]** desde el TCP/IP Configuration Wizard.Vamos ahora a definir el **nombre de host**, es decir, uno con el cual identificar a nuestra placa en la red, de modo que no tengamos que recordar la dirección IP. Por otra parte, como la IP puede ser asignada por el servidor en forma dinámica, resulta conveniente definir un nombre de host que no varíe ante diferentes asignaciones.
En vinculación estrecha al nombre de host está la dirección **MAC**. Por norma, esta debe ser unívoca; esto quiere decir que existe un ente (en este caso, la IEEE) que asigna a los fabricantes direcciones únicas. Por lo tanto, al momento de comercializar nuestro desarrollo, tendremos que adquirir un bloque de direcciones MAC a la IEEE. Existen memorias de Microchip que tienen pregrabada una MAC única, que podemos vincular al microcontrolador a través de la interfaz **SPI** o **I2C**. Claro que para hacer pruebas, podemos recurrir a la MAC predefinida que trae el stack. ■

El nombre de host identifica la NIC para no tener que ingresar la dirección IP.



> FABRICACIÓN PROFESIONAL

Hay otros métodos para obtener un acabado más profesional en la construcción de circuitos impresos. Por ejemplo, en lugar de dibujar a mano el circuito, se realiza el diseño en una PC y se imprime en un papel vegetal con una impresora láser. Ese dibujo se pasará a la plaqueta virgen, por medio del planchado del papel impreso sobre la capa de cobre. De esa forma, la impresión quedará adherida al cobre, obteniendo un dibujo perfecto.

CONFIGURACIÓN DEL WEB SERVER

Vamos a definir el nombre de la página principal, la ubicación del sitio web al cual accederá el usuario final y la cantidad de conexiones simultáneas que permitirá la aplicación.

A continuación, vamos a establecer el nombre de la página principal, es decir, aquella que se mostrará por defecto; en este caso, es **index.htm**. Cada vez que coloquemos la IP de nuestro Web Server en el explorador, se cargará la página por defecto. En cambio, si queremos acceder a una página distinta, tendremos que ingresar: **http://192.168.1.103/variables.htm**, donde primero observamos la dirección IP y, luego, el nombre de la página a la cual estamos accediendo (**variables.htm**). Un parámetro relevante es la cantidad de conexiones que vamos a permitir, lo cual hace referencia a la cantidad de usuarios que pueden monitorear el sistema simultáneamente.
El stack permite un máximo de 32 conexiones a la vez, pero antes de establecer este valor, debemos considerar aspectos como el ancho de banda y la estabilidad de la red, ya que una excesiva demanda puede deteriorar el rendimiento del sistema.

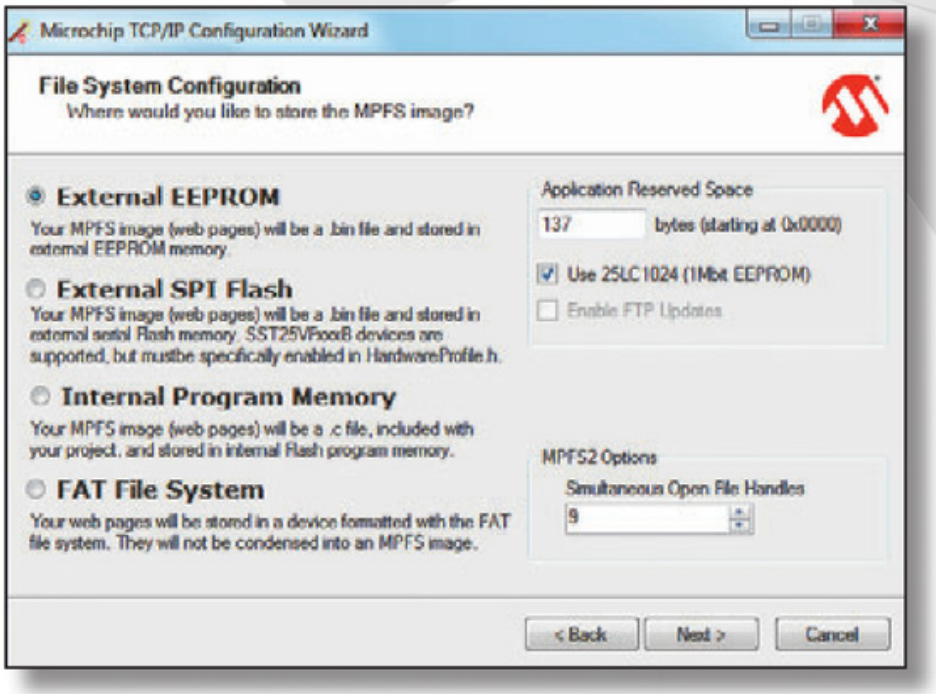
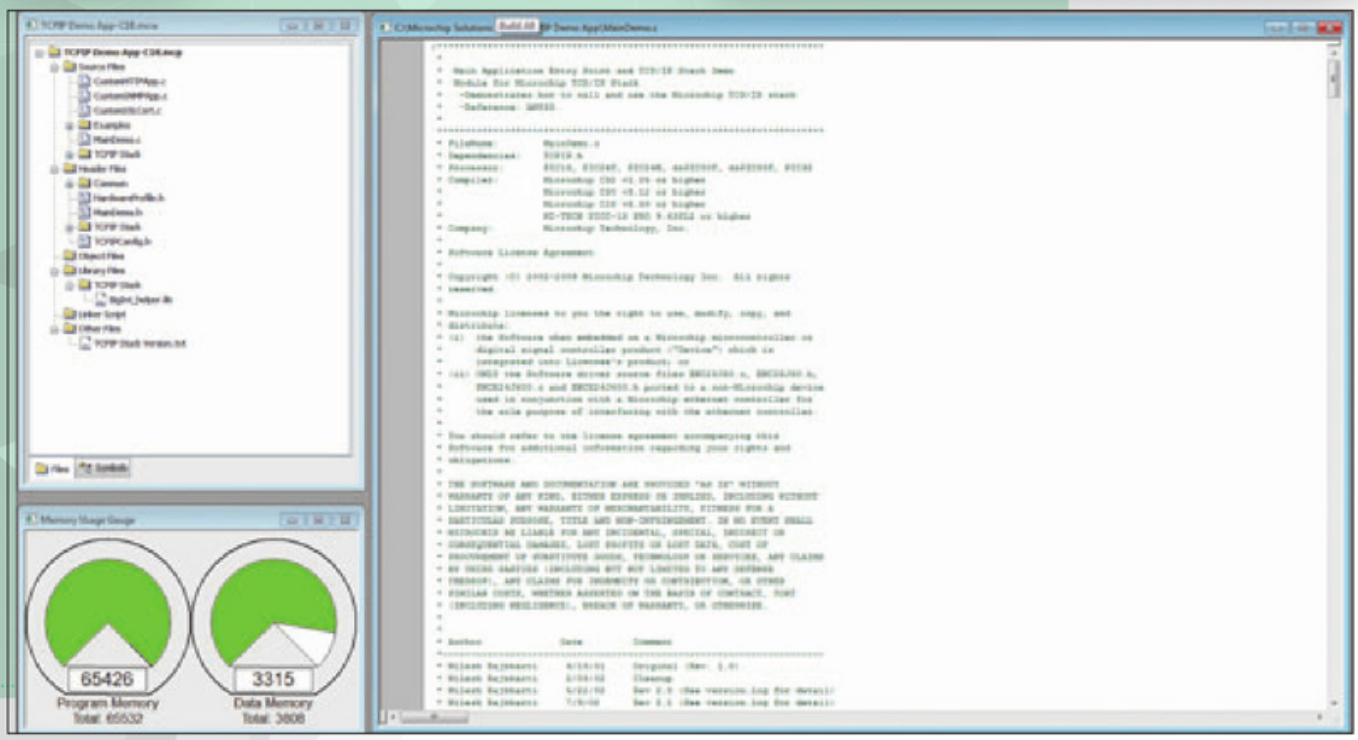


Figura 6. Opciones de almacenamiento para la página web.

Ubicación de la página web

Es posible establecer diferentes ubicaciones para almacenar la página web que verá el usuario final del **Web Server**. Esta puede encontrarse en la memoria de programa del microcontrolador o en una memoria externa del tipo **EEPROM**, a la cual se accede desde el PIC a través de SPI. Si bien en la **Figura 6 de esta misma página** vemos otras opciones, como memoria Flash y sistema FAT (permite almacenar la página en una tarjeta SD), las mencionadas anteriormente son las más representativas, porque determinan si la página está alojada en la memoria del microcontrolador o en una externa. En los ejemplos de este libro, vamos a utilizar la memoria externa de **Microchip**: **25LC1024**. Es del tipo EEPROM, tiene una capacidad de **1 Mbit** y se conecta al PIC por SPI. Al elegir una memoria externa, dispondremos de toda la capacidad del programa para escribir nuestro código y alojar el stack TCP/IP.

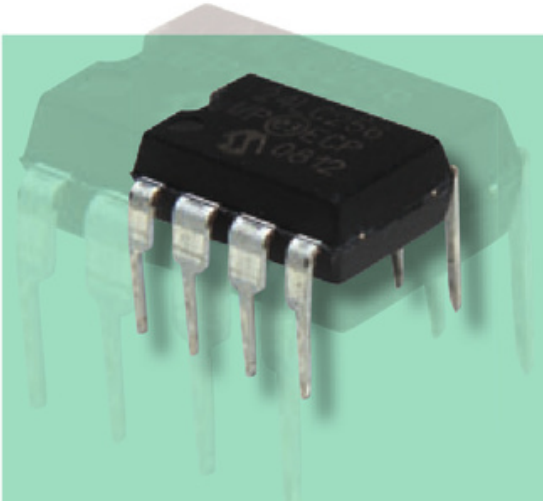


Compilar el stack

Una vez configurados los parámetros básicos de la aplicación a través del Microchip TCP/IP Configuration Wizard, estamos en condiciones de compilar el stack y cargarlo en el PIC. Para empezar, abrimos el proyecto **TCPIP Demo App-C18.mcp** desde el MPLAB, y ejecutamos la compilación haciendo clic en **[Build all]**. De este modo, estamos generando un archivo **HEX** listo para grabar en el PIC. El proceso de compilación se explicó en detalle en la **clase 12**. Dentro de la carpeta **[TCPIP Demo App]**, encontramos varios proyectos preparados para utilizar con micros de 8 a 32 bits. Es interesante destacar que los parámetros configurados con el asistente sirven para cualquiera, ya que solo se está modificando el archivo de cabecera **TCPIPConfig.h**. Seleccionamos el proyecto **TCPIP Demo App-C18.mcp** porque vamos a trabajar con un micro de la familia PIC18 (8 bits), como venimos haciendo a lo largo de la colección. En este caso, el micro es el PIC18F97J60, que incorpora el módulo Ethernet. La ruta predefinida para acceder al proyecto es: **C:/Microchip Solutions Library/TCPIP Demo App/TCPIP Demo App-C18.mcp**. Una vez compilado, precisamos un microcontrolador donde cargar el proyecto.

> Los parámetros configurados con el asistente sirven para cualquier proyecto.

Vamos a utilizar una placa comercial denominada **MCE WebServer DEV**, desarrollada por la empresa mc electronics (www.mcelectronics.com.ar). En este punto, la única finalidad es mostrar la capacidad del stack y todas sus funciones, motivo por el cual no veremos en detalle el hardware hasta la próxima clase. Simplemente, vamos a mencionar que la placa trae una memoria externa de 1 Mbit para alojar la web, un sensor de temperatura que utilizaremos en las prácticas y dos relays para controlar un motor.



> CUESTIÓN DE ESPACIO

Básicamente, tenemos dos opciones para almacenar la página web en el sistema: memoria externa o memoria interna del PIC. Debemos decidir por una u otra considerando el costo adicional que significa una EEPROM como la **25LC1024**. Otro parámetro importante es la capacidad de memoria del PIC, que, dependiendo de la aplicación, requerirá incluir una EEPROM externa.



> Figura 7. Página inicial index.htm, a la que se accede colocando la dirección IP asignada por el router.

> Figura 8. Página para configurar los parámetros de la red.



Acceder a la página web

El stack predefinido habilita al cliente DHCP, por lo que podemos conectar la placa a un router con **DHCP** habilitado (capacidad de asignar direcciones IP) y obtener conectividad desde una computadora en la misma red. Si no contamos con un router, es posible conectar el Web Server directamente a la placa de red de la computadora. En este caso, la IP es fija y se asigna en forma estática. La IP fija puede cambiarse desde el **TCP/IP Configuration Wizard** o desde el archivo **TCPIPConfig.h**. Vamos a conectar el servidor web a un router; en este caso, la IP asignada es **192.168.0.101**. Si lo conectamos a la PC, la IP cambia, y por eso, que el ejemplo sirve para ambos casos. Sobre la izquierda de la página inicial del **Web Server** aparece un menú con las distintas opciones de la placa (**Figura 8**), las cuales habilitamos a través del TCP/IP Configuration Wizard, como envío de e-mails, servidor HTTP y servidor seguro (nos permite acceder a una página protegida). Lo que nos interesa en este momento es el recuadro ubicado arriba a la derecha, que muestra en tiempo real el estado de **8 LEDs** y la temperatura enviada desde el sensor que tiene la placa (**Figura 7**). Podríamos, por ejemplo, monitorear otro LED o controlar un relay desde la interfaz web. Este tipo de interacción, en la que una variable reporta o modifica el estado de un puerto, se realiza mediante funciones **callback**, que veremos más adelante. Es interesante destacar que la actualización de los datos es automática y no requiere refrescar la pantalla del navegador. En la sección **[Configuración]**, encontramos parámetros como la dirección IP, el default gateway, la máscara de red, la MAC y el servidor DHCP (**Figura 8**). Todo esto puede cambiarse en tiempo de ejecución, por lo que es importante protegerlo mediante contraseña.

El stack predefinido habilita al cliente DHCP, por lo que podemos conectar la placa a un router.

También tenemos la posibilidad de enviar mensajes desde un formulario web, para lo cual se utiliza el protocolo **SMTP**. Simplemente, hay que completarlo con el nombre del servidor SMTP, el nombre de usuario y la clave. Por último, colocamos la dirección del destinatario y el cuerpo del mensaje.



> **Figura 09. Formulario para envío de e-mails en forma manual.**



> **Figura 10. Formulario de prueba para los métodos GET y POST.**

Por defecto, se adjunta un archivo **CSV** (se puede abrir con Excel) donde se muestra el estado de los LEDs, los pulsadores y el valor de temperatura. Más adelante veremos el código de este ejemplo. Antes de pasar a la programación del stack, analicemos una última pantalla de la interfaz web (**Figura 10**). Se trata del envío de variables a través de los métodos **GET** y **POST**. Quien esté acostumbrado a programar formularios web sabrá la ventaja que conlleva utilizar este recurso. Por ejemplo, con el método GET podemos enviar una serie de variables junto con la URL; es lo que habitualmente vemos como: **http://192.168.1.101/forms.htm?relay2=on&relay1=off**. Luego de la dirección IP de la placa, está el nombre del formulario que vamos a actualizar y, a continuación del signo de interrogación, el nombre de cada variable y su valor, separados por el signo **&**.

Podemos enviar un e-mail desde la interfaz web para reportar el estado de las variables.

Notemos que estamos actualizando el estado de dos variables (**relay1 y relay2**) de una sola vez. GET tiene una limitación, y es el hecho de que la URL no puede contener más de 1024 bytes. Si necesitamos enviar una mayor cantidad de variables, debemos utilizar el método POST, que no tiene restricciones de longitud. Si bien implementar este método en el stack TCP/IP es un poco más laborioso, suele hacerse para actualizar el texto de un LCD. Otra ventaja de este sistema radica en que la URL no se ve afectada, y el usuario final no ve las variables que estamos actualizando.

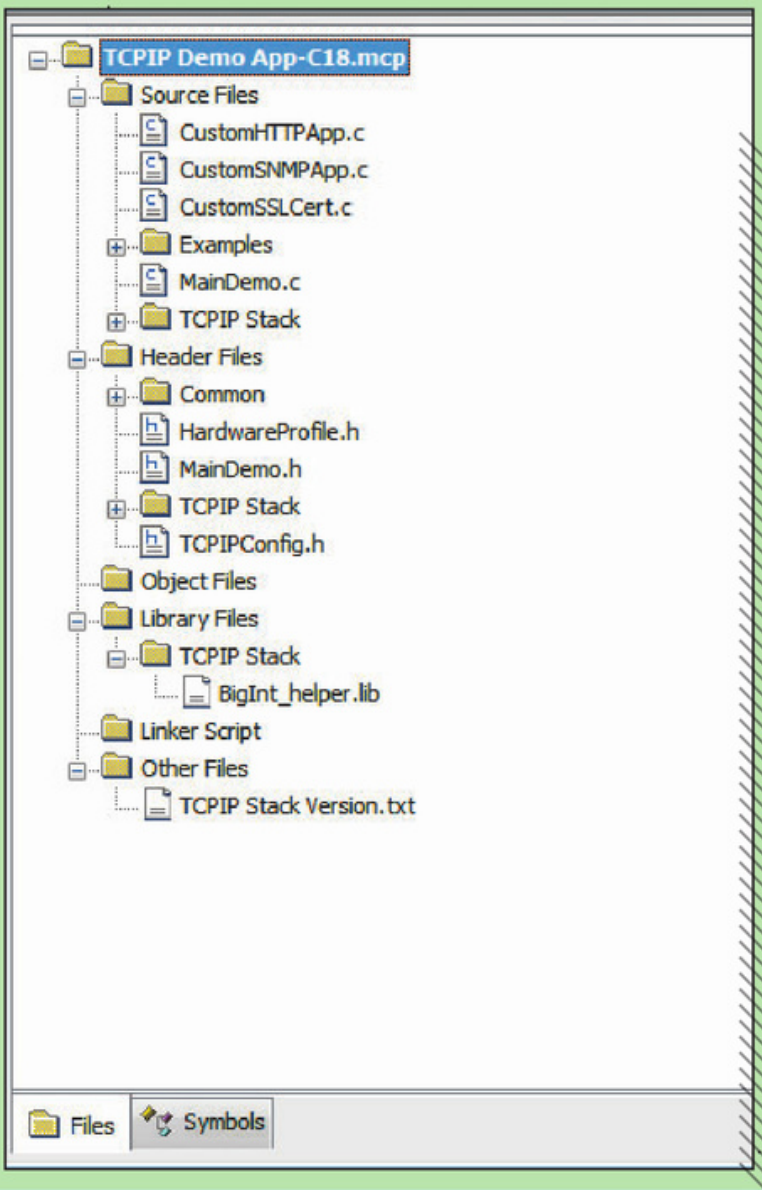


Los archivos principales del stack

Como mencionamos anteriormente, es necesario conocer el stack para poder agregar nuestras propias funciones y, así, lograr un sistema de telemetría y control remoto acorde con nuestras necesidades. Para este fin, vamos a desarrollar los archivos principales del stack. En la **Figura 11** vemos la estructura general del TCP/IP Stack en el MPLAB.
Se distingue la estructura dividida en archivos de cabecera (**.h**) y el código en los archivos (**.c**). Los archivos que necesariamente debemos conocer y que nos permiten personalizar nuestra aplicación son los siguientes:

> FUNCIONES CALLBACK

Se encuentran en el archivo CustomHTTPApp.c, dentro del proyecto del TCP/IP Stack. Son las funciones que nos permiten, por ejemplo, encender un LED, monitorear la temperatura o controlar un motor a través de Ethernet. Veremos que, necesariamente, debemos programarlas en el stack y vincularlas a través de variables dinámicas a la interfaz web.



> **Figura 11.** Estructura de archivos del stack.

- **HardwareProfile.h**: en este archivo, simplemente, asociamos un puerto con un nombre fácil de recordar. Por ejemplo, si colocamos un LED en el puerto RF3 del PIC (**Código 02**), estamos asociando el nombre **LED_IO** al puerto **RF3**. Entonces, cada vez que pongamos **LED_IO = 1**, el **port RF3** se pondrá en "uno lógico".
- **TCPIPConfig.h:** como vimos, se utiliza para habilitar o deshabilitar funciones del stack –como DNS, FTP, SNTP, SSL y UART–, dependiendo de las características de nuestra aplicación y de la memoria disponible en el micro.
- **MainDemo.c**: es el programa principal, en el que encontramos el mapeo de puertos, la configuración de los registros de la **USART**, el **ADC** (conversor analógico digital), la interfaz **SPI** y la inicialización de la placa. Por ejemplo, colocamos **ADCON0=0x0D**; (Registro ADCON0 del PIC) para seleccionar el pin **AN3** como entrada analógica.
- **CustomHTTPApp.c**: este es el cuarto archivo que necesitamos para una aplicación básica. En él se encuentran las funciones **callback**, es decir, aquellas que interactúan con la página web y reportan el estado de las variables dinámicas.

Funciones callback

Echemos un vistazo al archivo **CustomHTTPApp.c**, donde se encuentran las funciones callback. En este ejemplo, vamos a mostrar el estado de un LED en la página web por medio de **ON** y **OFF**.
La función **callback** contiene el

> **Código 02**

```
#define LED_TRIS  (TRISFbits.TRISF3)
#define LED_IO    (PORTFbits.RF3)
```

nombre y el valor de la variable dinámica (LED) que va a mostrarse en la web. Un error común es enviar directamente al buffer Ethernet el estado del LED mediante la variable **int LED_IO**. Pero hay que tener en cuenta que **solo pueden enviarse strings por HTTP**, por lo que, en numerosas ocasiones, es sumamente útil emplear las funciones **itoa** o **ftoa** (entero a ASCII y float a ASCII) para convertir los tipos de datos (**Código 03**).
En este caso, en vez de usar los conversores de tipo, creamos dos variables auxiliares, ESTADO_LED_UP y ESTADO_LED_DOWN,

La función callback contiene el nombre y el valor de la variable dinámica (LED) que se mostrará en la web.

> **Código 03**

```
ROM BYTE ESTADO_LED_UP[] = "ON";
ROM BYTE ESTADO_LED_DOWN[] = "OFF";

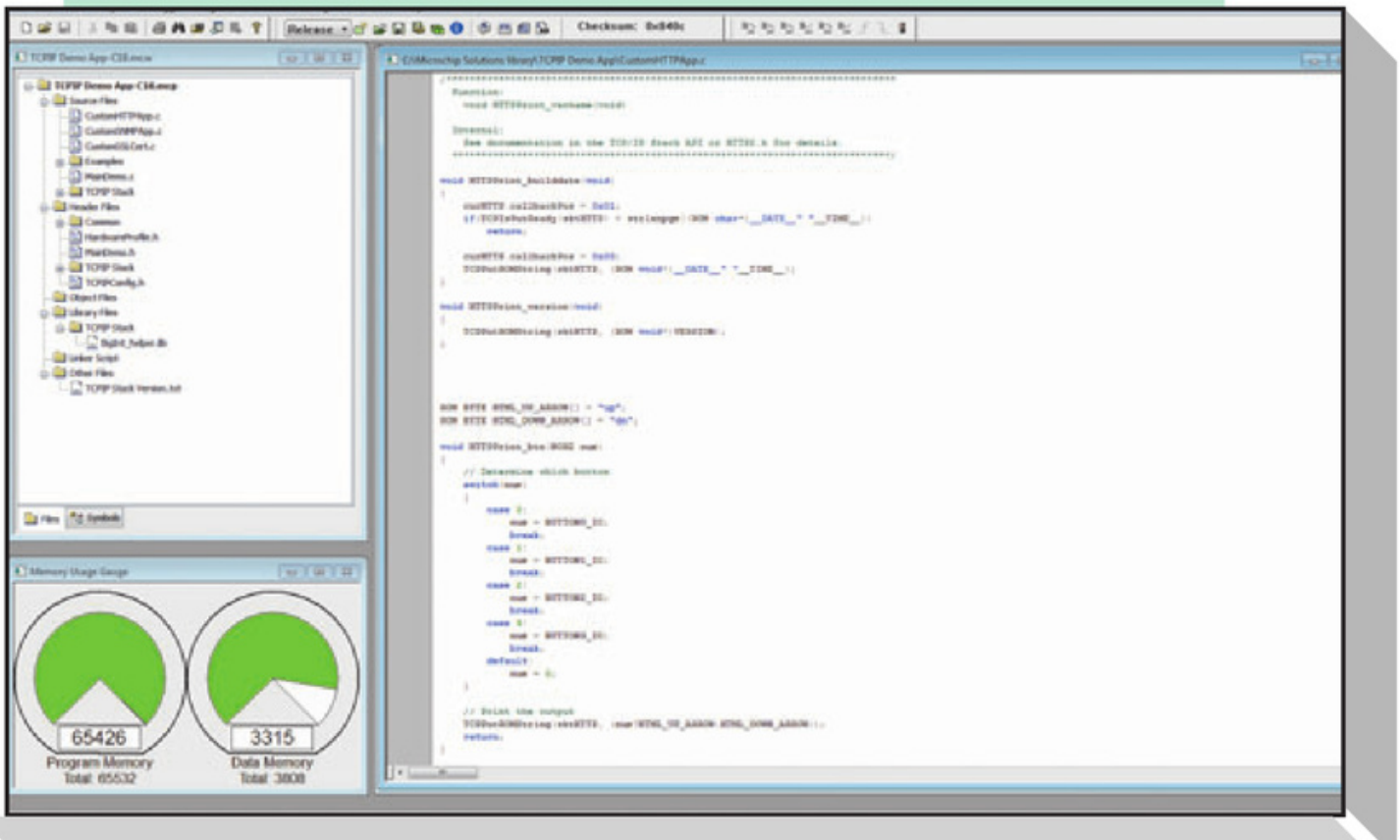
void HTTPPrint_LED(void)
{
 TCPPutROMString(sktHTTP, (LED_IO?ESTADO_LED_UP:ESTADO_LED_
DOWN));
}
```

> **Código 04**

```
LED_IO?ESTADO_LED_UP:ESTADO_LED_DOWN
```

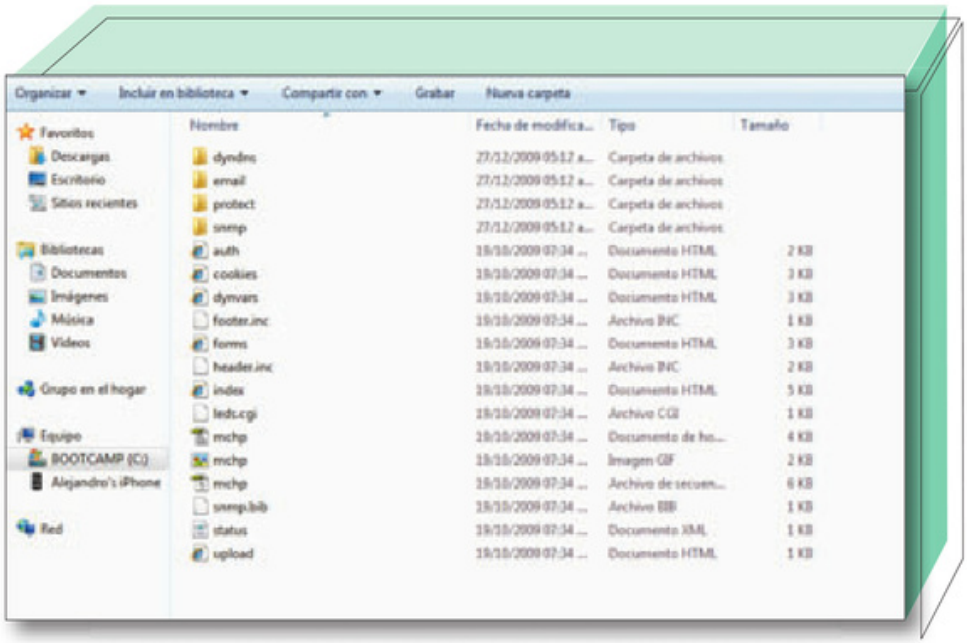
para mostrar el estado del LED mediante ON y OFF.
La función **void HTTPPrint_LED(void)** envía a la página web el valor de la variable dinámica **LED**. Para verificar el estado del puerto RF3 (donde está mapeado el LED), utilizamos la sentencia condicional IF, aquí representada en una línea (**Código 04**).
Recordemos que, anteriormente, definimos **LED_IO** como el estado del puerto **RF3**. El **IF** revisa el estado de **LED_IO**. Si el puerto está en 1, entonces la sentencia es verdadera y envía, por **HTTP**, **ESTADO_LED_UP**; en cambio, si está en cero, la sentencia es falsa y envía **ESTADO_LED_DOWN**.
De esta forma, tenemos un sistema básico que reporta el estado de las variables ante un llamado del servidor HTTP. Ahora debemos diseñar una página web que muestre las variables al usuario final.

> **Figura 11. Vemos una parte del archivo CustomHTTPApp.c, donde se definen las funciones callback.**



Programación de la página web

Microchip organiza los archivos del sitio web de modo de ahorrar espacio. Se trata de no repetir la información. Por ejemplo, al pie de todas las páginas aparece la misma leyenda. Entonces, utilizamos el archivo **footer.inc** que será llamado desde cada página, en vez de escribir el texto otra vez. Esto permite reducir sensiblemente el tamaño total del sitio; recordemos que tenemos solo 1 Mbit para almacenar la web.
En el sitio hay que incluir la variable dinámica **LED**, para lo cual tenemos que modificar los archivos **status.xml** e **index.htm**. La ruta predefinida para acceder a los archivos que componen el sitio es **[C:/Microchip Solutions Library/TCPIP Demo App/WebPages2/]**.
Por un lado, **status.xml** contiene todas las variables dinámicas que se actualizan (LEDs, temperatura y pulsadores). Agregamos la línea correspondiente a la nueva variable dinámica **(Código 05)**.
Entre los signos **< >** se indica el nombre del divisor que se utilizará en el index para definir la posición de la variable.
Por su parte, Index.html es el archivo que ve el usuario final. Donde aparezca **<"LED">**, el Web Server lo reemplazará con **ON** u **OFF** según el valor de **LED_IO (Código 06)**.
Es importante destacar que este ejemplo utiliza **AJAX** para actualizar la información sin tener que presionar la tecla **<F5>** en el explorador. Debemos agregar la siguiente sentencia al final del archivo **index.htm** (**Código 07**).
Lo que hicimos hasta ahora fue crear una **variable dinámica LED** en el código del stack y enviarla a la web a través de una función **callback.** Luego, configuramos los archivos **index.htm** y **status.xml** para visualizar el estado de la variable en la página.



> **Figura 13. Estructura de archivos que componen el sitio web.**

> **Código 05**

```
<LED>~LED~</LED>
```

> **Código 06**

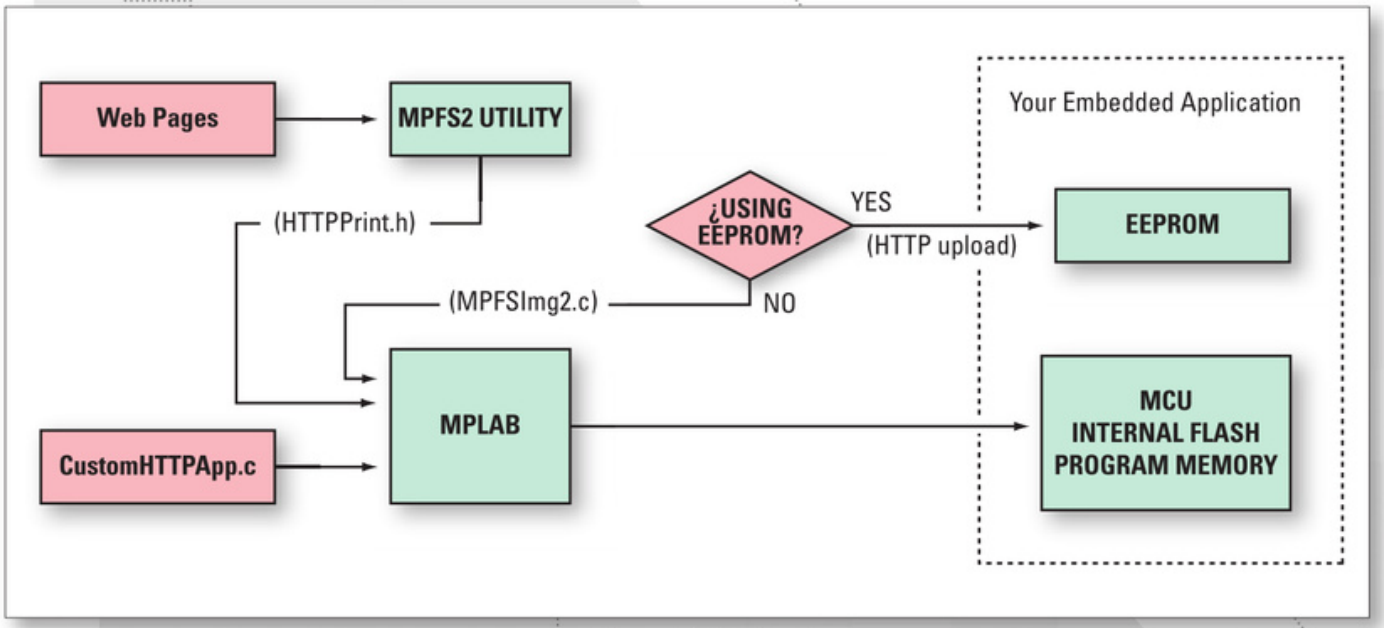
```
<span id="LED">?</span>;
```

> **Código 07**

```
document.getElementById('LED').innerHTML = getXMLValue(xmlData, 'LED');
```

> SOBRE AJAX

AJAX significa Asynchronous JavaScript And XML.
Es una técnica de desarrollo web destinada a crear aplicaciones interactivas que se ejecutan en el navegador del cliente mientras se mantiene la conexión asíncrona con el servidor. Permite actualizar la información de la web sin necesidad de refrescar la pantalla, y es soportado por la mayoría de los navegadores.



> **Figura 14. En este diagrama vemos las distintas opciones para almacenar la página.**

Sistema de archivos MPFS

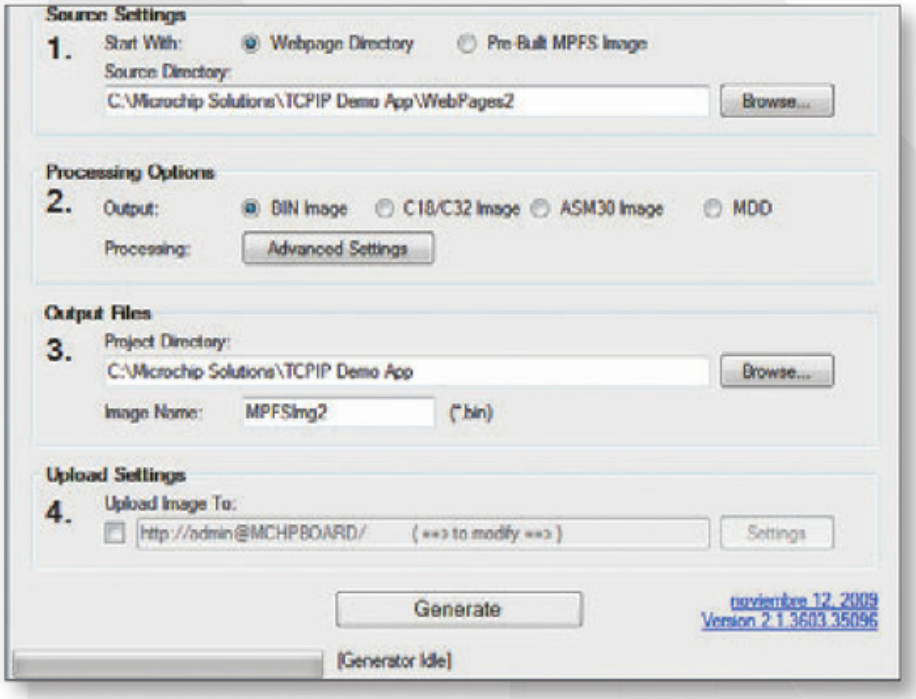
Ahora necesitamos cargar la web en la memoria externa **25LC1024**. Antes de hacerlo, debemos compactar los archivos que componen el sitio mediante la aplicación MPFS (Microchip PIC File System), que se instala con el stack y se encuentra en la ruta que venimos trabajando. El acceso directo desde el menú **Inicio** es **Microchip/TCPIP Stack – MPFS2**. MPFS es el sistema de archivos que utiliza el micro para leer la página web. Una vez que tenemos nuestro sitio terminado con los archivos XML, HTML, JPG y GIF, debemos convertirlos al sistema **MPFS** para, luego, poder cargarlos en el micro o en la memoria externa. Lo que hace internamente este conversor es juntar todos los archivos en uno solo; de hecho, podemos abrir el resultante con el Bloc de notas y veremos, ordenado en forma secuencial, el contenido de cada archivo que compone el sitio.

La aplicación **MPFS.exe** convierte el conjunto de archivos que integran la página web a dos formatos posibles: **.bin** para cargar el website en la memoria EEPROM externa y **.c** para hacerlo en la memoria de programa del PIC.

Como mencionamos anteriormente, al cargar la web en la memoria externa, toda la memoria de programa del PIC queda disponible para la aplicación. Si bien esta opción es más costosa (porque requiere una memoria externa), priorizamos la disponibilidad de la memoria interna para escribir las funciones callback y demás código necesario.

El MPFS nos da la opción de transferir la web automáticamente a la memoria externa luego de compactarla. Para hacerlo, la placa debe estar conectada a la red (o la PC) y tener cargada la última versión del Microchip TCP/IP Stack.

> **Figura 15. Pantalla principal del MPFS.exe, con las opciones requeridas para compactar la página y cargarla en la memoria.**



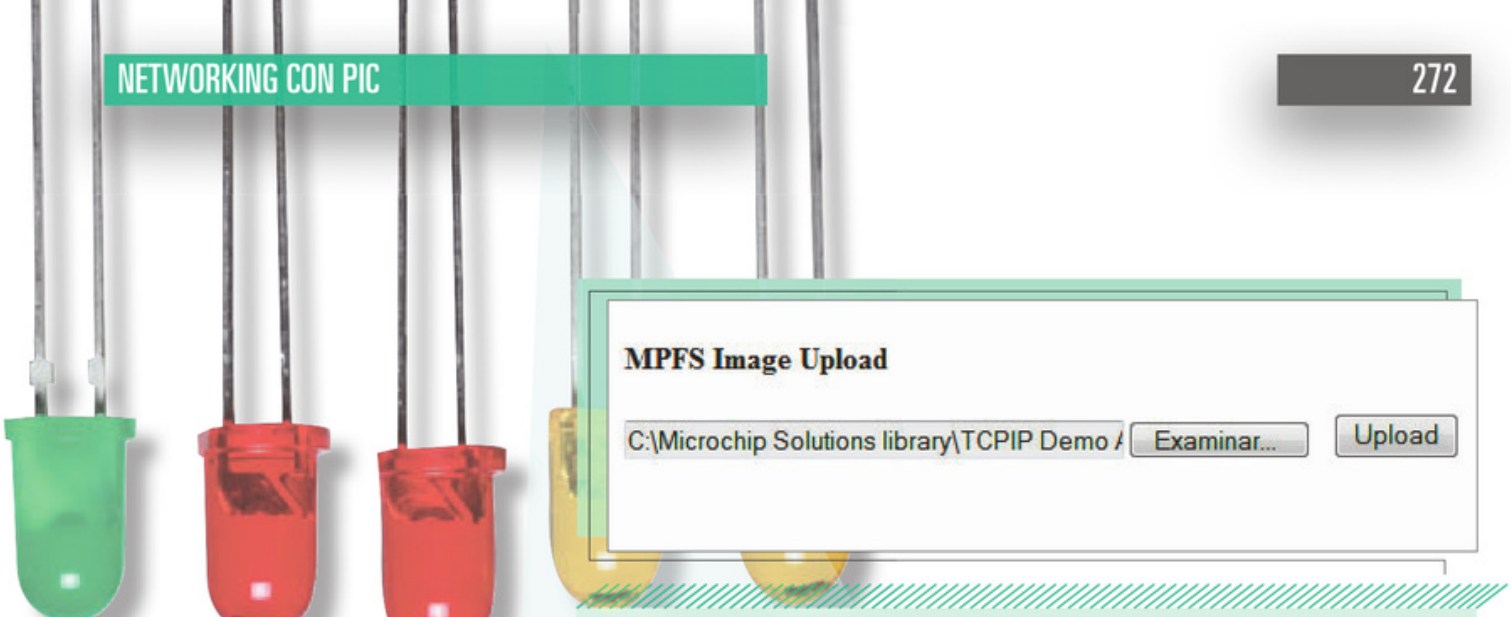


Figura 16. Servicio para cargar el web site en forma remota, con el cual debemos incluir el archivo .bin resultante de la compactación.

Debemos tener en cuenta que cada variable que agreguemos en la web requiere su correspondiente variable en el stack. Por lo tanto, es necesario recompilar el stack y cargarlo en el PIC antes de compactar la página y transferirla a la memoria.
La secuencia correcta es la siguiente:

- Modificar el stack según las necesidades de telemetría requeridas.
- Agregar las variables dinámicas correspondientes en status.xml e index.htm.
- Compactar la web con MPFS.
- Compilar el stack con el MPLAB.
- Cargar el stack en el PIC.
- Transferir la página a la memoria externa.

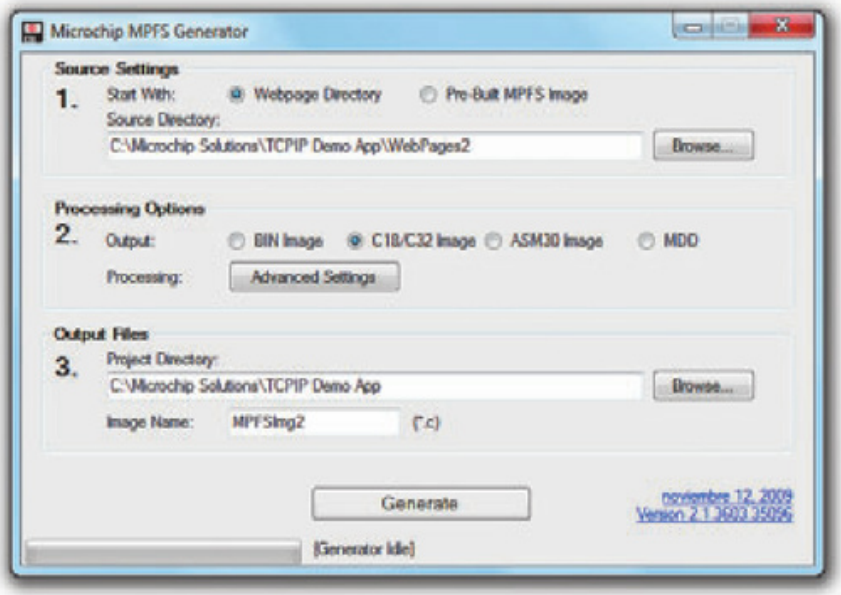
Es preciso respetar este procedimiento porque, si las variables de la web no se corresponden con las del stack, la página no se cargará en el explorador.
Por lo general, lleva mucho tiempo darse cuenta de este error.
Tenemos dos alternativas para cargar la web en la memoria externa: a través del MPFS o en forma remota por medio del explorador. Este último es un sistema de transferencia de archivos por medio de HTTP, para lo cual debemos ingresar en **http://192.168.1.101/mpfsupload.**
Las aplicaciones más seguras no permiten transferir la web a la memoria externa por este medio, porque cualquiera podría acceder a la dirección mencionada conociendo solo la IP, y así modificar la página. En este caso, debemos proteger el sistema de carga mediante contraseña o bien almacenar la web en la memoria de programa del PIC. Hay que colocar la menor cantidad posible de imágenes en la aplicación final, dado que la memoria **2 1024** posee **1 Mbit** de espacio, y si nuestra web ocupa más, no se cargará correctamente.

> HTTPPRINT.H

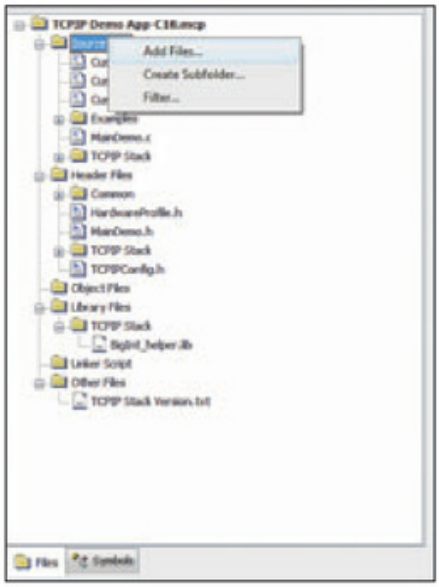
Es importante tener en cuenta que, al agregar o quitar variables de nuestra web (por medio de status.xml), se modifica el archivo HTTPPrint.h, por lo que hay que volver a compilar el stack antes de cargar la aplicación en el micro. Es recomendable seguir la secuencia mencionada en esta página para modificar, compilar y, finalmente, cargar el stack en el PIC.

Cargar la web en la memoria interna del PIC

En aplicaciones que deben ser extremadamente económicas o en las que la seguridad es un factor prioritario, podemos optar por almacenar la web en la memoria de programa del microcontrolador.
Para hacerlo, ejecutamos el MPFS y seleccionamos **C18/C32 Image** desde el menú principal.
Esto generará un archivo **.c** que debemos agregar al stack para que sea compilado con el resto del código. Por defecto, el nombre del archivo es **MPFSImg2.c** (es decir, imagen de la web en Microchip PIC File System).
El número 2 hace referencia a que el stack TCP/IP incorpora una versión mejorada del MPFS.
Para agregar la web –es decir, el archivo MPFSImg2.c–, debemos seguir dos pasos.
Primero, comentamos la sentencia **#define MPFS_USE_EEPROM** dentro del archivo **TCPIPConfig.h**.
De este modo, estamos indicando que la web no estará en la memoria EEPROM externa. Segundo, incluimos el archivo **MPFSImg2.c** en el proyecto, para lo cual hacemos clic con el botón derecho del mouse sobre **Source Files** y seleccionamos **Add Files**, como se muestra en la **Figura 18 de la página 273**.
De esta forma, prescindiremos de la memoria **EEPROM** externa, con el consecuente ahorro de dinero y espacio físico en la placa.
Ya tenemos la web incluida en el stack, ahora debemos compilar el



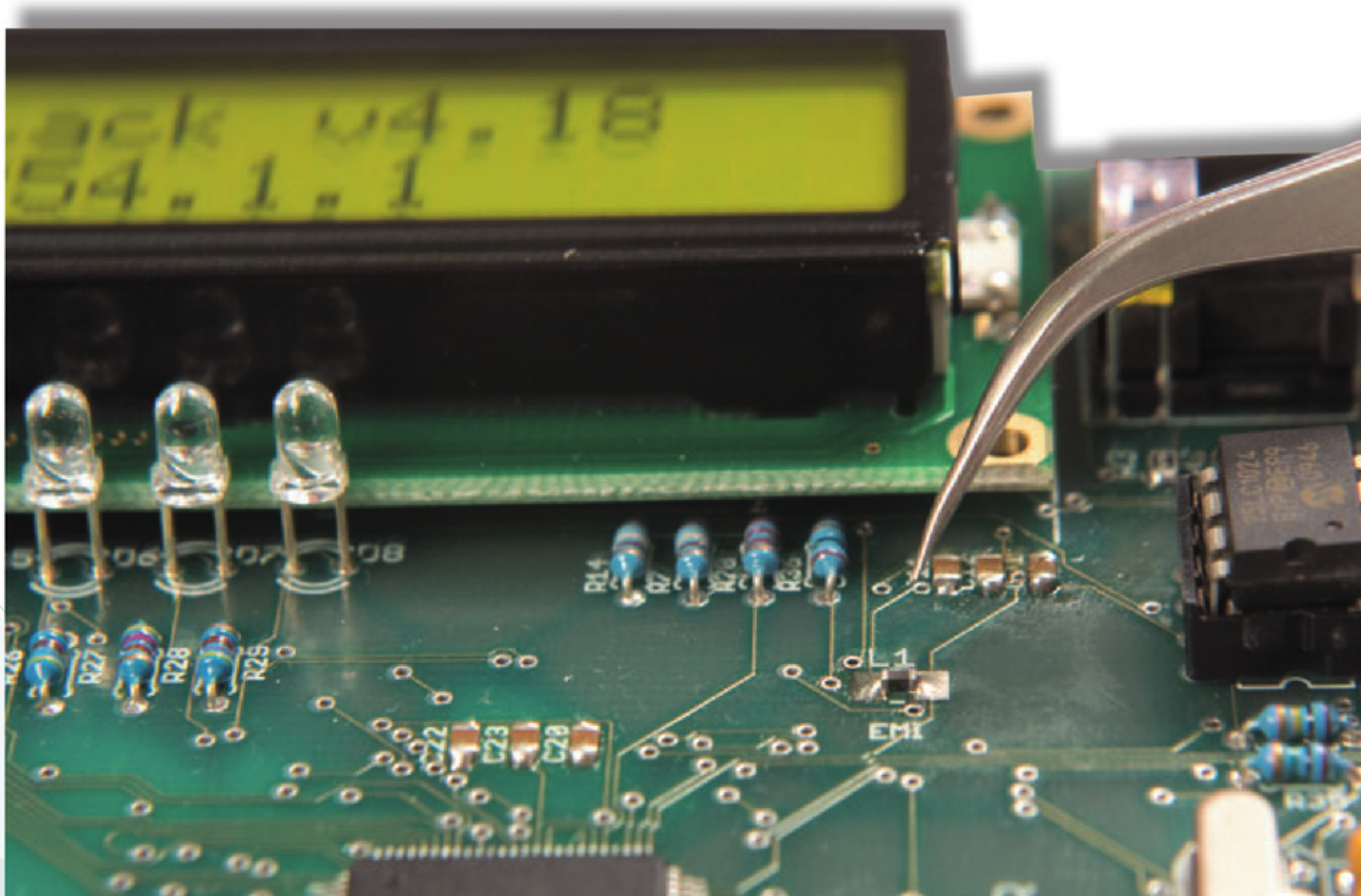
> Figura 17. MPFS configurado para generar un archivo .c.



> Figura 18. Cómo agregar el archivo MPFSImg2.c al proyecto.

proyecto completo y cargarlo en el PIC. Una vez que compilamos, podemos notar que queda mucho menos espacio disponible en la memoria de programa.

Una página promedio ocupa cerca del 25% de la memoria de un **PIC18F97J60**. Este es el PIC que está en la placa **MCE WebServer DEV** y que vamos a estudiar en detalle en la próxima clase. Por lo tanto, en caso de que necesitemos almacenar la web de esta forma, es crítico reducir la cantidad de imágenes y minimizar su resolución al máximo posible.



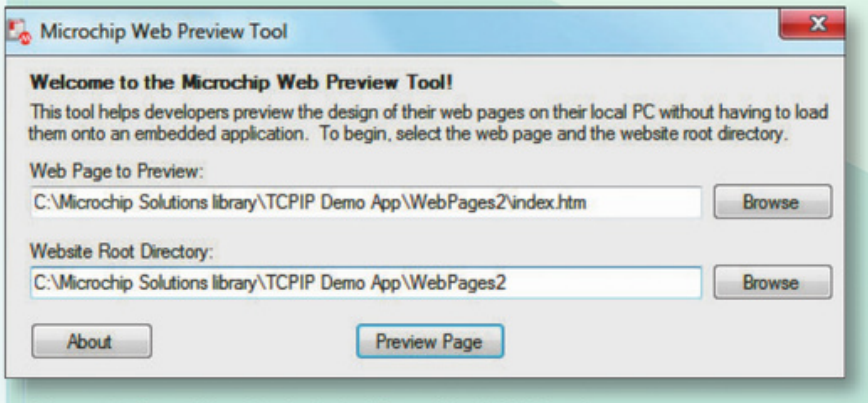
Microchip Web Preview

Esta aplicación permite visualizar el aspecto que tendrá la página antes de cargarla en el sistema, y resulta de gran utilidad porque, de otra forma, deberíamos compactarla y transferirla a la memoria (ya sea del PIC o la memoria externa) cada vez que modifiquemos algún parámetro estético. Microchip WebPreview se encuentra dentro de la carpeta del stack en el menú **Inicio**.
En primer lugar, ingresamos la ruta del archivo **index.htm** y la **ubicación** de la carpeta donde están las imágenes y demás archivos que componen el sitio. Respectivamente, son:

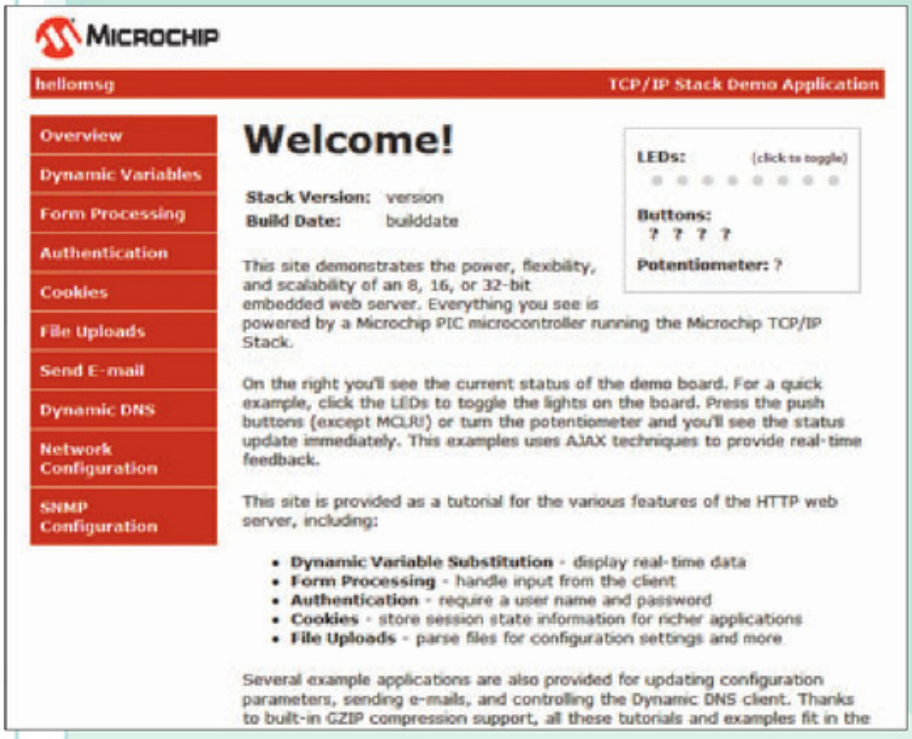
[C:\Microchip Solutions library\TCPIP Demo App\WebPages2\index.htm]
[C:\Microchip Solutions library\TCPIP Demo App\WebPages2]

Web Preview permite previsualizar el aspecto que tendrá la página web desde la PC.

Luego, hacemos clic en **Preview Page** y veremos la página generada por la aplicación en una ventana de Windows. Notemos que las variables, como las que indican el estado de los pulsadores, aparecen con **(?)**. Esto es lógico, pues la aplicación no está cargada en el micro y, por lo tanto, no hay una función callback que devuelva esos valores. Tampoco tenemos posibilidad de interacción con los botones ni vemos el valor del sensor de temperatura.
Si bien el Microchip Web Preview es útil para visualizar el aspecto de nuestra página, necesariamente debemos compilar el proyecto y grabarlo en el microcontrolador para, luego, transferir la web a la memoria. De este modo, verificamos que todas las funciones callback estén funcionando como corresponde. Además, debemos asegurarnos de que las variables dinámicas responden de manera adecuada. Por último, recordemos que siempre es necesario recompilar el stack si agregamos o quitamos variables dinámicas de la página web. ■



> **Figura 19. Pantalla principal de Microchip Web Preview.**



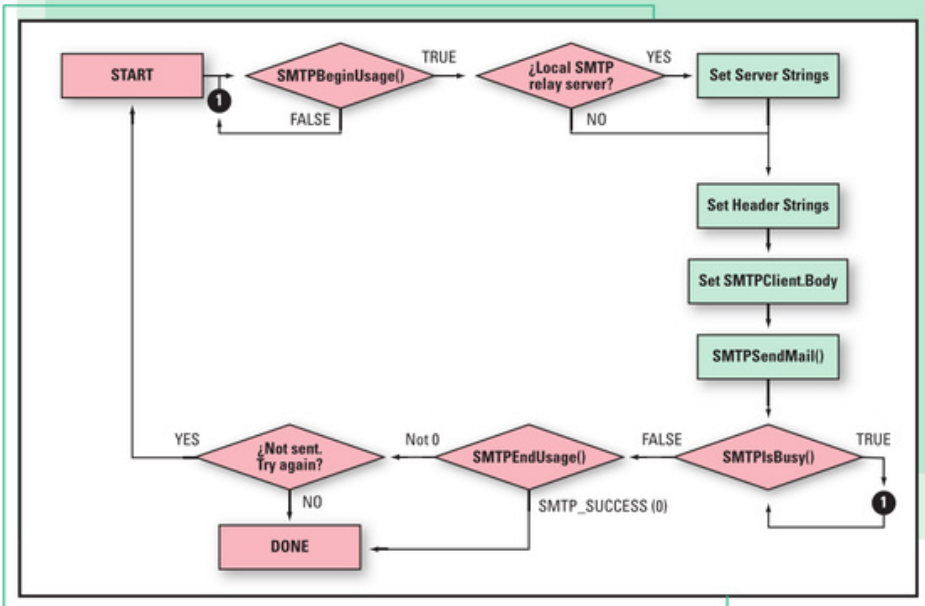
> **Figura 20. Página generada con el Web Preview.**

ENVÍO DE E-MAILS



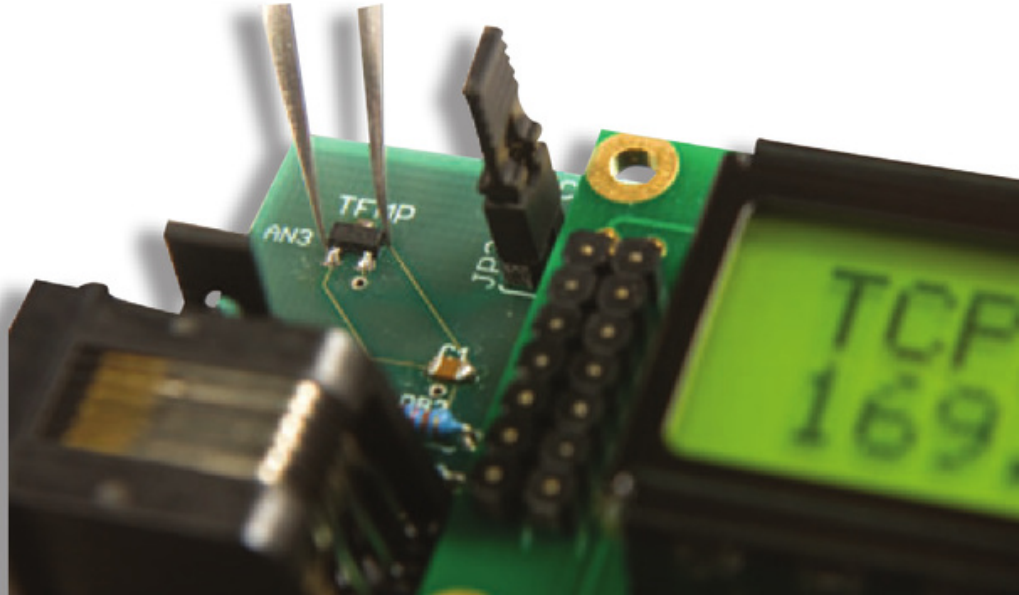
Vamos a trabajar con el protocolo SMTP para enviar un mensaje automáticamente desde el stack. Esto es de gran utilidad para reportar eventos.

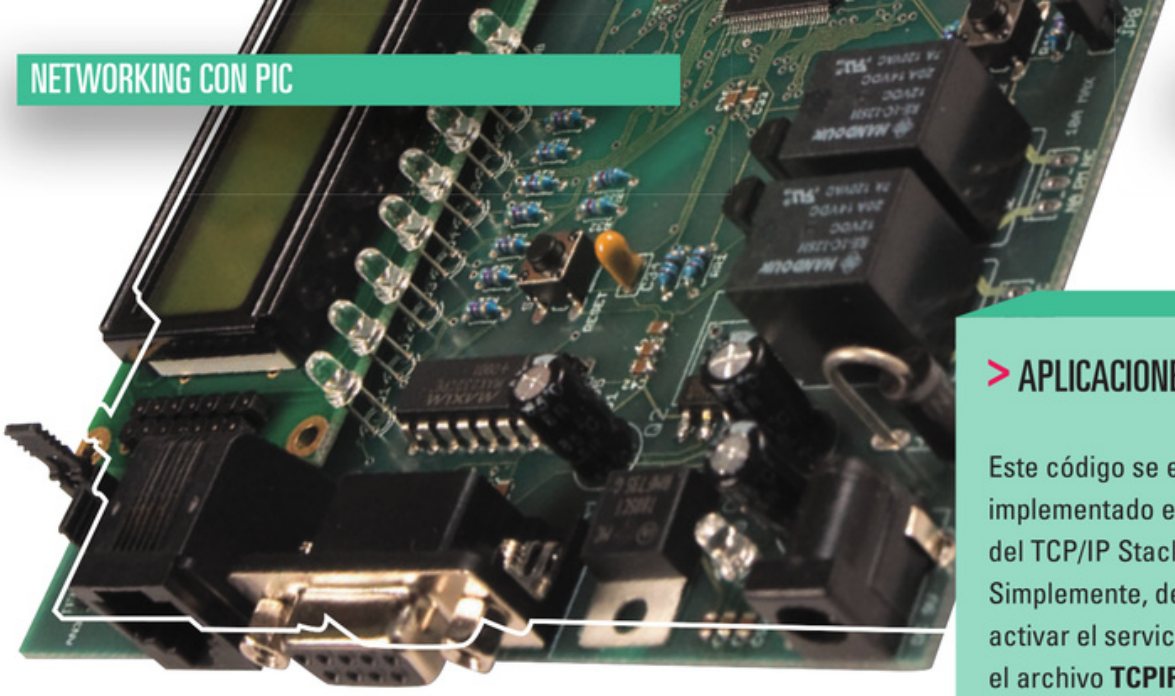
Para enviar e-mails desde el stack TCPIP debemos habilitar el cliente **SMTP**, dejando de comentar la sentencia **STACK_USE_SMTP_CLIENT** en el archivo **TCPIPConfig.h**. Las funciones requeridas agregan aproximadamente 1 K de código a la memoria de programa.
El proceso puede invocarse en cualquier momento, por ejemplo, si sube la temperatura, si se presiona un pulsador, etc. El e-mail de este ejemplo envía el estado de los pulsadores a la dirección que se especifique. Veamos un esquema que representa el funcionamiento del cliente SMTP. Antes de intentar enviar un e-mail con la función **SMTPSendMail()**, tenemos que verificar si no hay otro en proceso, para lo cual utilizamos, en primer término, **SMTPBeginUsage()**, como indica el diagrama. Si nuestro servidor de correo saliente requiere autorización, usamos **SMTPClient.Server, SMTPClient.Username** y **SMTPClient.Password** para configurar los parámetros. Recordemos que esto lo hicimos cuando enviamos un e-mail desde el formulario generado por el stack. Por último, incluimos los campos De, Para, Asunto y Mensaje antes de llamar a **SMTPSendMail()**. Mientras se intenta enviar el mensaje, estamos en el bucle de la función **SMTPIsBusy()**. En este punto, si el e-mail se envió



> **Figura 21 . Funciones del stack TCP/IP para envío de e-mails.**

correctamente, **SMTPEndUsage()** devolverá **0**; de lo contrario, tendremos la opción de comenzar el proceso otra vez o cancelar el envío. Un e-mail puede no enviarse debido a fallas en la conexión o porque los datos de autentificación son incorrectos. Si ingresamos erróneamente la dirección del destinatario, el stack lo mandará de todas formas.





Ejemplo completo

Veamos el proceso de envío con un ejemplo concreto (**Código 08**). En este caso, utilizamos **SMTPClient.Username** y **SMTPClient.Password** para loguearnos en el servidor seguro. En negrita aparecen los parámetros como el destinatario, los datos del servidor y el asunto del mensaje. Las variables que indican el estado de los pulsadores se llaman **BUTTON?_IO,** donde (**?**) representa el número de pulsador. Como dijimos anteriormente, podemos invocar este código dentro del programa principal. Por ejemplo, es posible hacer un **IF** que compare el valor de temperatura con un umbral y, en caso de superarlo, envíe un mensaje de alerta a la dirección especificada por **RAMStringTo[].** ■

> APLICACIONES DEL SMTP

Este código se encuentra implementado en el main del TCP/IP Stack. Simplemente, debemos activar el servicio SMTP desde el archivo **TCPIPConfig.h**; e ingresar los datos del destinatario, los de nuestra cuenta de e-mail y el cuerpo del mensaje. Podemos hacer una alarma que compare el valor de temperatura del sensor con un umbral, y envíe un e-mail si el valor está tantos grados por encima.

> Código 08

```
if(SMTPBeginUsage())
{
        static BYTE RAMStringTo[] = "soporte@mcelectronics.com.ar";
        static BYTE RAMStringBody[] = "Mensaje generado por el MCE WebServer" VERSION " \r\n\r\nBotones: ";
        RAMStringBody[sizeof(RAMStringBody)-2] = '0' + BUTTON0_IO;
        RAMStringBody[sizeof(RAMStringBody)-3] = '0' + BUTTON1_IO;
        RAMStringBody[sizeof(RAMStringBody)-4] = '0' + BUTTON2_IO;
        RAMStringBody[sizeof(RAMStringBody)-5] = '0' + BUTTON3_IO;
        SMTPClient.Server.szROM = (ROM BYTE*)"mail.mcelectronics.com.ar";
        SMTPClient.ROMPointers.Server = 1;
        SMTPClient.Username.szROM = (ROM BYTE*)"info@mcelectronics.com.ar";
        SMTPClient.ROMPointers.Username = 1;
        SMTPClient.Password.szROM = (ROM BYTE*)"123456";
        SMTPClient.ROMPointers.Password = 1;
        SMTPClient.To.szRAM = RAMStringTo;
        SMTPClient.From.szROM = (ROM BYTE*)"\"SMTP Service\" <info@mcelectronics.com.ar>";
        SMTPClient.ROMPointers.From = 1;
        SMTPClient.Subject.szROM = (ROM BYTE*)"Mensaje de prueba";
        SMTPClient.ROMPointers.Subject = 1;
        SMTPClient.Body.szRAM = RAMStringBody;
        SMTPSendMail();
        MailState++;
}
```

PUENTE SERIE-ETHERNET



Mediante el TCP/IP Stack podemos realizar un puente entre sistemas que utilicen el protocolo RS232 y Ethernet, con mínimos cambios en la configuración.

Mediante este puente podemos actualizar sistemas que originalmente no tenían conexión a **Ethernet**. Por ejemplo, podemos monitorear celdas de comunicaciones no atendidas desde Internet.

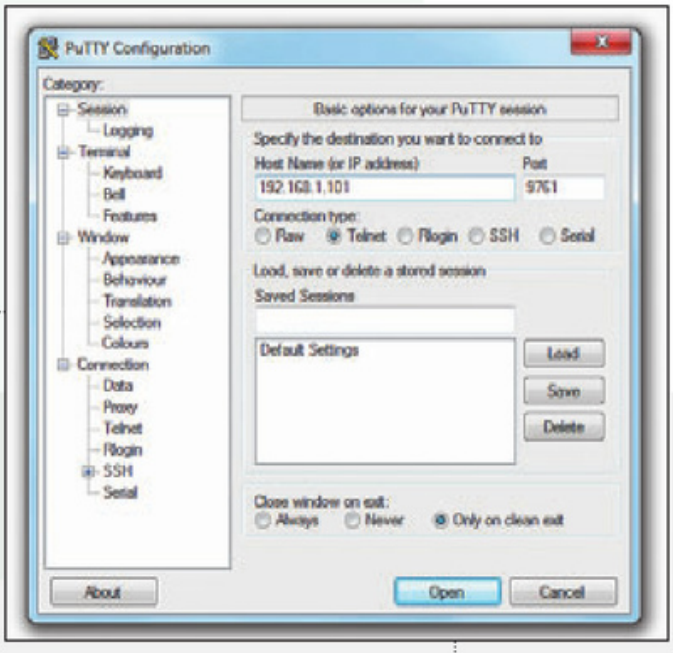
Como la celda reporta su estado a través del puerto serie (utilizando RS232), es posible conectarle un Web Server configurado como puente serie-Ethernet y, así, monitorear y controlar la celda a través de la Red.

Vemos en la **Figura 23** cómo activar la función **Serial to Ethernet Bridge** desde el TCP/IP Configuration Wizard. Este ejemplo acepta conexiones en el **puerto 9761**. Una vez que establecemos la comunicación a través de un cliente **Telnet**, todos los datos recibidos serán retransmitidos por la **UART** del **PIC**, y los datos recibidos desde la UART serán enviados al cliente por Ethernet.

De esta forma, siguiendo el ejemplo anterior, conectamos el Web Server al puerto serie de la celda de comunicaciones y podemos monitorearlo desde un cliente Telnet a través de Internet. Recordemos que el **cliente Telnet** es un programa que nos permite realizar conexiones a determinados equipos, como el Web Server; solo necesitamos ingresar la dirección IP y el número de puerto.

El cliente **puTTY** se descarga desde **www.putty.org**. Es un archivo **.exe**, que simplemente se ejecuta y en el cual debemos ingresar la **IP** del **Web Server** y el puerto al cual queremos acceder, en este caso, el **9761**. Una vez establecida la comunicación con el Web Server, nos pide usuario y contraseñaque, por defecto son admin y microchip, respectivamente. Ahora estamos en condiciones de recibir los datos desde el puerto serie; de hecho, podemos conectar un sistema que envíe el estado de un sensor por **RS232** y lo veremos en la pantalla del **puTTY** a través de Ethernet. También es posible enviar comandos mediante el puTTY y controlar desde Ethernet el dispositivo serie. ■

> **Figura 22. Cliente Telnet PuTTY configurado para conectarse al puerto 9761 del Web Server.**



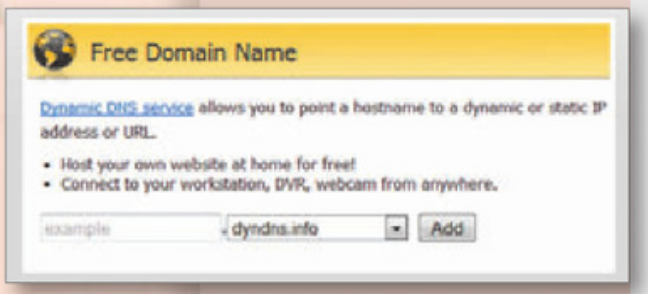


> **Figura 23: Seleccionamos la opción para habilitar el puente serie-Ethernet y así poder controlar la celda desde Internet.**

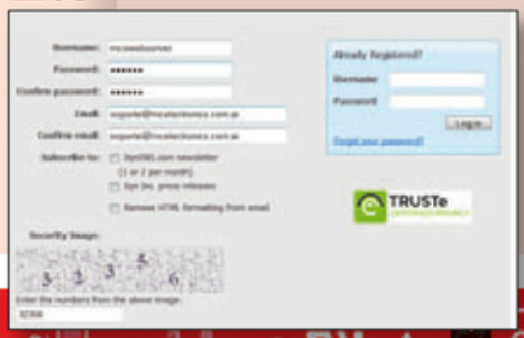
ACCESO DESDE INTERNET

¿Qué sucede cuando queremos acceder a nuestro Web Server desde Internet para monitorear el estado de un puerto o realizar un control remoto? Veamos este tema en detalle.

01



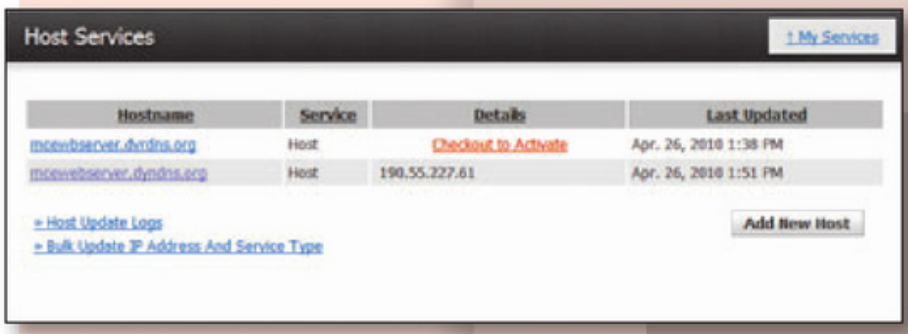
03



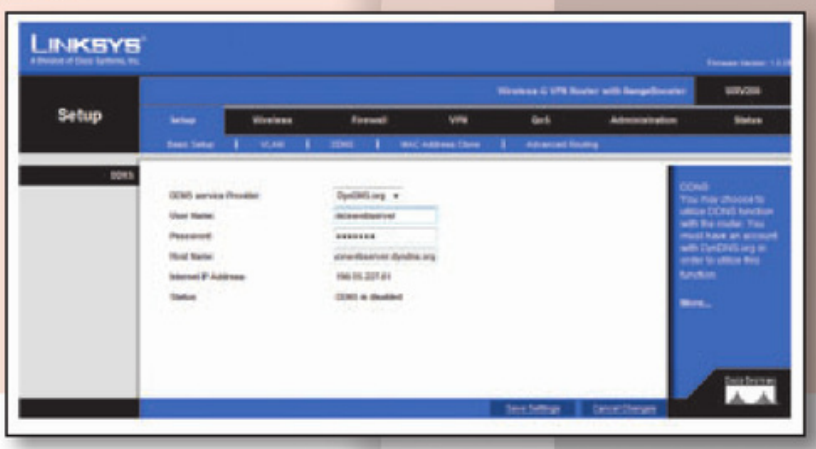


Una de las preguntas más recurrentes es cómo acceder al Web Server con PIC a través de Internet. Si bien podemos hacerlo desde la red local mediante la dirección IP privada (por ejemplo, 192.168.1.101), esto no es posible desde Internet, a menos que contratemos una IP fija a nuestro ISP. Para la mayoría de los usuarios, la IP es dinámica, es decir, cambia dentro de un rango cuando nos conectamos. Para solucionar este inconveniente, debemos asociarle lo que se denomina DDNS (Dynamic DNS) o DNS dinámico, el cual aplica un nombre de dominio a nuestra IP pública, como el siguiente: www.mcewebserver.com.ar. De este modo, siempre tendremos acceso al servidor web desde Internet. ■

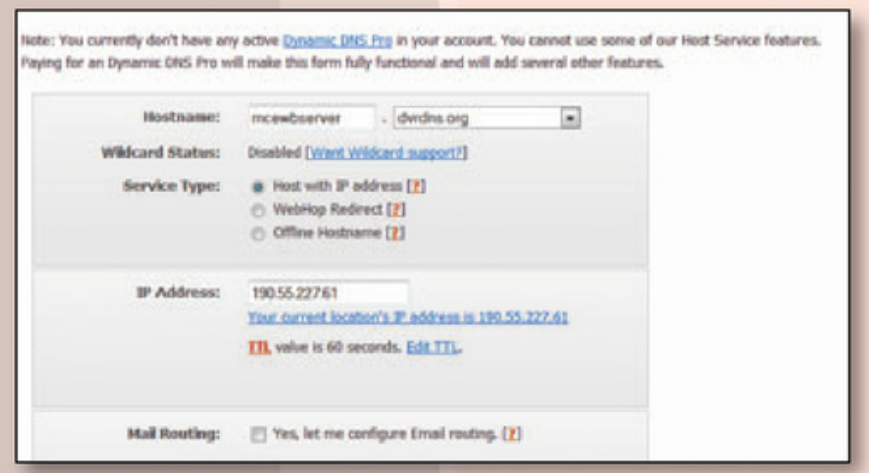
04



05



PASO A PASO



01. Una opción para vincular nuestra IP pública a un nombre de dominio es abrir una cuenta en **www.dyndns.org** y configurar el router para que actualice la base de datos de dyndns cada vez que la IP pública cambie. Para registrarnos en dyndns, hacemos clic en **[Join NOW]** desde la pantalla inicial y, luego, seleccionamos la opción gratuita.

02. Estos son los datos que debemos completar: nombre de host y dirección IP actual (ingresamos la que nuestro ISP nos está asignando es este momento). El nombre de host puede ser cualquiera que no haya sido previamente registrado en dyndns.



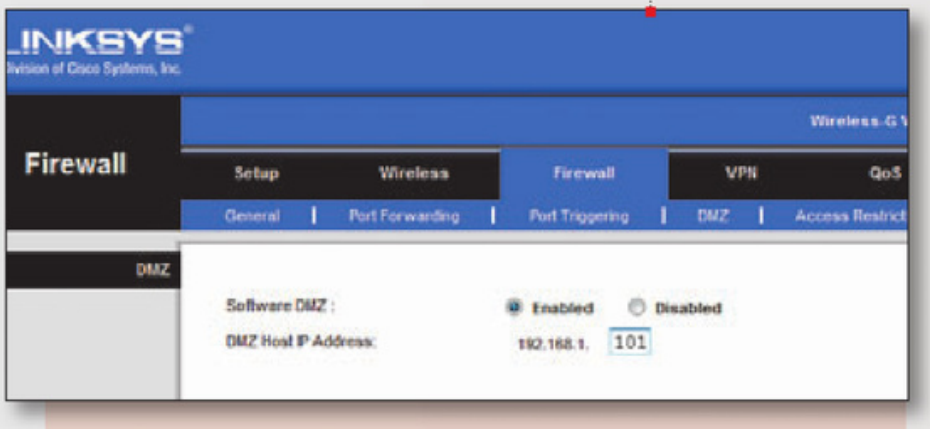
DMZ expone los puertos de una determinada dirección IP a Internet.



03. Para finalizar, hacemos clic en **[Next]**. El asistente nos pedirá un nombre de usuario y una clave para acceder a nuestra cuenta. Debemos recordar estos datos ya que después los utilizaremos para configurar el router al que va a estar conectado nuestro servidor web. Ya tenemos el host creado en dyndns; en este caso, el nombre es **mcewebserver.dyndns.org.**

04. Vemos el host creado recientemente y la dirección IP actual. Si hacemos clic sobre el nombre de host, podemos configurar el TTL (tiempo de vida), parámetro que hace referencia al tiempo durante el cual el host permanece vinculado a la IP en el servidor antes de actualizar la información. Vamos a seleccionar un TTL de 4 horas. En este caso, dyndns revisará nuestra IP cada 4 horas y actualizará su base de datos.

Faltaría tener acceso al Web Server, para lo cual vamos a habilitar la función DMZ (Demilitarized Zone), que expone todos los puertos de una determinada dirección IP a Internet. En este caso, lo hacemos para 192.168.1.101, que es la IP asignada por el router al Web Server. Cada vez que accedamos desde Internet a **http://mcewebserver.dyndns.org**, seremos redirigidos a 192.168.1.101 en nuestra red local.



05. Configuramos el router para que esté vinculado a la cuenta de dyndns y actualice la IP en la base de datos automáticamente. Entonces, vamos a **[Setup/DDNS]** e ingresamos los datos del host creado: usuario y clave. Ya tenemos acceso al router desde Internet mediante un DNS.

ARQUITECTURA DEL PIC18F97J60

El PIC18F97J60 posee controladores para conectarse a una red Ethernet. Veamos cuáles son las características de este elemento.

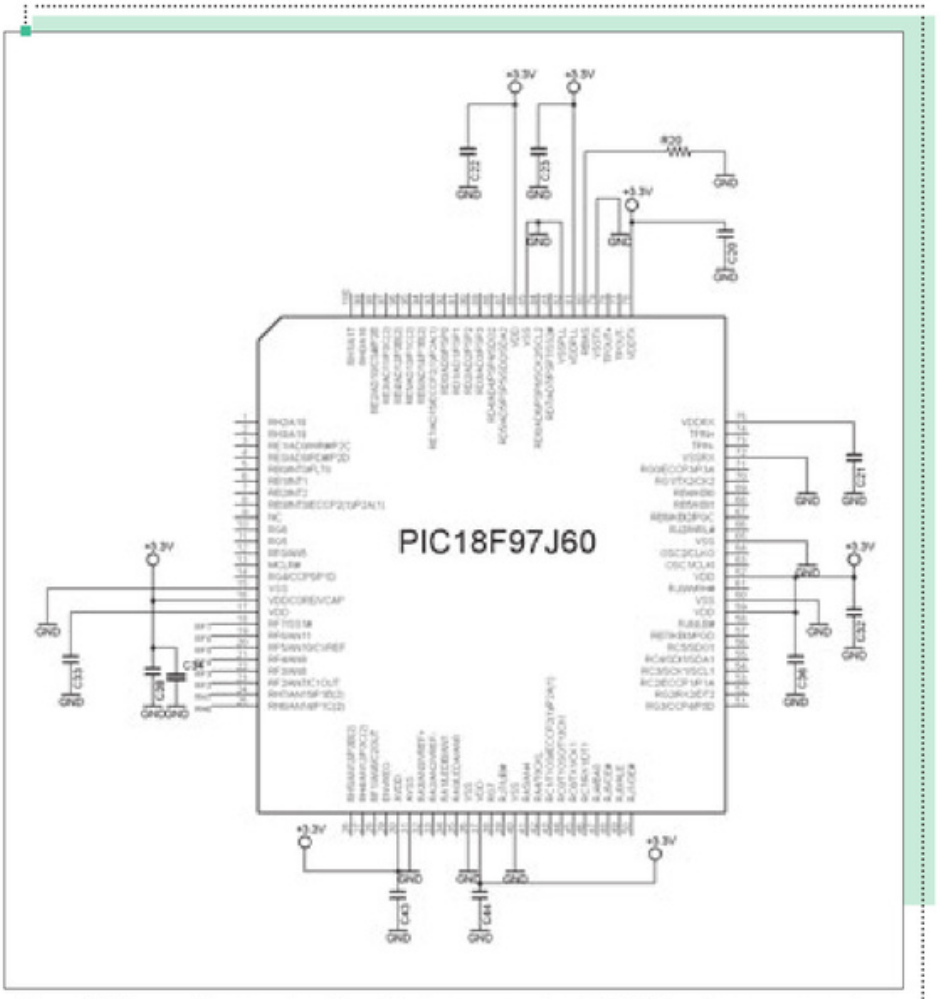
Para alojar el stack TCP/IP en un microcontrolador, precisamos que este sea muy potente y tenga suficiente memoria de programa. Es así como se desarrolló el **PIC18F97J60**, para que un sistema dedicado al monitoreo y control pueda tener una comunicación con la red Ethernet, y evitar los incómodos controladores de red externos, que consumían muchas líneas de datos para el flujo de información.
Las características del PIC18F97J60 son las siguientes:

- Rango de alimentación: 2,35 V a 3,6 V (3,1 V a 3,6 V utilizando el módulo Ethernet)
- Regulador interno de 2,5 V para alimentar al núcleo del procesador
- 128 KB de memoria de programa flash
- 3 KB de memoria de datos RAM
- Encapsulado SMD TQFP de 100 pines
- 70 pines I/O digitales
- Controlador Ethernet compatible con el estándar IEEE 802.3
- Compatible para redes 10/100/1000BaseT
- MAC integrada

Estas especificaciones hacen del PIC18F97J60 un microcontrolador poderoso, con capacidad de conexión a una red, al mismo tiempo que conserva la arquitectura de los PIC18F que estudiamos en esta obra.

Alimentación

El PIC18F97J60 debe ser alimentado con una fuente capaz de entregar entre 2,35 V y 3,6 V; si deseamos utilizar el módulo Ethernet, tenemos que restringir el rango de alimentación entre 3,1 V y 3,6 V. Con estos valores, el fabricante garantiza el funcionamiento correcto del dispositivo.



> **Figura 24. Esquemático con las alimentaciones generales del PIC. Se colocaron capacitores de filtro en los pines de alimentación y el capacitor externo para el regulador interno.**

Además, el núcleo de procesamiento de este microcontrolador trabaja a 2,5 V. Es importante aclarar que el PIC cuenta con un regulador interno que solo necesita un capacitor externo en el pin **Vddcore/Vcap** de 10 µF con un bajo **ESR** (resistencia serie equivalente), como los que proporcionan los capacitores de tantalio. En caso de alimentar el microcontrolador con 3,3 V, debemos colocar el pin ENVREG a un nivel alto para habilitar el regulador interno.

Puertos I/O

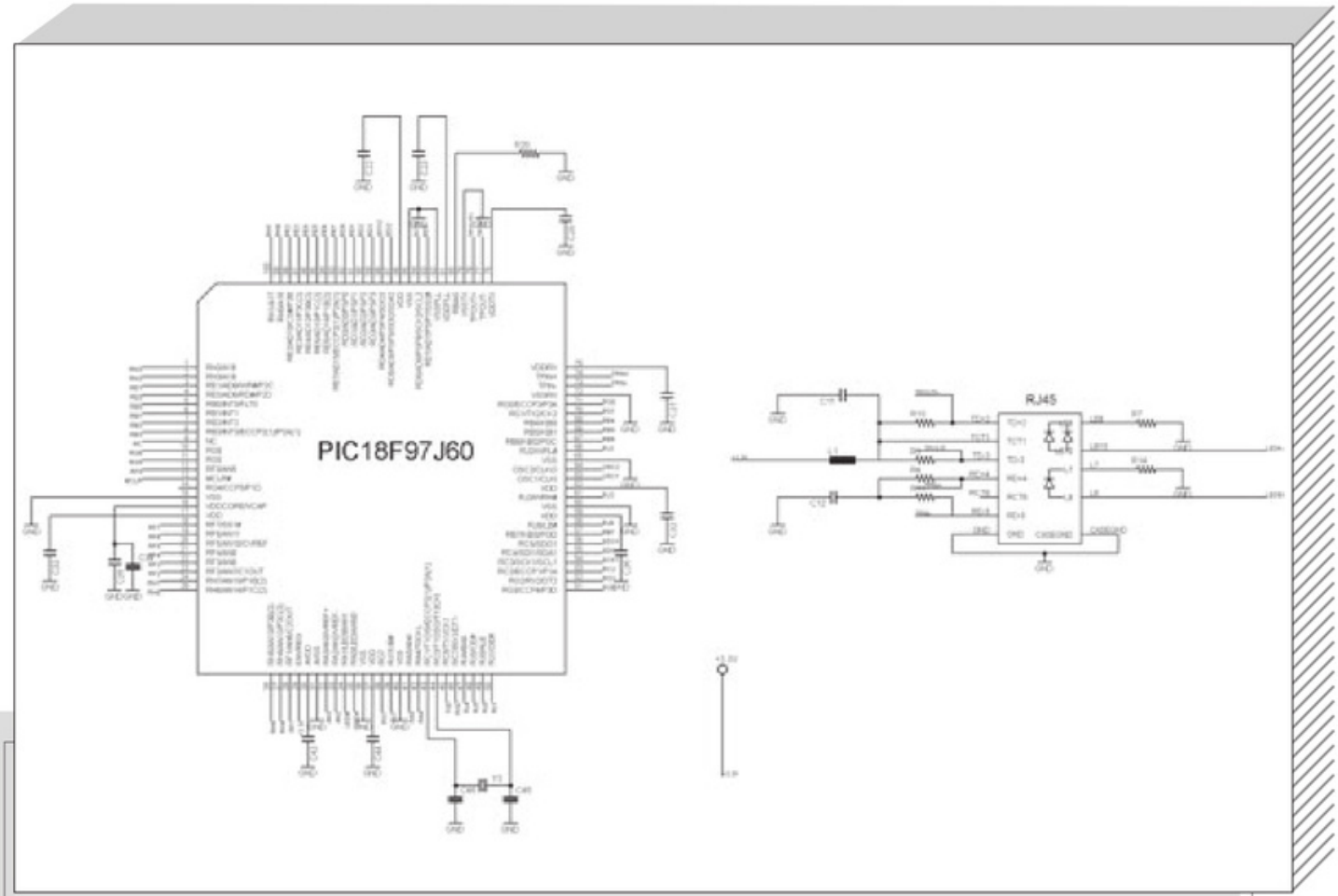
Este microcontrolador dispone de nueve puertos de entrada y salida digital, enumerados de la **A** a la **H** y luego el **J (no existe puerto I)**; es decir PORTA, PORTB, etc. Todos son de 8 pines, excepto el PORTA, que tiene 6. Los puertos poseen diferentes drivers para manejar distintos niveles de potencia. Entonces, los B y C pueden manejar cargas mayores, como la de un LED; en tanto que los D, E y J trabajan con cargas medias, como la interfaz a memoria externa; y los A, F, G y H lo hacen con cargas mínimas y pueden utilizarse como indicadores de estados. Todos los puertos tienen los registros PORT, LAT y TRIS para su control y configuración. Los A y F pueden usarse, además, como canales de entrada al conversor A/D.

Módulo Ethernet

La familia de PICs a la que pertenece este microcontrolador cuenta con controladores Ethernet incorporados que soportan el estándar **IEEE 802.3** para una conexión 10BaseT con par trenzado. Incluyen implementaciones completas para el control de acceso al medio (MAC) y el transceptor de la capa física (PHY). Además, este módulo dispone de un buffer de 8 K de memoria RAM para almacenar los paquetes de datos recibidos y los que serán transmitidos. La conexión física se realiza a través de 4 pines dedicados a la recepción y la información de datos; ellos son:

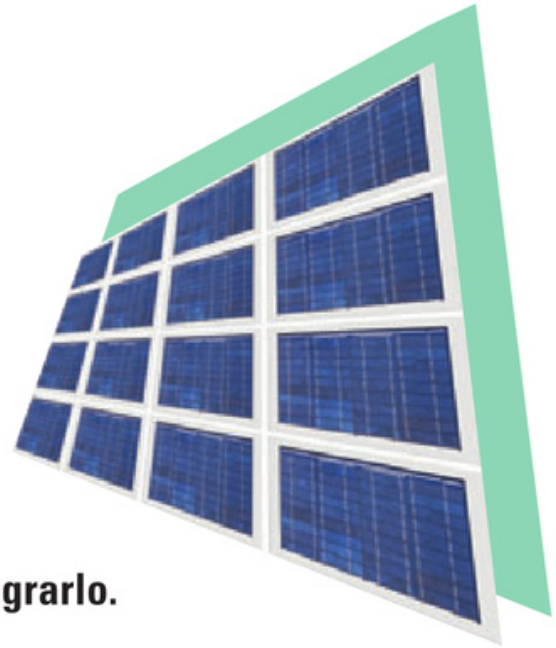
- TPIN+
- TPIN-
- TPOUT+
- TPOUT-

En la **Figura 25** podemos ver el esquemático correspondiente. Lo único que se necesita de manera externa son unos transformadores, pero en la actualidad encontramos conectores RJ-45 que los tienen en su interior. ■



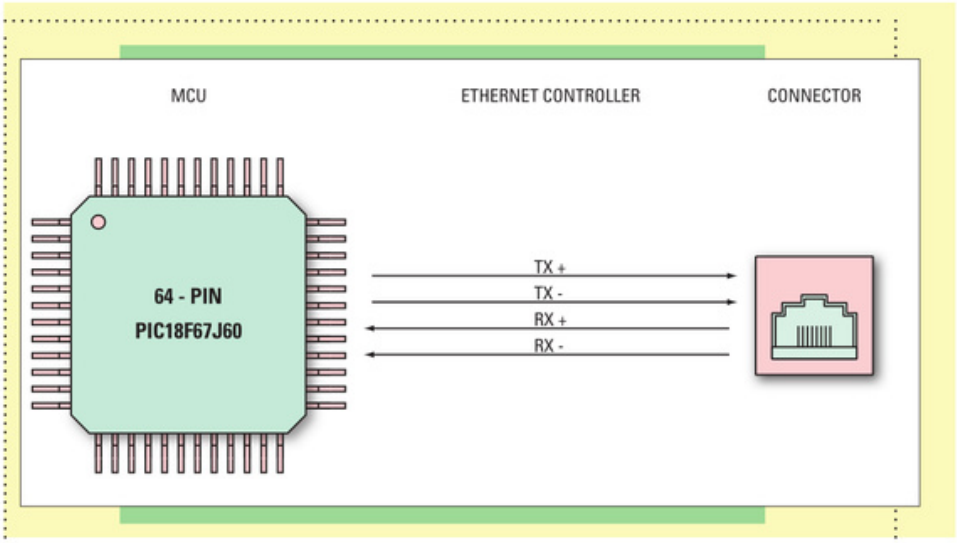
> **Figura 25. Conexión del PIC a un conector RJ-45 que posee bobinas en su interior.**

TELEMETRÍA



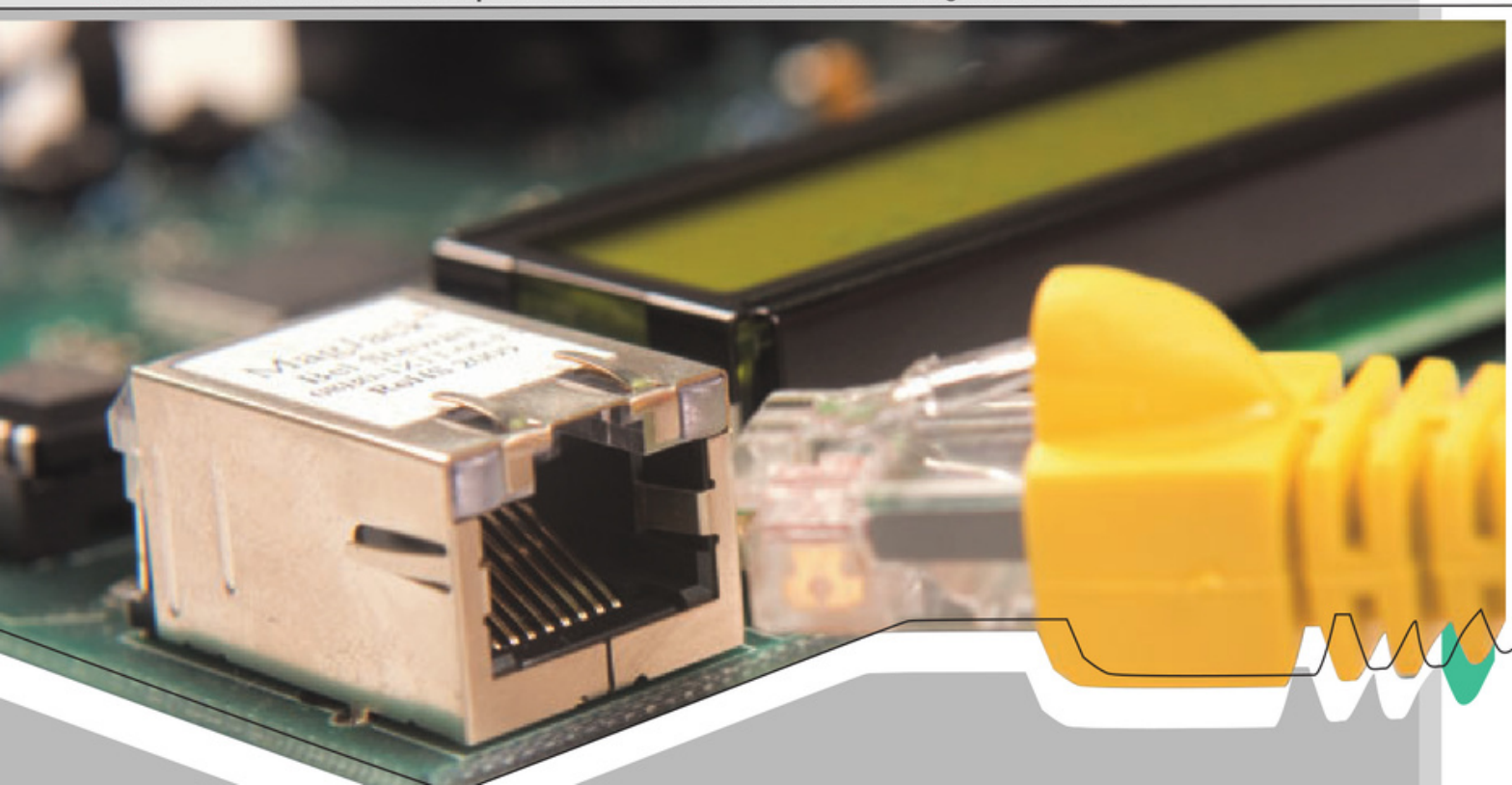
Podemos realizar tareas de monitoreo a distancia a través de una red Ethernet, de modo de controlar un sistema desde nuestros hogares. Veamos la manera de lograrlo.

Como parte de las tareas de automatización, es importante realizar el monitoreo de variables para que el sistema pueda compensar las variaciones no deseadas de estas magnitudes; muchas veces, resulta de utilidad hacerlo a distancia.
En las industrias químicas o en domótica (casas inteligentes), por ejemplo, el monitoreo de variables a distancia es muy beneficioso, porque permite tener un punto de control central de diferentes procesos y mantener así un flujo eficiente del procesamiento de datos.
Una posibilidad de la telemetría es



> **Figura 26. Usando los microcontroladores PIC con módulos Ethernet incluidos, no tenemos necesidad de recurrir a módulos que realicen la interfaz entre ambos.**

> **Gracias a la Red de Redes, Internet, podemos efectuar el monitoreo de variables a grandes distancias.**



enviar los datos necesarios para el monitoreo de un proceso utilizando una red Ethernet. Y aquí la pregunta que surge es: ¿por qué Ethernet? Simplemente, porque en la actualidad, esa es la red más utilizada, presente en muchas oficinas y edificios industriales, porque está basada en estándares **IEEE 802.3,** que garantizan la fiabilidad de las conexiones de red y las transmisiones de datos. Además, el uso de Internet permite alcanzar grandes distancias entre la estación de monitoreo y el operador.

Telemetría de variables analógicas

Los sistemas embebidos o dedicados permiten el monitoreo a distancia de variables analógicas –como temperatura, presión o caudal– gracias a que poseen conversores A/D que digitalizan la señal. El sistema luego encapsula el paquete de datos TCP/IP y lo envía por la red Ethernet, que puede ser una LAN local o Internet, a través de un proveedor del servicio (**ISP**).
El microcontrolador PIC18F97J60 es ideal para estas aplicaciones, ya que incorpora un conversor A/D y tiene el stack TCP/IP cargado en su interior.

Variables dinámicas

Las variables dinámicas permiten que el servidor web tome datos del sistema, como puede ser el valor de un sensor o de la memoria de programa; luego, combinados con una plantilla de una página web, es posible mostrar esos valores en la pantalla de una PC remota.
La página web finalizada se transmite a través de una red o de Internet, y los datos del sistema se presentan en la pantalla del usuario. Por ejemplo, la **Figura 26** indica cómo el sistema toma el dato de una temperatura, el cual, combinado con la plantilla de una página web, se muestra en pantalla.
Con estas variables, podemos insertar cadenas de textos en las páginas web y generar vistosos gráficos. Esto nos ayuda a diseñar el formato y los elementos necesarios de una página.

> CUSTOMHTTPAPP.C

Cuando descargamos la librería de Microchip con el stack TCP/IP, podemos encontrar el archivo CustomHTTPApp.c en el directorio TCPIP Demo App. Este se utiliza para definir las funciones callback que manejan las variables dinámicas.

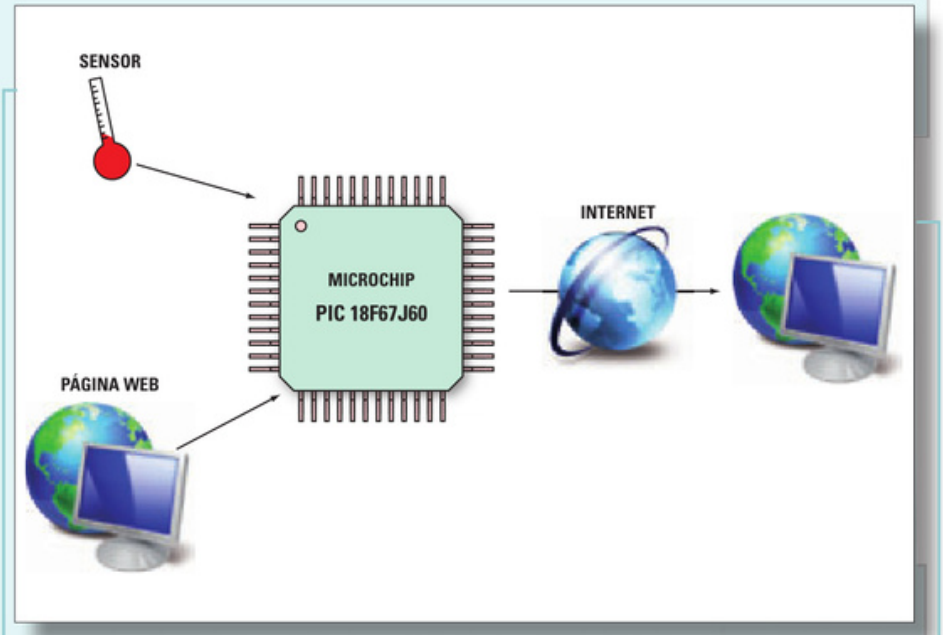
Cómo insertar una variable dinámica

Para insertar una variable dinámica en nuestro proyecto, debemos realizar dos pasos.
Primero, colocar el nombre de dicha variable dentro del código de la página en HTML. Luego, implementar la función que tendrá el control sobre esa variable dinámica. Para realizar el primer paso, debemos contar con una plantilla o una página web a la que le colocaremos una variable dinámica. Estas variables se identifican en el código HTML porque están escritas entre dos caracteres "tilde" (~). Entonces, si queremos agregar una llamada "**VarDin**", incluimos en el código la expresión **~VarDin~.**
Cuando el servidor web se encuentra con esa variable dinámica, ejecuta un llamado a una función callback, que debe estar implementada en el código en C, más precisamente, en el archivo **CustomHTTPApp.c** explicado en la clase anterior.
La función debe tener el formato que se observa en el **Código 09**.
Esta función será llamada cada vez que el servidor web se encuentre con la variable dinámica **VarDin**.

> Código 09

```
Void HTTPPrint_VarDin (void)
{
        // Sentencias
}
```

> Figura 27. Aquí vemos cómo actúa una variable dinámica. El sistema se encarga de combinar el dato con ella y de enviar la página completa por la red.



Cómo utilizar una variable dinámica

Supongamos que debemos supervisar distintas máquinas expendedoras, y nuestro trabajo es asegurarnos de que estas no se queden sin el producto que venden. Esta sería una ardua labor si tuviéramos que revisar las máquinas todos los días. Por eso, resulta mucho más sencillo monitorear el estado de cada una desde la PC de nuestro escritorio.
Esta tarea puede realizarse por medio de variables dinámicas que nos muestren información, como el nombre de la máquina, su ubicación y la cantidad de productos que tiene almacenados a la espera de ser vendidos.
Entonces, si queremos ver, por ejemplo, el nombre de la máquina o su identificación cuando navegamos por la página del servidor web, creamos una variable dinámica **~hostname~** en algún lugar de la plantilla de la página web alojada en el servidor (imaginemos que un desarrollador de páginas web armó esta plantilla para nosotros). Para el ejemplo, dentro del archivo index.htm colocamos el **Código 10**.

> Código 10

```
<div id = "locations">
Maquina ~hostname~
</div>
```

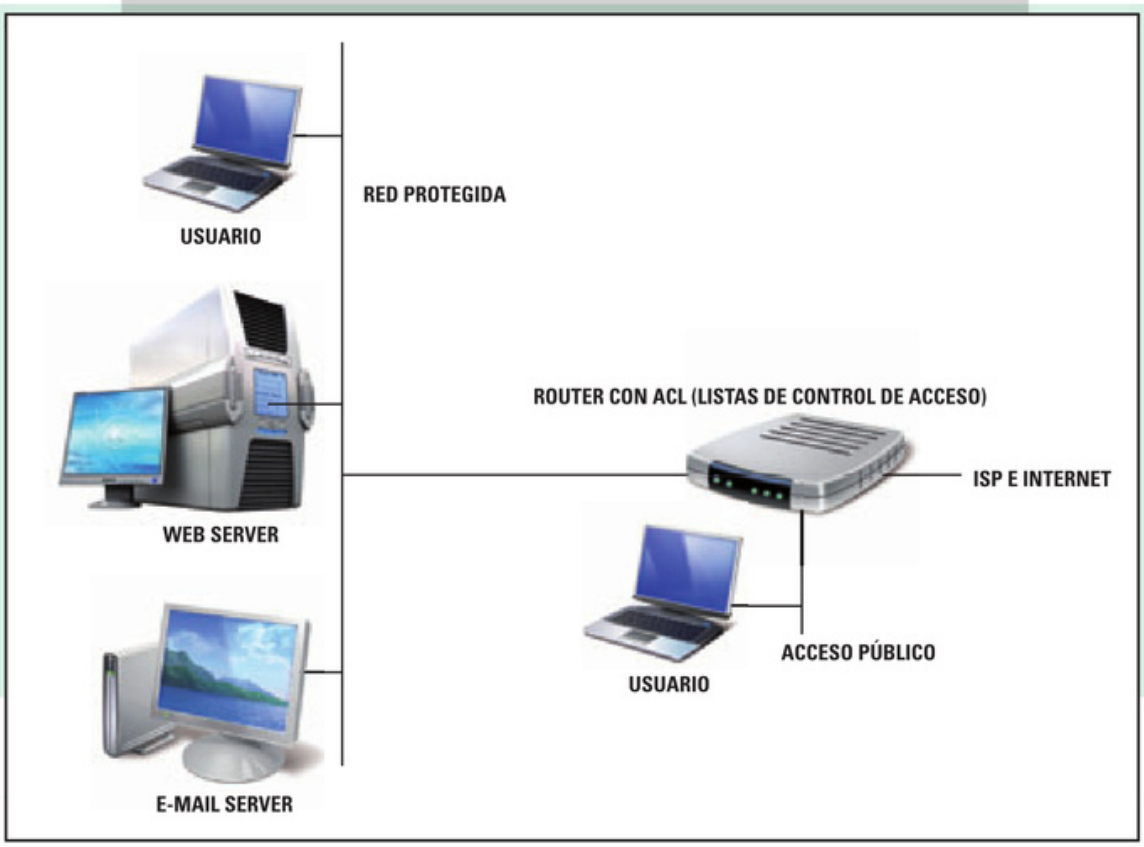
> SOBRE CALLBACK

Las funciones callback deben contener el nombre y el valor de la variable que se define. Además, es necesario tener en cuenta que los datos enviados desde el servidor web tienen que estar codificados en código ASCII. Si pasamos esto por alto, visualizaremos errores en nuestra página web.

> Código 11

```
<div id = "locations">
Maquina ~hostname~
</div>
```

Aquí podemos observar que después de la palabra **máquina**, sigue el nombre de la que corresponde. Luego, escribimos la función que será llamada por el servidor cada vez que encuentre esta variable. Recordemos que esta se escribe dentro del archivo **CustomHTTPApp.c** (**Código 11**).
Esta función se encarga de colocar el nombre de la estación dentro de la página web, reemplazando el valor de la variable dinámica y mostrándola en pantalla. En el nombre de la función podemos ver cómo, luego de **HTTPPrint_**, se debe acoplar el nombre de la variable dinámica insertada en la página.
Dentro de la función, enviamos una cadena de caracteres sobre un segmento



> Figura 28. Esquema de una red con acceso a un servidor de Internet. Con esta configuración, lograremos el intercambio de datos entre el servidor web y una PC remota.

> Código 12

```
<div class="productname">~name(0)~:</div>
<div class="stock">~stock(0)~</div>
</div>
```

> Código 13

```
void HTTPPrint_name(WORD item)
{
TCPPutString(sktHTTP, items[item].name);
}

void HTTPPrint_stock(WORD item)
{
BYTE digits[4];
uitoa(items[item].stock, digits);
TCPPutString(sktHTTP, digits);
}
```

> CONVERSIÓN

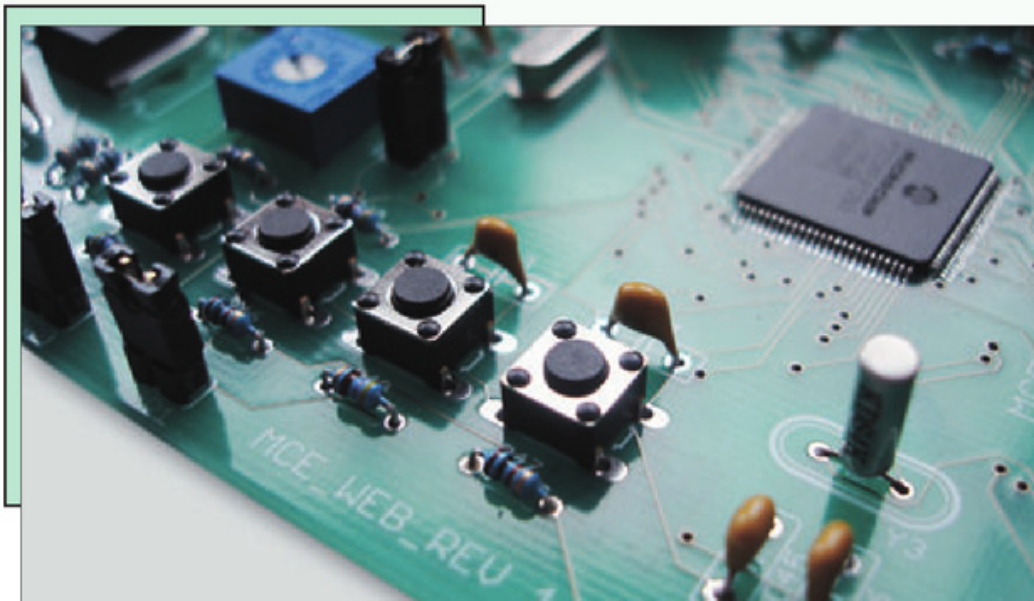
Para realizar la conversión de números enteros a código ASCII, podemos utilizar las funciones **itoa** o **uitoa** (esta última, para valores sin signo). Ambas se encuentran definidas en la librería estándar de C, dentro de la biblioteca stdlib.h. Es un dato que siempre debemos tener en cuenta para ahorrar tiempo.

TCP con el nombre de la estación, que en esta oportunidad es almacenada en una variable que podemos definir al comienzo del código.

Parámetros de las variables dinámicas

También es posible transmitir parámetros junto con las variables dinámicas, y mostrarlos en la página. Esto resulta muy útil cuando tenemos que enviar un vector o un array de datos a la página del servidor.
Veamos cómo agregar los parámetros continuando con el ejemplo anterior. Supongamos que cada estación contiene siete productos para vender, por lo que la información que nos interesará mostrar es el nombre del producto y la cantidad de cada uno. Entonces, en la página vamos a encontrarnos con siete bloques del **Código 12** por cada producto. Vamos a incluir estos bloques por cada producto que contiene la máquina, y a aumentar el índice del array a cada paso. Luego, realizaremos las funciones **callback** en el archivo **CustomHTTPApp.c**. La ventaja de esto es que, en vez de escribir siete funciones para la variable **name** y otras siete para **stock**, solo escribiremos dos, y los valores los pasaremos como parámetros (**ver Código 13**).
En cada función, colocamos entre paréntesis el parámetro de la variable que tomará un valor de 16 bits, llamado **item**. Luego, una estructura devolverá el nombre del producto en la primera función. Para la segunda función, como queremos enviar un número, debemos convertirlo en una cadena de caracteres **ASCII** y después enviarla.

El uso de variables dinámicas es útil cuando tenemos que enviar un array a la página de un servidor.



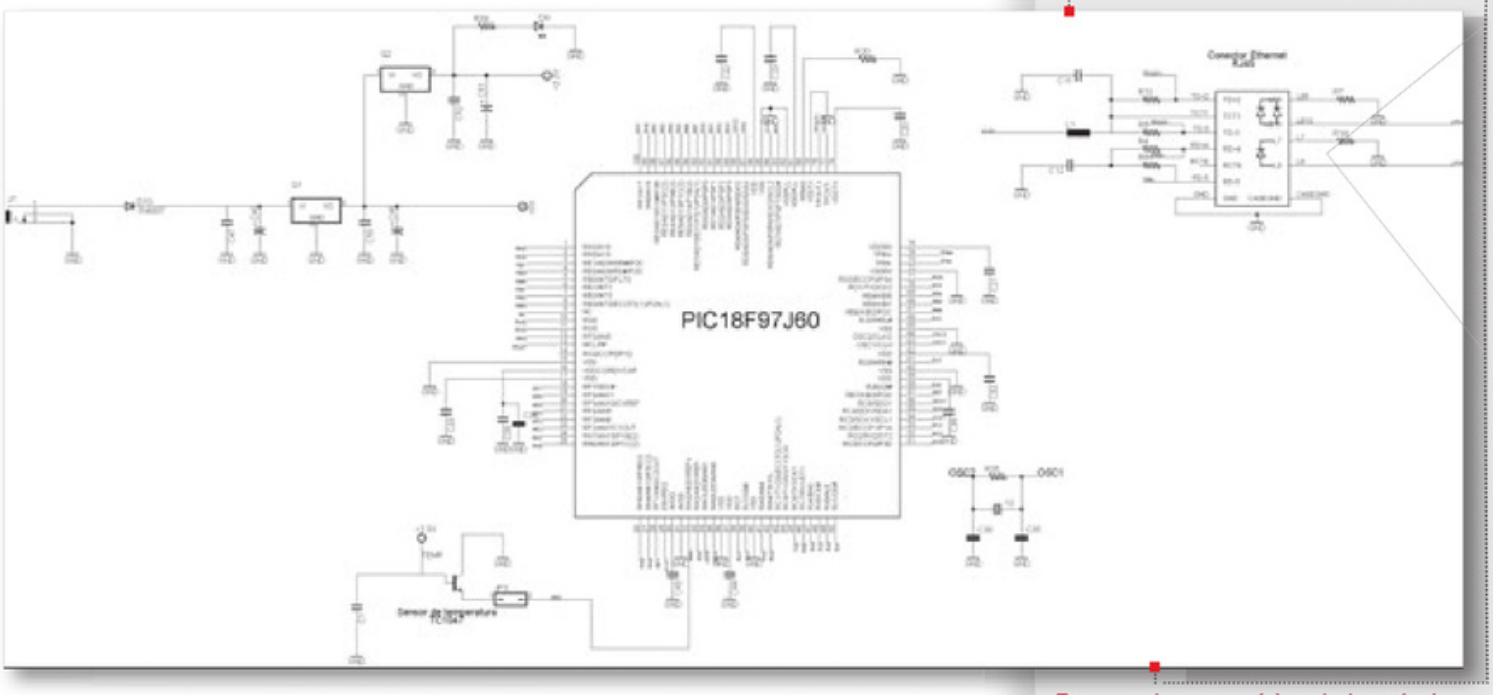
MONITOREO DE LA TEMPERATURA POR ETHERNET

Con todo lo visto hasta ahora, aprenderemos a enviar el estado de un sensor a través de una red, para visualizarlo en una página web.

01

```
- <response>
    <led0>~led(0)~</led0>
    <led1>~led(1)~</led1>
    <led2>~led(2)~</led2>
    <led3>~led(3)~</led3>
    <led4>~led(4)~</led4>
    <led5>~led(5)~</led5>
    <led6>~led(6)~</led6>
    <led7>~led(7)~</led7>
    <btn0>~btn(0)~</btn0>
    <btn1>~btn(1)~</btn1>
    <btn2>~btn(2)~</btn2>
    <btn3>~btn(3)~</btn3>
    <temp0>~temp~</temp0>
    <relay1>~relay1~</relay1>
    <relay2>~relay2~</relay2>
  </response>
```





Este es el esquemático de la práctica, donde, básicamente, tenemos el microcontrolador conectado a la red a través de un conector RJ-45. Luego están el circuito de alimentación y el sensor de temperatura, conectado en el pin RA3/AN3, el cual es el canal 3 del conversor AD.

En este paso a paso, realizaremos el monitoreo de una variable analógica, como la temperatura de un recinto, desde un lugar remoto que estará apartado de nuestra ubicación, y en donde veremos la variación de esta magnitud en tiempo real. Mostraremos el esquemático y los pasos para asignar una variable dinámica a nuestra página y compilar el proyecto. ■

PASO A PASO

02

```
29      <p>Temp: <span id="temp0" style="font-weight:normal">?</span>
```

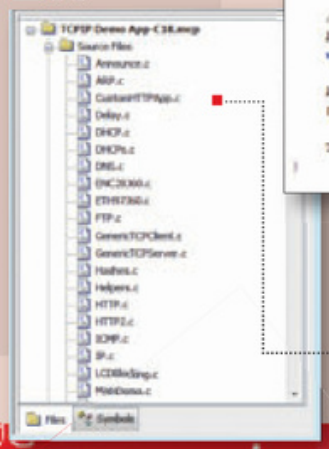
03

```
void HTTPPrint_temp(void)
{
    char ANOString[30];
    float ADval;

    // Wait until A/D conversion is done
    ADCON0bits.GO = 1;
    while(ADCON0bits.GO);

    ADval = (float)ADRES/10.00;
    ftoa(ADval ,ANOString,2,'.');

    TCPPutArray(sktHTTP, (void *)ANOString, strlen((char*)ANOString));
}
```



01. Vamos a comenzar insertando la variable dinámica, para lo cual procedemos como se explicó en la clase anterior. Primero editamos el archivo **status.xml**, en el que vamos a declarar la variable dinámica temp con la siguiente sentencia: **<temp0>~temp~</temp0>**.





Recordemos que el microcontrolador se conecta a la red a través de un conector RJ-45.

04

```
void HTTPPrint_temp(void)
{
    char ANOString[30];
    float ADval;

    // Wait until A/D conversion is done
    ADCON0bits.GO = 1;
    while(ADCON0bits.GO);

    ADval = (float)ADRES/10.00;
    ftoa(ADval ,ANOString,2,'.');

    TCPPutArray(sktHTTP, (void *)ANOString, strlen((char*)ANOString));
}
```

02. Buscamos la plantilla de la página web index.html que provee la librería de Microchip y colocamos la siguiente sentencia, que se ve en la línea 29: **<p>Temp: ?**. Esta reemplazará a la variable dinámica y colocará el valor de la temperatura que arroje la función callback.

03. En el archivo CustomHTTPApp.c escribimos la función callback que controlará la variable dinámica definida. En esta función, realizamos la conversión a digital de la temperatura, y la llevamos a ASCII para mandarla a la página mediante la función ftoa. Luego, enviamos los caracteres con TCPPutArray().

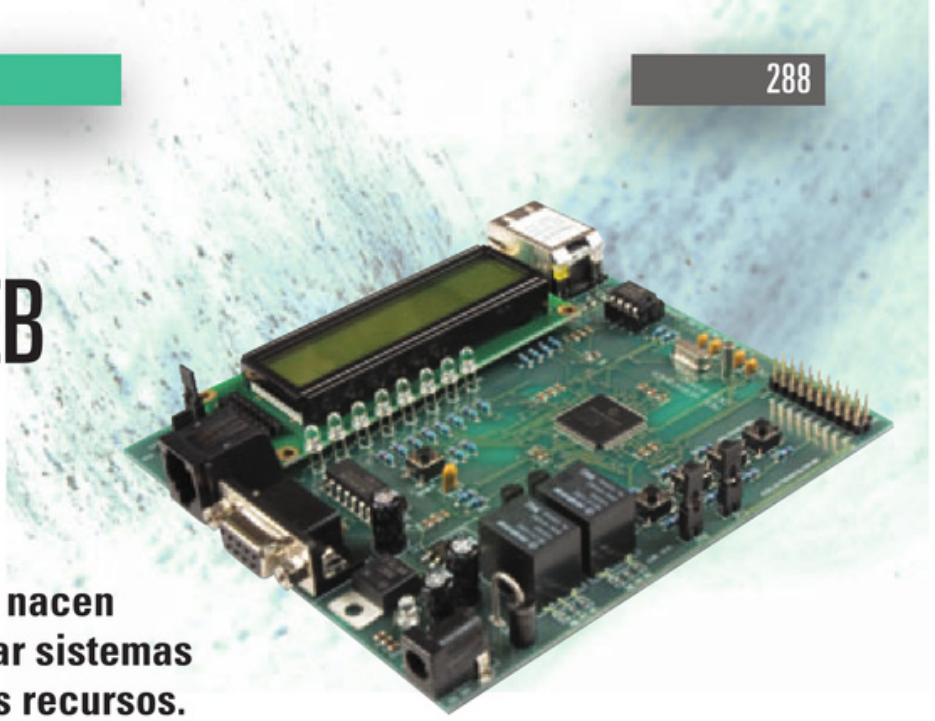
04. Ejecutamos la utilidad que vimos la clase anterior, TCP/IP Configurations Wizard. Configuramos y compilamos el proyecto, y lo grabamos al PIC. A continuación, cargamos la página con el MPFS en la memoria externa del PIC.

05



05. Una vez hecho esto, podemos ver la página accediendo desde el browser a la dirección IP del servidor. Podemos observar que la temperatura se actualiza con cada llamada a la función que la controla.

SERVIDORES WEB DEDICADOS



Los servidores web dedicados nacen de la necesidad de implementar sistemas de control eficientes y de bajos recursos.

En la actualidad, muchas industrias tienden a incorporar dispositivos embebidos o dedicados a tareas específicas.
A su vez, se trata de que estos puedan comunicarse con otros dispositivos para crear una estructura de supervisión mucho más efectiva. Los servidores web dedicados son un claro ejemplo.

Funciones de un servidor web dedicado para control

Los sistemas embebidos realizan, en general, tres tareas:

- Toman una entrada, que puede ser de un sensor.
- Realizan un proceso en tiempo real.
- Obtienen una salida, que puede ser una respuesta o el control de un actuador.

> La ventaja de un servidor web es que permite realizar el control de una planta a distancia.

La ventaja de un servidor web es que presenta una interfaz mucho más amigable, y permite realizar el control de una planta desde la PC de nuestro hogar.

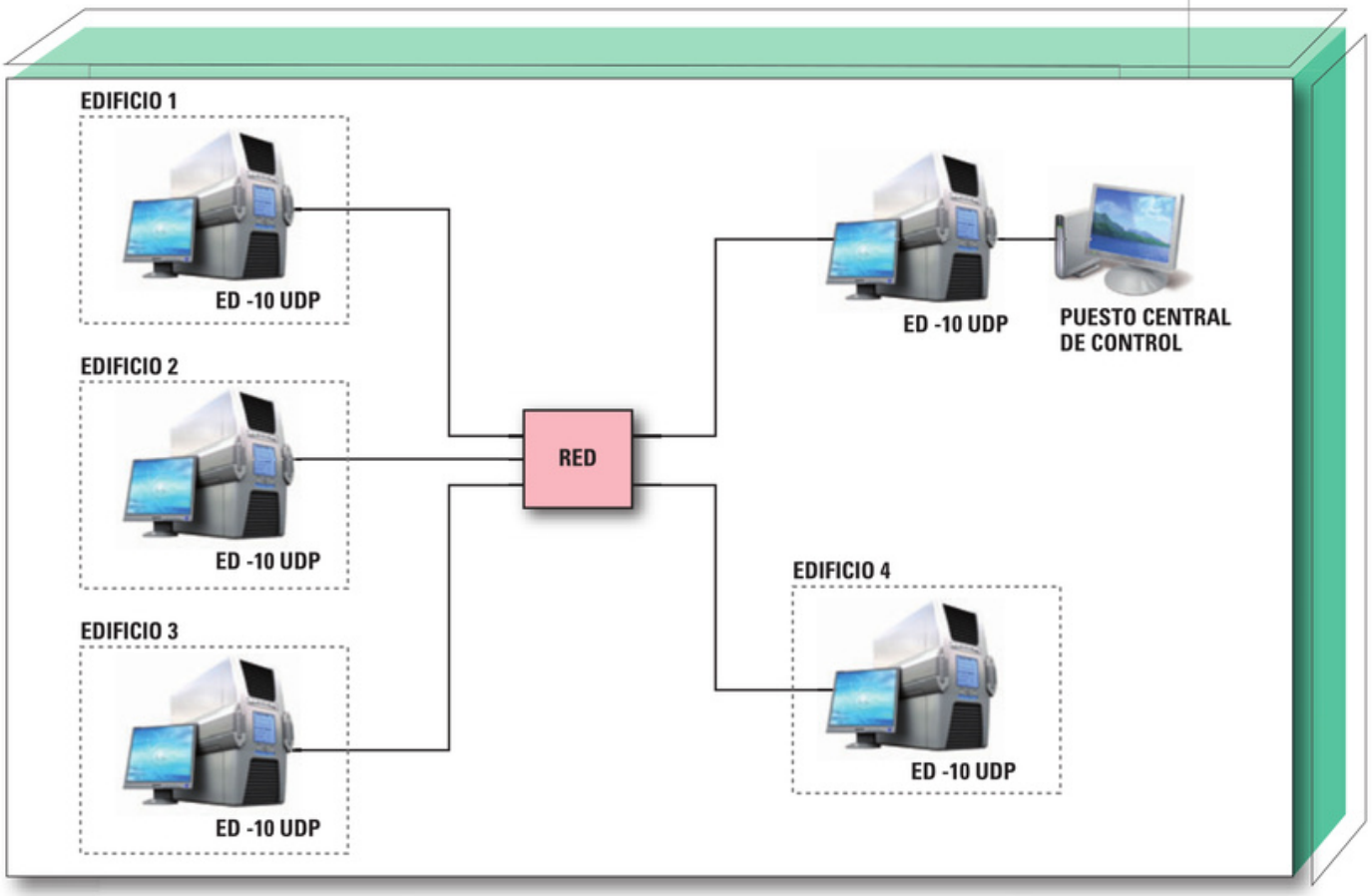
Alternativas comerciales

Si necesitamos acortar los tiempos de desarrollo o queremos evitar la complejidad del proceso de armado de un servidor web dedicado, existe la alternativa de comprar módulos de entrenamiento, que proveen las herramientas necesarias para ejercitarse en el manejo del stack TCP/IP de Microchip. Estos módulos incluyen el hardware necesario para conectarse a una red Ethernet y vienen provistos de un firmware para comunicarse con el módulo a través de una página web.
En el mercado encontramos el módulo de entrenamiento **MCE Web Server DEV**, que cuenta con un microcontrolador PIC18F97J60.
El PIC está cargado con un programa demostrativo para aprender a utilizar el stack TCP/IP de Microchip y realizar todas las aplicaciones sobre Ethernet con PIC que estudiamos en esta obra. Además, cuenta con algunos



> APLICACIONES COMERCIALES

Para aplicaciones comerciales, contamos con módulos que solo disponen del hardware necesario para efectuar una conexión a la red. Este es el caso del **MCE Web Server CORE**, que tiene, simplemente, el PIC18F97J60, el circuito de alimentación interno, un conector para programación in-circuit y un conector RJ-45 para los cables de red. Con este módulo, podemos agregar conectividad a nuestros desarrollos; es un módulo de producción.



> **Figura 29. Ejemplo de control para un sistema de alertas de incendio municipal, donde las redes de los edificios son Ethernet.**

periféricos, como LEDs, display LCD y un sensor de temperatura para interactuar con el PIC.

Control remoto

Hasta el momento, vimos cómo monitorear parámetros a distancia; ahora aprenderemos a controlar esas variables de manera remota. En muchas industrias, es necesario usar algún equipo que permita realizar, por diferentes cuestiones, el control de la planta en forma remota, ya sea porque el lugar no es accesible para el personal de planta o por razones de seguridad. Existen muchos equipos que permiten realizar esta tarea y que utilizan diferentes protocolos de comunicación. En la actualidad, en diversas industrias y en casi todas las nuevas líneas de producción se está implementando Ethernet industrial como red de comunicación básica, en especial, en aquellos casos en que todo el sistema se controla desde una computadora central. La conectividad y la sencillez de la instalación han convencido a los departamentos de ingeniería de producción sobre la conveniencia de reconvertir las líneas de producción existentes para que puedan emplear Ethernet industrial.

En la actualidad, casi todas las nuevas líneas de producción están implementando Ethernet industrial.

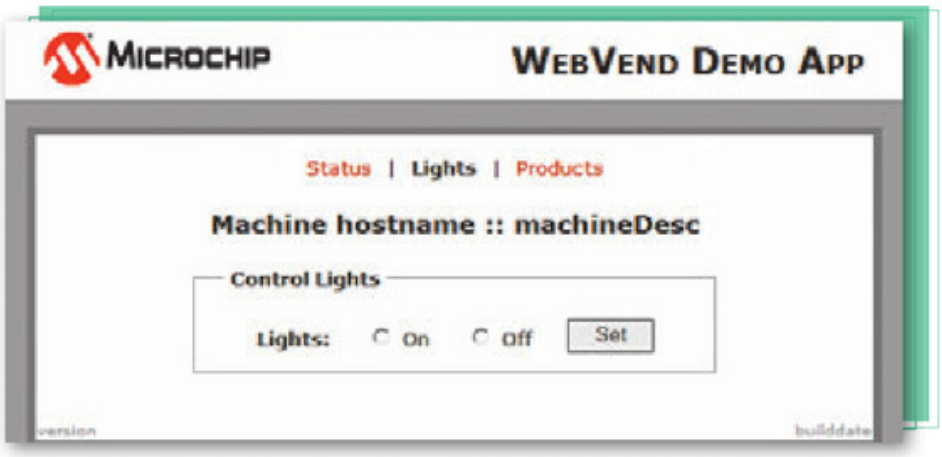
Ejemplo de control remoto

Sobre este tema encontramos una infinidad de aplicaciones, como los sistemas de alerta de incendio municipal. Estos trabajan sobre Internet porque los puestos de control de incendio están distribuidos por toda la ciudad. Como todos los edificios cuentan con una red Ethernet, podemos utilizar el protocolo IP para establecer la comunicación entre las estaciones, y transportar los datos en los protocolos TCP o UDP para que una computadora central lleve adelante las acciones de control. También encontramos muchos ejemplos en domótica (casas inteligentes), donde podemos enviar un e-mail y controlar, por ejemplo, el sistema de refrigeración y las alarmas de un hogar.

ENCENDER UN LED Y UN RELAY POR ETHERNET

Hemos aprendido a enviar datos a través del método GET para formularios web; ahora explicaremos cómo interactuar con nuestro servidor.

Vamos a efectuar de manera práctica el encendido de un LED y de un relay a través de un formulario web, desde el cual enviaremos los datos al servidor
web conectado a la PC por medio del conector de red o, si fuera necesario, a través de un router.
En primer lugar, realizaremos en forma sencilla el encendido de un LED por el método GET y, luego, extenderemos el ejercicio para encender de manera simultánea varios LEDs y relays. Con todas estas nociones, estaremos preparados para llevar a cabo el control básico a distancia de cualquier sistema que lo requiera.



Esta es la plantilla que podemos encontrar en la librería de Microchip, donde está definido el formulario que acabamos de ver.

Encender un LED por Ethernet

Comenzaremos con un ejemplo sencillo para encender un LED a través del método que estuvimos estudiando. Lo primero que debemos tener es una plantilla web con un formulario. En la librería de Microchip que trae el CD de la obra, encontraremos una plantilla en la carpeta **TCPIP WebVend App**; es un ejemplo simple con un formulario para el encendido de un LED.
En el código HTML reconoceremos el formulario, porque comenzará con el campo **<form>** y, luego, seguirán las etiquetas **<input>**. Veremos algo similar a lo que muestra el **Código 14**.

Una vez que reconocemos el campo del **<form>**, podemos ver cómo se define el método GET para enviar los datos, con la sentencia **method = "get"**.

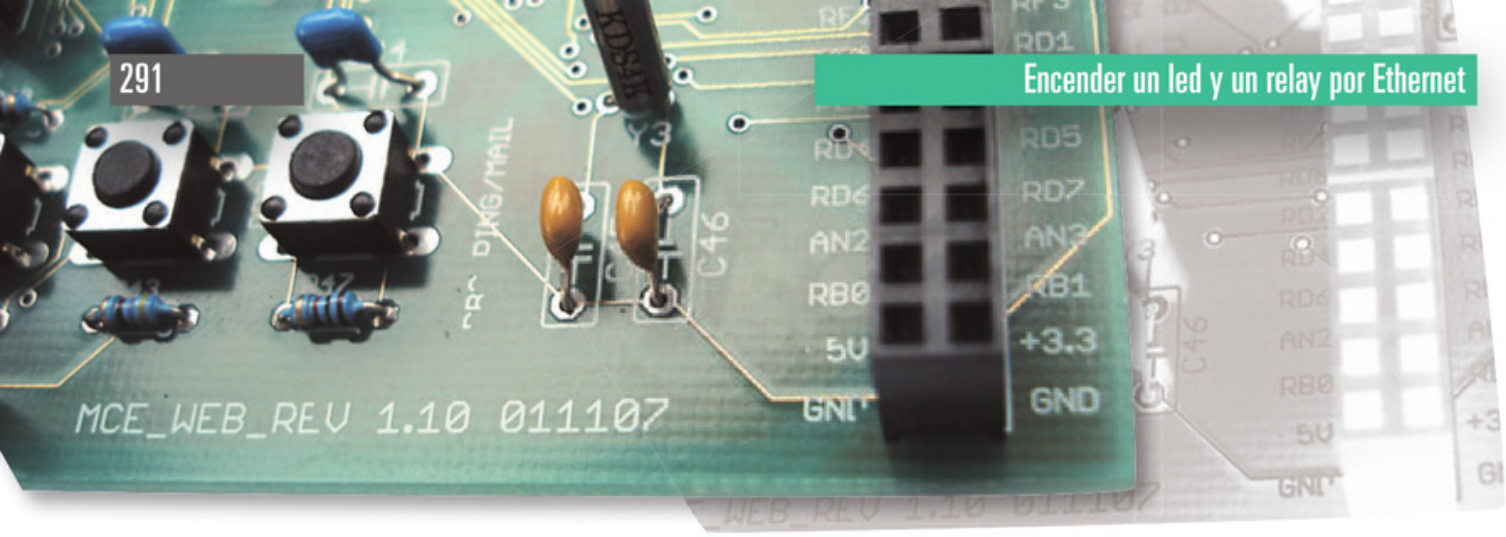
> **Código 14**

```
<form method="get" action="lights.htm">

<!-- Estos dos campos son creados para seleccionar el On/Off -->
<input type="radio" name="lights" value="on" ~lights_chk(1)~ /> On
<input type="radio" name="lights" value="off" ~lights_chk(0)~ /> Off

<!-- En el formulario hay un botón de seteo -->
<input type="submit" class="btn" value="Set"/>

</form>
```



Hagamos un análisis del **Código 09** de la página 19. Comenzamos definiendo la función **HTTPExecuteGet()**, que devuelve un parámetro del tipo **HTTP_IO_RESULT**, una variable definida por el stack TCP/IP de Microchip.
Luego de definir las variables que vamos a utilizar en la función, el primer paso es comprobar el nombre del archivo; podemos utilizar la función **MPFSGetFilename();** para buscar su nombre. Si tenemos varios formularios, nos será de utilidad saber cuál es el que generó el envío de datos.
Para ese fin, realizamos el bloque condicional **if**. Simplemente, hacemos la comparación de las cadenas de caracteres con la función **strcmppgm2ram()** y, si son diferentes, salimos de la función. En nuestro caso, de todos modos tenemos un solo formulario para atender. El próximo paso es buscar el campo de datos.
En nuestro ejemplo, solo tenemos un dato, pero es bueno saber que si hay más de uno para recibir, los campos no siempre llegan en orden. Por eso, efectuamos la búsqueda correspondiente con la función **HTTPGetROMArg()**, que recibe como argumento la estructura que contiene los datos **(curHTTP.data)** y el nombre del formulario del cual estos provienen. Lo

> UN DATO IMPORTANTE

La función strcmppgm2ram() compara cadenas de string. Por ejemplo, si dos cadenas de caracteres son iguales, la función devuelve 0, lo cual es muy útil para validar datos ingresados por el usuario. Esta función está definida en la librería string.h, además de haber otras que se utilizan para manejar cadenas de este tipo.

Luego de definir la variable que utilizaremos en la función, comprobamos el nombre del archivo.

> **Código 15**

```
<form method="get" action="forms.htm">
<div class="examplebox">
<b>R2:</b> <select name="relay2"><option value="on"~ledSelected(9,TRUE)~>On</option><option value="off"
~ledSelected(9,FALSE)~>Off</option></select> 
<b>R1:</b> <select name="relay1"><option value="on"~ledSelected(8,TRUE)~>On</option><option value="off"
~ledSelected(8,FALSE)~>Off</option></select> 

<b>L3:</b> <select name="led3"><option value="1" ~ledSelected(3,TRUE)~>On</option><option value="0"
~ledSelected(3,FALSE)~>Off</option></select> 
<b>L2:</b> <select name="led2"><option value="1" ~ledSelected(2,TRUE)~>On</option><option value="0"
~ledSelected(2,FALSE)~>Off</option></select> 
<b>L1:</b> <select name="led1"><option value="1" ~ledSelected(1,TRUE)~>On</option><option value="0"
~ledSelected(1,FALSE)~>Off</option></select> 
<br /><input type="submit" value="Enviar" style="margin-top:5px;">
</div>
</form>
```

que nos devuelve es un puntero a los datos almacenados, que después comparamos con la cadena de caracteres "**ON**". Si son iguales, el LED se enciende; si son diferentes, se apaga. Por último, retornamos con el valor **HTTP_IO_DONE** para indicar que terminamos y que no necesita ser llamado otra vez.

Manejo de varios LEDs y relays

Vamos a agregar más comandos al ejemplo, esta vez, para poder manejar tres LEDs y dos relays. Ahora, en la página tendremos un formulario más grande, donde agregaremos todos los comandos necesarios. Podremos encontrar una plantilla con la cual trabajar en el directorio **[TCPIP Demo App\WebPages2]**, con el nombre **form.htm**.
La estructura del formulario donde agregamos dos relays y tres LEDs puede observarse en el **Código 15** de la página 291.
Luego, tenemos el código de la función que será llamada para controlar el estado de los LEDs y de los relays (**Código 16**).

> Código 16

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
        BYTE *ptr;
        BYTE filename[20];

        // Cargamos el nombre del archivo
        // Asegurarse que filename[] tiene el tamaño adecuado
        MPFSGetFilename(curHTTP.file, filename, 20);

        // Si los datos provienen de form.htm
        if(!memcmppgm2ram(filename, "forms.htm", 9))
        {
                // Revisamos el estado de cada LED
                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"led4");
                if(ptr)
                        LED4_IO = (*ptr == '1');

                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"led3");
                if(ptr)
                        LED3_IO = (*ptr == '1');

                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"led2");
                if(ptr)
                        LED2_IO = (*ptr == '1');

                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"led1");
                if(ptr)
                        LED1_IO = (*ptr == '1');


//--------------------Control del relay-----------------------------

                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"relay1");
  if(ptr) // nos aseguramos que el valor no sea nulo
  {
                        if(strcmppgm2ram((char*)ptr,(ROM char*)"on") == 0)
                                RELAY1_IO = 1;
                        else
                         RELAY1_IO = 0;
                }

                ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"relay2");
  if(ptr) // nos aseguramos que el valor no sea nulo
  {
                        if(strcmppgm2ram((char*)ptr,(ROM char*)"on") == 0)
                                RELAY2_IO = 1;
                        else
                         RELAY2_IO = 0;
                }

}
// Finalizamos
return HTTP_IO_DONE;
}
```

> FORMULARIOS

Si tenemos varios formularios que pueden enviar datos con el método GET, existe la posibilidad de utilizar la función **MPFSGetFilename** para obtener el nombre del archivo que está interactuando con el servidor. De este modo, podemos tener varios formularios que interactúen con nosotros, y discernir cuál es el que envía los datos.

A continuación, vemos el campo de las etiquetas **input**, donde podemos ver el parámetro "**name**". Este dato es muy importante porque será utilizado en la función **callback** que llame el envío de los datos.
En este caso, el parámetro toma el nombre "**lights**". Luego, podemos ver la definición de dos variables dinámicas: **lights_chk(0)** y **lights_chk(1)**, ambas escritas entre los caracteres (~). Estas variables dinámicas están para ajustar un valor predeterminado cuando se carga el formulario. Además, tienen su propia función **callback**, y la única misión que cumplen es mantener un valor predeterminado al cargar el formulario. La última etiqueta de **<input>** genera un botón para enviar los datos del formulario. Como no tiene en su sentencia el parámetro "**name**", no manda ninguna información adicional. Este es el detalle de la porción del **Código 08** de la página 18.

Función callback para encender el LED

Como mencionamos anteriormente, los datos enviados por el método GET son procesados por la función **HTTPExecuteGet()**. Los que recibimos del formulario automáticamente se decodificarán y se almacenarán en la estructura **curHTTP.data**.
Luego, podemos obtener los datos recibidos con la función **HTTPGetArg()** y localizar los valores asociados a los nombres de los parámetros. Entonces, la forma básica de una función que es llamada luego del envío de datos GET es la siguiente: primero, inicia la búsqueda de los valores de entrada y, luego, realiza las acciones necesarias, según los valores hallados. El **Código 17** muestra de qué manera se realiza esta función.

> **Código 17**

```
HTTP_IO_RESULT HTTPExecuteGet(void)
{
BYTE *ptr, i, name[20];

// Busco el nombre del archivo
MPFSGetFilename(curHTTP.file, name, 20);

// Nos aseguramos que es el archivo deseado
if(strcmppgm2ram((char*)name,(ROM char*)"lights.htm") != 0)
return HTTP_IO_DONE;

// Buscamos el valor del nuevo estado
ptr = HTTPGetROMArg(curHTTP.data, (ROM BYTE *)"lights");

if(ptr) // nos aseguramos que el valor no sea nulo
{// seteamos el nuevo estado
if(strcmppgm2ram((char*)ptr, (ROM char*)"on")== 0)
LED1_IO = TRUE;
else
LED1_IO = FALSE;
}
// Finalizamos
return HTTP_IO_DONE;
}
```

Los datos recibidos se almacenarán y decodificarán en la estructura curHTTP.data.



MÓDULO ZeroG

Conoceremos todas las características de este módulo que nos servirá para agregar conectividad inalámbrica a nuestros desarrollos. Veamos de qué se trata.

La posibilidad de agregar conectividad inalámbrica en nuestros desarrollos ofrece muchas ventajas, como la capacidad de interconectar varias estaciones sin tener que modificar la infraestructura edilicia, como ocurre con las redes cableadas. Además, las redes **Wi-Fi** se encuentran estandarizadas por la norma **IEEE 802.11**, lo que nos asegura la compatibilidad entre distintos dispositivos. Entre las desventajas que presentan podemos nombrar la baja velocidad de transferencia, si las comparamos con las redes cableadas; y la falta de seguridad en el envío de datos, ya que estos pueden ser capturados durante la transmisión. Pero esto último puede solucionarse trabajando con redes seguras, que utilizan encriptación de datos.



Figura 30. Estructura típica de una red Wi-Fi hogareña. El router combinado con un punto de acceso trabaja para que todas las estaciones cliente tengan conectividad.

Estándar 802.11

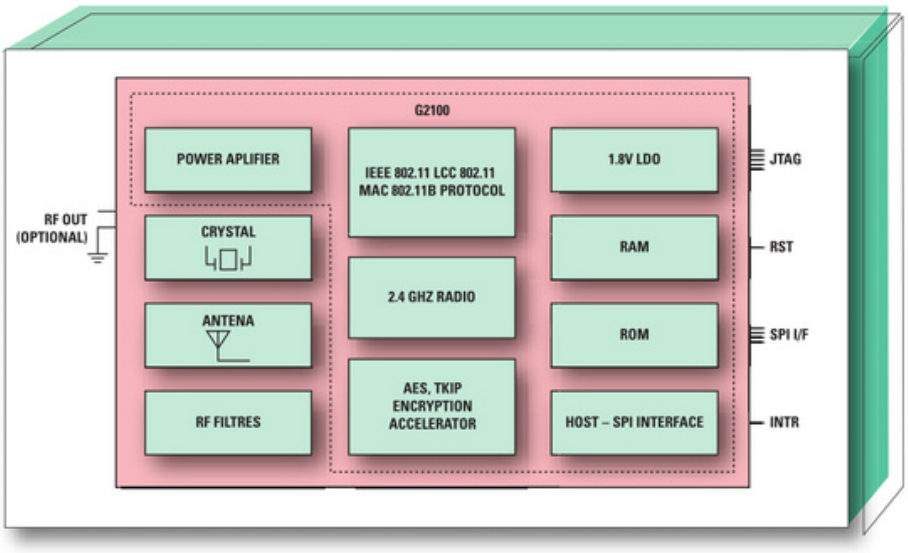
Este estándar establece las especificaciones y las normas de funcionamiento para las capas 1 y 2 del modelo OSI. Como características especiales, esta red trabaja en las zonas de frecuencia de **2,4 GHz** para los estándares **802.11b** y **802.11g**, y en la frecuencia de **5 GHz** en la norma **802.11a**. Las velocidades más comunes de transmisión están en **11 Mbps** y en **54 Mbps**. Básicamente, encontramos dos estructuras para la interconexión de los diferentes dispositivos. Una de ellas se llama red Wi-Fi de **infraestructura**, donde existen dos elementos: uno o más puntos de acceso y las estaciones cliente, que se conectan al servidor a través de los primeros. La otra estructura es la red **ad-hoc**, donde las estaciones cliente se conectan entre sí para intercambiar información.



Módulo ZG2100M

Este módulo es fabricado por la empresa **ZeroG** y, básicamente, agrega conectividad Wi-Fi en dispositivos embebidos. Trabaja bajo el estándar **802.11b**, que especifica una frecuencia de trabajo de **2,4 GHz** y alcanza velocidades de hasta **2 Mbps.**

Posee internamente todo lo necesario para la conexión a una red Wi-Fi, porque tiene una dirección **MAC** pregrabada y los componentes de **RF** necesarios para emplear el medio físico, incluyendo la antena. Esto es ideal para nuestras aplicaciones, porque nos olvidamos de tener que lidiar con el arduo trabajo que implica diseñar una antena en el PCB. Simplemente, debemos tomar algunas precauciones en el layout (esquema de distribución de los componentes en el PCB) para poder integrar este módulo, que veremos más adelante.



> **Figura 31. Diagrama funcional del módulo ZG2100M.**

Características del módulo ZG2100M

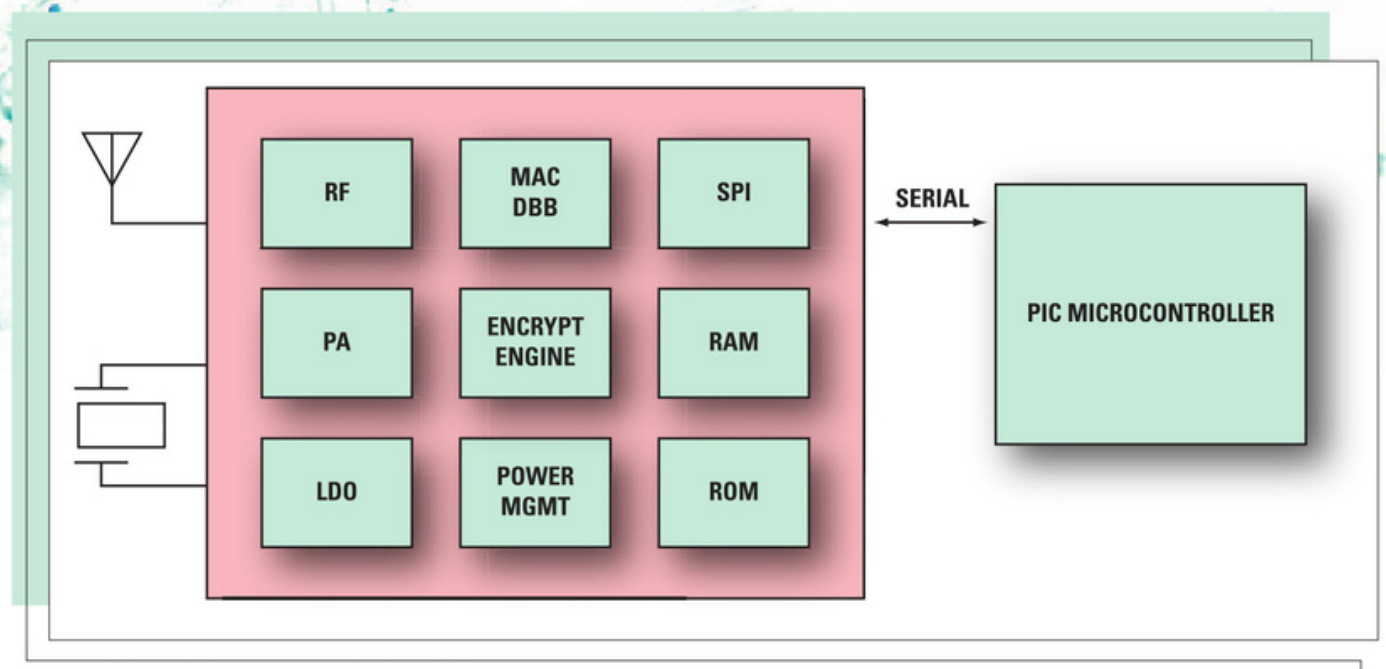
Estas son las características más interesantes sobre este módulo de conectividad:

- Trabaja con la estándar 802.11b de Wi-Fi.
- Tiene la dirección MAC pregrabada por defecto.
- Posee los componentes de RF y amplificadores de banda base para establecer la comunicación.
- Soporta protocolos de seguridad WEP, WPA y WPA2.
- Tiene bajo consumo de alimentación.
- Se alimenta a través de una fuente de 3,3 V.
- Presenta velocidades de 1 y 2 Mbps.
- Ofrece licencia gratuita de la pila (stack) TCP / IP para Internet.
- Incluye antena en el PCB.
- Sus dimensiones son mínimas: 21 mm x 31 mm.
- Cuenta con 36 pines de conexión.

Todas estas especificaciones hacen del módulo un elemento ideal para integrar conectividad en diferentes ambientes de la electrónica; por ejemplo, en sistemas de monitoreo y control; en industrias; en sistemas de computación; para realizar dispositivos inalámbricos como teclados, mouse y videocámaras; o en dispositivos electrónicos de consumo.

> SOBRE WI-FI

Wi-Fi es una marca de la Wi-Fi Alliance, una organización comercial dedicada a certificar los dispositivos que cumplen con los estándares de la IEEE 802.11 para redes inalámbricas de área local. El objetivo de esta organización es fomentar el uso de las redes inalámbricas de una manera simple y asegurar la compatibilidad entre los diferentes equipos.



> **Figura 32. El módulo de conectividad Wi-Fi permite la conexión con los microcontroladores a través de la comunicación SPI.**

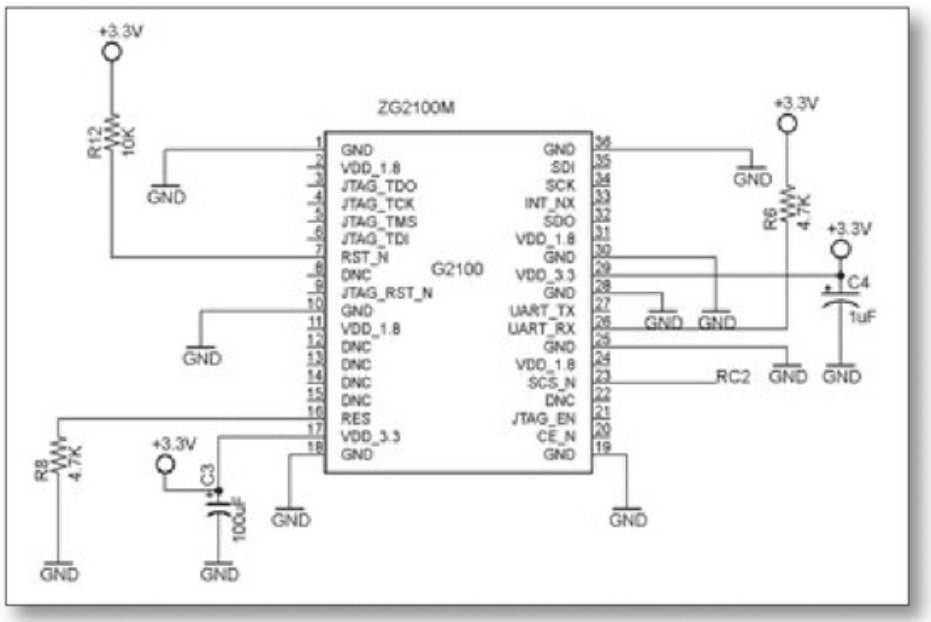


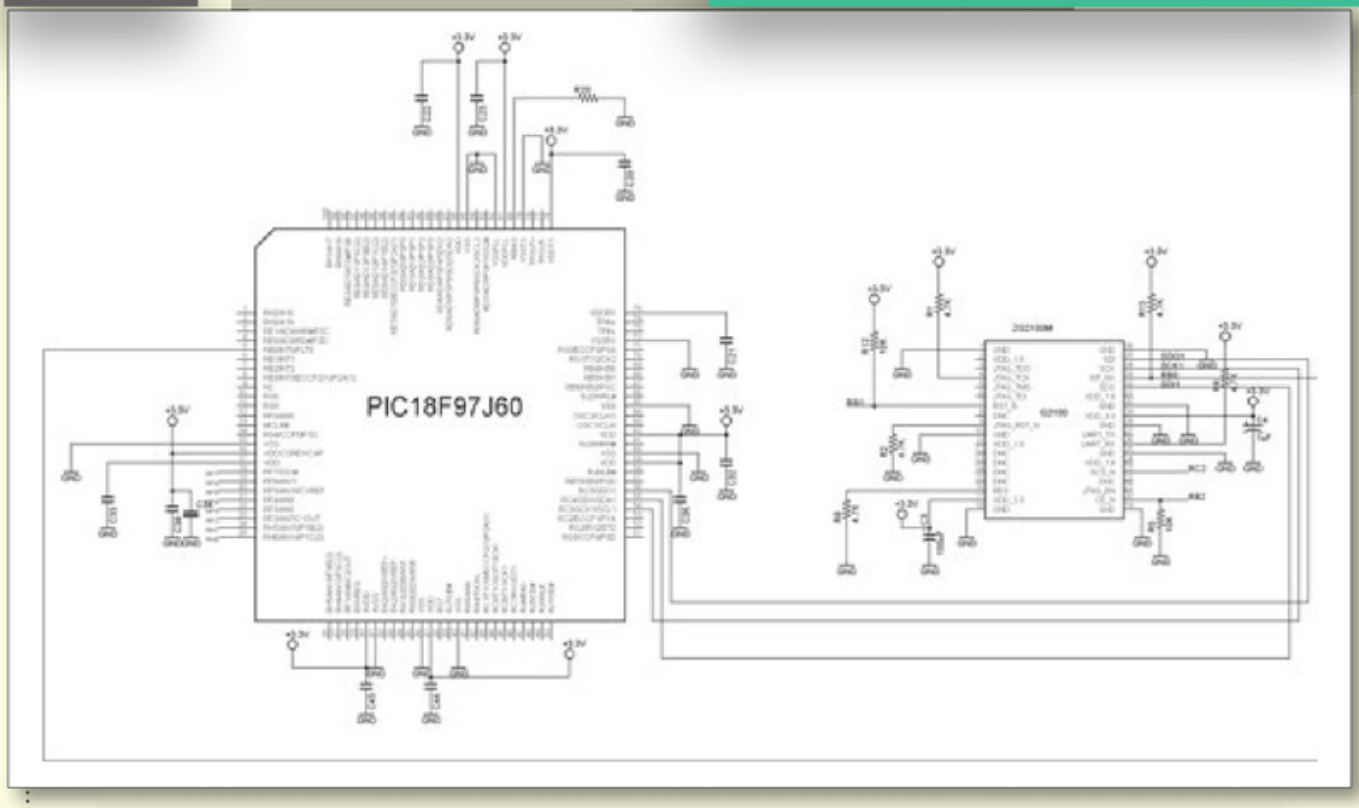
Figura 33. Conexiones de alimentación del módulo ZG2100.

Alimentación del módulo

El módulo se alimenta desde una simple fuente que entrega **3,3 V**, ya que internamente posee un regulador de tensión **LDO** (*low drop-out*), que genera las tensiones necesarias en el núcleo de procesamiento.
En este módulo contamos con una entrada directa opcional para una fuente de **1,8 V** que alimentará el núcleo del dispositivo. Sin embargo, podemos no utilizarla, gracias al regulador interno mencionado anteriormente. En la **Figura 4** podemos ver el diagrama de conexión del módulo **ZG2100M**. Otro aspecto para destacar es el bajo consumo que posee el dispositivo, que en modo **Sleep** consume **250 µA**, y en el de hibernación llega a **0,1µA**. Esto permite que sea alimentado desde una batería. Cuando transmite en su máxima potencia, el núcleo llega a consumir **230 mA** de la fuente de alimentación.

Comunicación con el módulo ZG2100M

Este módulo se caracteriza por la simplicidad en la programación, que se efectúa a través de **APIs** (*Application Programming Interface*) y no requiere de un sistema operativo.
En el manejo de la conexión, el módulo **ZeroG** controla la capa **MAC** y las capas de base, y se conecta al **microcontrolador que realizará el control del módulo (MCU Host)** por medio de un puerto SPI.
Los microcontroladores de **8**, **16** y **32** bits residentes en la placa de desa-rrollo se encargan del control de la pila de la red de TCP/IP y de correr las aplicaciones del sistema.
Actualmente, los módulos ZeroG incorporan soluciones Wi-Fi para microcontroladores PIC, porque se han integrado los **drivers Easy-FI** (estos permiten usar el módulo ZeroG de manera muy simple) en el stack de TCP/IP de microchip, de forma tal que no hay necesidad de descargar



> **Figura 34. Esquemático con las conexiones del módulo ZeroG al microcontrolador PIC18F97J60 a través de la comunicación SPI.**

un driver separado de Wi-Fi. Esta es una de las razones que vuelven tan atractiva la integración de este módulo en los desarrollos con PIC.

Conexión del módulo ZeroG por SPI

En el módulo ZG2100M hay dos interfaces SPI. Una está dedicada a las comunicaciones **SPI en modo esclavo**, en donde el dispositivo maestro controlará las acciones del módulo. La otra es un puerto **SPI maestro**, donde el módulo interactúa con una memoria externa tipo **ROM** para obtener un listado de las tareas por realizar. Nosotros utilizaremos la **interfaz SPI esclavo** para la comunicación entre el microcontrolador y el módulo.

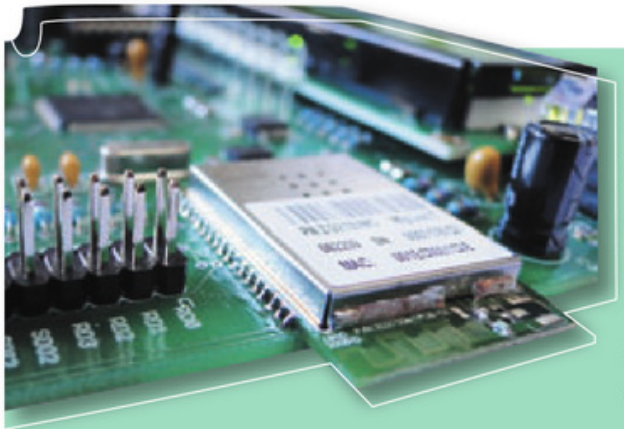
En la **Figura 34** vemos un esquemático donde se observa la conexión entre los dispositivos. La conexión será similar a los dispositivos SPI que estudiamos en la obra.

Interfaz SPI esclavo con interrupciones al host

Dijimos que la interfaz SPI esclavo se utiliza para la comunicación del módulo con el maestro, que en este caso, es el microcontrolador. Pero en este modo de comunicación se utiliza el pin **INT_NX** como pin de interrupción. Entonces, cuando los datos están disponibles para el maestro, la línea INT_NX del módulo se pone a nivel bajo. La línea INT_NX se pondrá en 1 después de que los datos sean transferidos al buffer del maestro. La velocidad máxima del reloj (CLK) es de **25 MHz**.

Control del módulo por el puerto

SPI. Cuando enviamos los comandos de control al módulo, debemos tener en claro que este trabaja en el formato **big endian.** Esto quiere decir que primero debemos enviar el bit más significativo y, luego, los menos significativos, como lo indica la comunicación por SPI. Los datos serán recogidos por el módulo después de que se active la señal de **Chip Select** correspondiente y la señal de reloj tenga un flanco de subida.



> CHIP SELECT

Debemos recordar que, haciendo uso del pin **Chip Select**, podemos conectar varios dispositivos en un mismo bus de comunicación. Esto nos permite tener el módulo de conectividad trabajando en el mismo bus que utiliza una memoria EEPROM SPI externa. De todas maneras, el microcontrolador PIC18 F97J60 cuenta con dos módulos SPI, por si queremos dedicar un puerto de comunicación solo para el módulo de conectividad.

Cuando operamos con el módulo ZG2100M, nos encontramos con dos regiones o espacios de trabajo que podemos manipular. Una es el espacio de acceso a los registros, que nos permite leer y modificar los registros internos del módulo. La otra está dedicada a la interfaz **FIFO** (*First In, First Out*, primero en entrar, primero en salir). Veremos más adelante en detalle cómo trabajar en estas zonas.

Acceso al espacio de registros

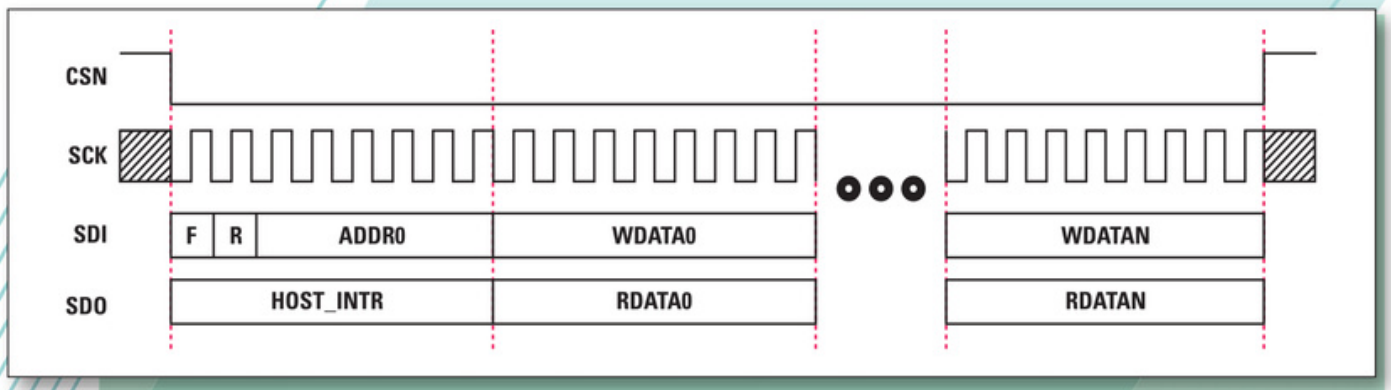
Desde el dispositivo maestro, tenemos acceso al espacio de registro del **módulo ZeroG,** para lo cual tenemos que enviar una **trama de datos** a través de la interfaz SPI (**Figura 35**). Cada trama de datos que enviemos debe mantener el pin **Chip Select** a nivel bajo, para habilitar el módulo y mantener la señal de clock funcionando. El primer byte que enviamos contiene la siguiente información:

- **F:** selecciona cuál es la zona a la que vamos a acceder, si es el espacio del FIFO o el de registros. Si este bit está en **1**, le estamos indicando al módulo que el acceso es hacia el **espacio FIFO**. En cambio, si transmitimos un **0**, indicamos que vamos a trabajar con el espacio de registros.
- **R:** es el bit de **lectura/escritura**. Si el bit se pone a **1**, indicamos que la operación es de lectura. En cambio, si colocamos un **0**, la operación es de escritura.
- **ADDR0:** es la dirección inicial para el envío de datos. Este valor solo se utiliza cuando accedemos al espacio de registro, y es ignorado cuando accedemos al FIFO.
- **WDATAn:** son los bytes de escritura. Solo se utilizan durante la escritura, mientras que en la operación de lectura son ignorados.
- **RDATAn:** son los bytes de lectura. Son siempre válidos, tanto en la lectura como en la escritura.
- **HOST_INTR:** es un registro de interrupción de 8 bits.

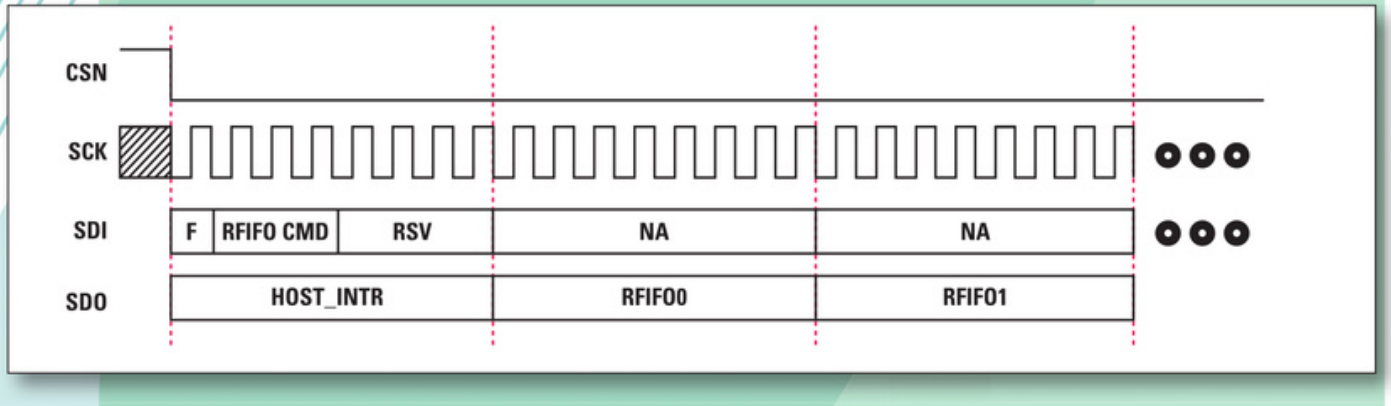
En el diseño de circuitos RF, necesitamos un software adecuado para estos dispositivos

Interfaz FIFO

En la **Figura 36** de la página 299, vemos el formato de la trama de datos que debemos enviar a la **interfaz FIFO** para leer o escribir datos. Básicamente, aparecen los mismos campos de control que en el acceso a los registros, salvo que en esta ocasión, vemos un campo de comandos (FCMD) y el campo **RSV** al inicio de la



> **Figura 35. Formato de las tramas para acceder al espacio de registros del módulo ZeroG.**



trama. FCMD especifica la tarea que queremos realizar con la pila y tiene un tamaño de 3 bits. Los comandos básicos que podemos enviar al módulo pueden verse en el **Código 18.** El campo **RSV** está reservado. Para obtener más información acerca de cómo comunicarse con el módulo, tendremos que consultar en las hojas de datos del dispositivo.

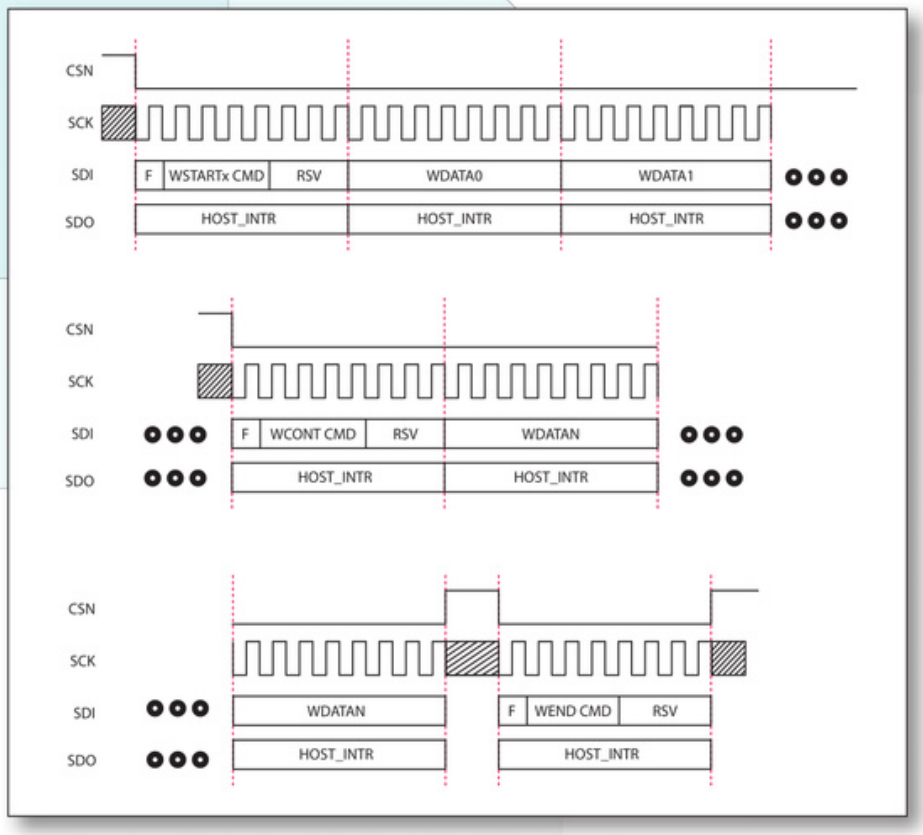
> **Código 18**

```
-FCMD[2:0] :
0x0 – RFIFO_CMD: comando de lectura
0x1 – WCONT: Continuar con la escritura
0x2 – WSTART0: Comenzar escritura de paquetes
0x3 – WSTART1: Comenzar escritura de paquetes
0x4 – WEND CMD: Fin de la escritura
0x5 – REND CMD: Fin de la lectura.
```

Consideraciones de layout

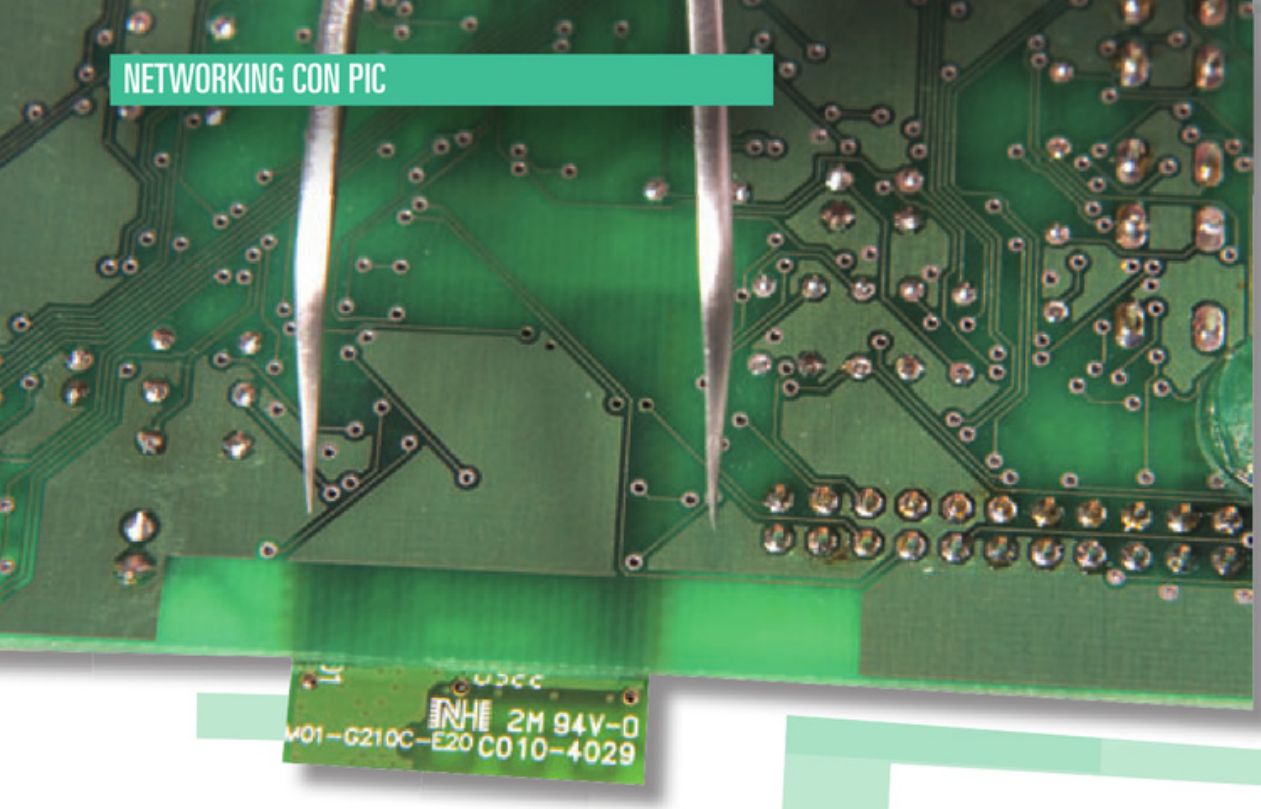
Cuando colocamos dispositivos de **RF** en nuestros desarrollos, debemos tomar importantes precauciones a la hora de diseñar el **layout**. Debido a que los sistemas RF trabajan en altas frecuencias, el diseño de las pistas de las placas influye en el funcionamiento del dispositivo. Este diseño afecta a los sistemas RF por los efectos de reflexión que pueden existir en una línea, y que dependen de las discontinuidades de la impedancia que puede mostrar hacia el dispositivo. También es preciso considerar que los dispositivos que se encuentran alrededor de un módulo RF pueden ocasionar interferencias en nuestras aplicaciones. Por este motivo, debemos asegurarnos de que aquellos que emiten un campo magnético, como bobinas y transformadores, estén alejados una cierta distancia del módulo. Entonces, cuando trabajamos en el diseño de circuitos RF, necesitaremos contar con las herramientas y el software adecuados al diseño de estos dispositivos.

> **Figura 36. Formato de las tramas de escritura hacia la interfaz FIFO.**





> **Figura 37. Ejemplo de cómo colocar el módulo de conectividad Wi-Fi para evitar las interferencias indeseadas.**



> Figura 38. Las pistas y los planos de masa pueden interferir en el correcto funcionamiento de la antena; por eso se delimita una zona donde no colocamos dispositivos.

En el mercado encontramos muchas empresas que se dedican a solucionar los problemas de interferencia. También debemos tener en cuenta que todo equipo que emita radiación electromagnética está regulado por normas para evitar que interfieran con otros dispositivos y proteger al usuario del efecto nocivo de estas radiaciones.
Debemos estar atentos a todas estas consideraciones en el momento de insertar el módulo de conectividad Wi-Fi, ya que trabaja en zonas de altas frecuencias.
Pero, además, tenemos la ventaja de que el **módulo ZG2100M** incorpora en su PCB todos los componentes de RF y amplificadores de banda base para establecer la comunicación, lo que nos evita estar lidiando con el diseño tan complicado de las pistas de RF. Solo debemos tomar algunas precauciones que nos indica el fabricante para colocar el módulo sobre el PCB.

Ejemplo de diseño del PCB

En la **Figura 37** de la página 299 podemos ver un ejemplo de cómo colocar el módulo sobre el PCB. En esa zona, el fabricante indica que no se deben trazar pistas por debajo del módulo, y tampoco deben encontrarse dispositivos cercanos.

En la **Figura 39** tenemos una visión de la parte inferior del módulo, donde podemos ver algunos puntos de prueba que coloca el fabricante y varias vías que interconectan las capas del PCB. Debemos asegurarnos de que las vías que queden debajo del módulo tengan una máscara antisoldante.

La **Figura 38** muestra una zona del PCB donde no se coloca ninguna pista y tampoco, ningún plano de masa. Como esta zona es muy sensible a las interferencias por la proximidad de la antena, debemos tomar esta precaución al momento de diseñar el PCB que alojará el módulo. ■



> Figura 39. Vista de la parte inferior del módulo ZeroG, donde podemos apreciar varios puntos de prueba colocados por el fabricante.

CONFIGURACIÓN DEL STACK TCP/IP PARA WI-FI

Ahora que conocemos a fondo el módulo ZeroG, veremos cómo configurar el stack de Microchip para agregar conectividad Wi-Fi a nuestro servidor web.

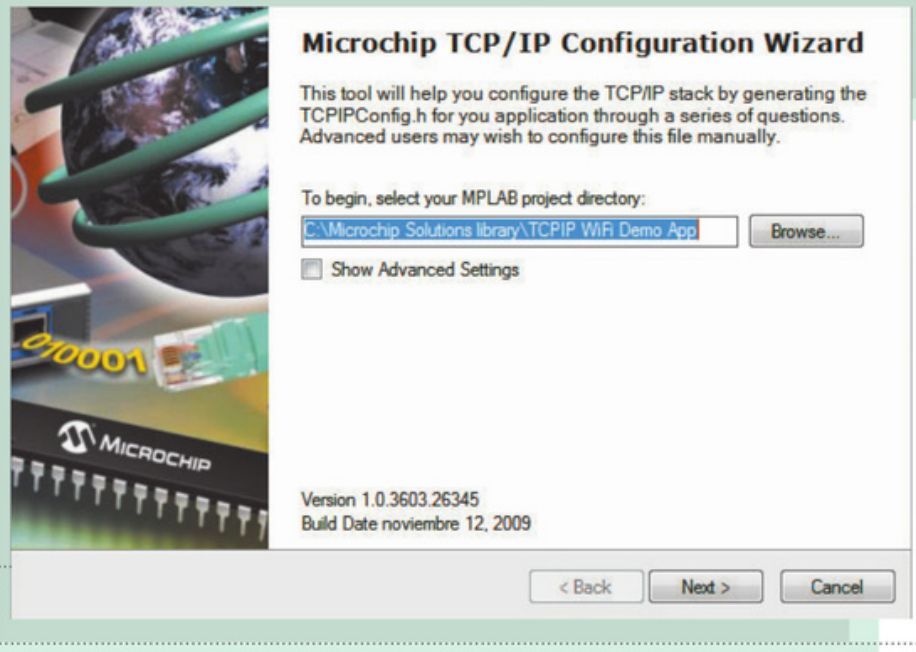
Descubrimos cuáles son las ventajas de contar con conectividad inalámbrica y las diferentes tecnologías que brindan soluciones a esta necesidad, como **ZigBee**, **Bluetooth** o **Wi-Fi**. Particularmente, las redes Wi-Fi han extendido su espacio de uso, y hoy es posible encontrarlas en aplicaciones auxiliares, como sensores, sistemas de seguridad integrados, electrodomésticos inteligentes y una gran variedad de aplicaciones relacionadas con dicho mundo. En este esfuerzo por agregar conectividad en sus aplicaciones, Microchip ofrece para sus líneas de microcontroladores soluciones basadas en esta familia de productos **Wi-Fi I/O** de **ZeroG**.

Microchip Configuration Wizard para Wi-Fi

En clases anteriores vimos cómo utilizar esta herramienta para configurar los distintos aspectos y funciones que presenta el **stack de TCP/IP** para los microcontroladores PIC. Ahora, estudiaremos de qué manera configurarlo para que nuestras aplicaciones cuenten con conectividad Wi-Fi. Comenzamos ejecutando el programa Microchip Configuration Wizard y procedemos a configurar el stack de TCP/IP. Primero, indicamos el directorio donde se encuentra el proyecto general. Si descargamos la librería de soluciones de Microchip, podemos buscarlo donde se instala por defecto la carpeta **[TCPIP WiFi Demo App]**. Allí encontramos los archivos fuente y las plantillas de las páginas web que emplearemos. Recordemos que la elección de las distintas funciones que puede ejecutar el stack TCP/IP de Microchip se realiza desde el archivo **TCPIPConfig.h**. La utilidad del programa Microchip Configuration Wizard reside en la facilidad de llevar a cabo estos cambios por medio de una interfaz gráfica, pero en realidad, no se está haciendo otra cosa que modificar este archivo. Podemos ver el contenido de este archivo ingresando en la carpeta **TCPIP WiFi Demo App**.

> Las redes Wi-Fi son cada vez más seguras, veloces y sencillas de montar.

> Figura 40. Ejecutamos el Microchip Configuration Wizard para configurar el stack de TCPIP, y lo haremos sobre el proyecto TCPIP WiFi Demo App.



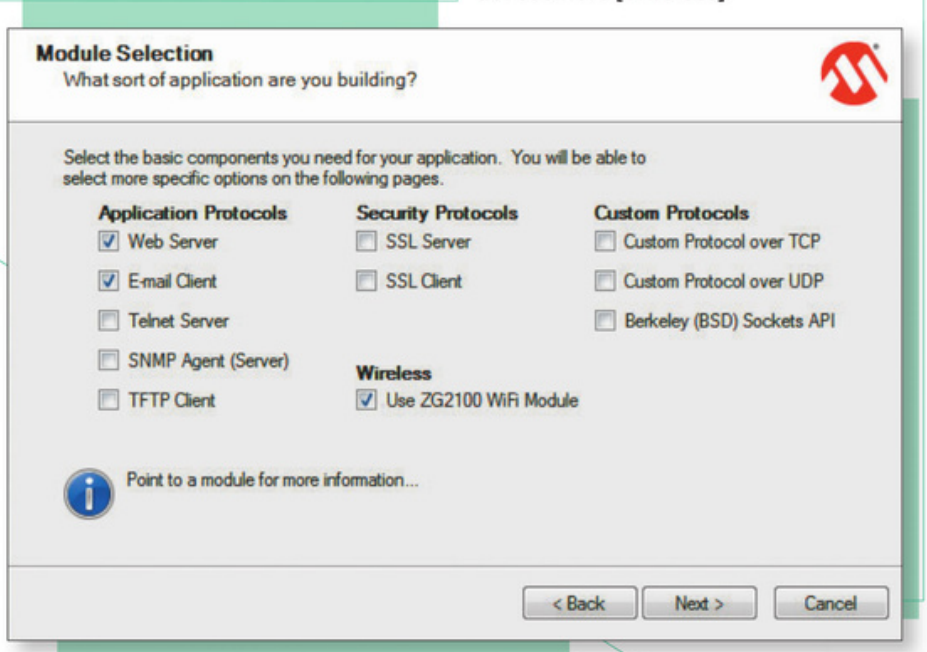
A continuación, nos encontramos con la página donde podemos seleccionar las diferentes funciones que soporta el **stack**. Allí debemos asegurarnos de que estén seleccionadas las pestañas para las funciones de servidor web y el cliente de e-mail. Luego, en la opción de conectividad inalámbrica (wireless), elegimos el módulo que venimos estudiando, el **ZG2100**. Es importante observar que, a diferencia de lo que hacíamos con Ethernet, el microcontrolador solo se encarga de transportar el paquete en los protocolos de niveles superiores, y el módulo **Wi-Fi** de las capas 1 y 2 del modelo OSI; es decir, que tiene el manejo de la capa de enlace y de la capa física. En la próxima pantalla de configuración, seleccionamos **[TCP Performant Test]**, que es la utilidad que nos permite generar **pings**, de gran practicidad a la hora de encontrar errores en las redes.

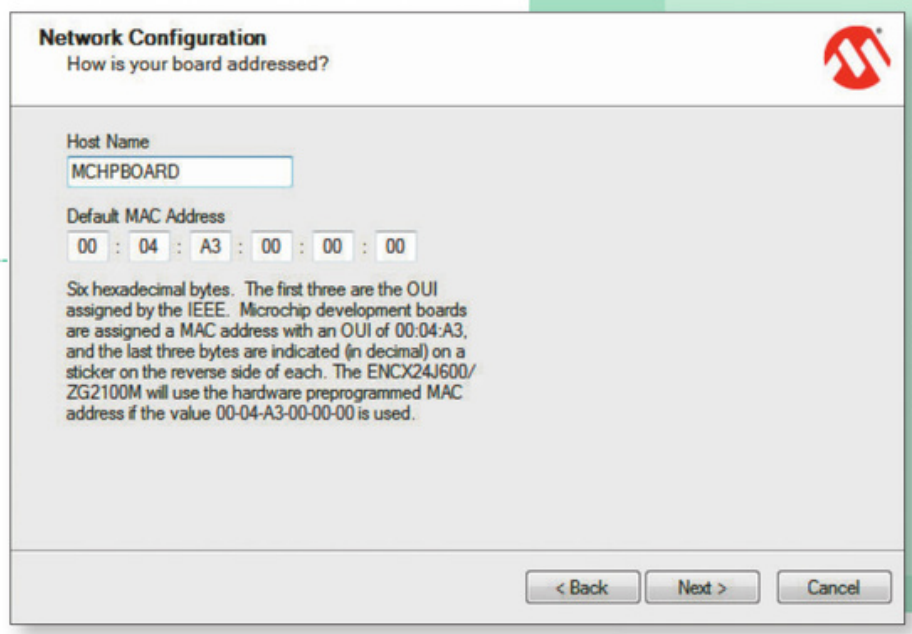
Asignación de la dirección MAC pregrabada

En el siguiente cuadro de diálogo, indicamos el nombre de host con el cual se identificará el servidor en la red donde esté trabajando. Luego, tenemos que señalar la dirección MAC del dispositivo. Recordemos que esta debe ser única para nuestro equipo; de lo contrario, podrían generarse errores en la red. En este aspecto debemos destacar cuáles son las fuentes donde el stack puede tomar una dirección MAC. Básicamente, existen **tres**. Una de ellas es la dirección **MAC** del módulo **ZeroG**, grabada desde fábrica, de modo que

La dirección MAC que se encuentra grabada en la EEPROM es la que tiene mayor prioridad.

>Figura 41. Selección de las funciones que se utilizarán del stack y activación del casillero [Wireless].





>Figura 42. Asignación de la dirección MAC que utilizará el dispositivo; recordemos que debe ser única en la red.

nos garantiza que sea única. Recordemos que las direcciones MAC se encuentran organizadas de modo tal de evitar que dos dispositivos tengan la misma. Es así que la **IEEE** asigna a los fabricantes los primeros **tres octetos** (24 bits) que componen la MAC, llamados **OUI** (*Organizationally Unique Identifier*). La MAC que posee el módulo **ZeroG** puede verse en la etiqueta ubicada en la parte superior de este.
Si la revisamos, podemos ver que la **OUI** que identifica a los dispositivos de la empresa ZeroG es la siguiente: **00:1E:C0**. Cada OUI garantiza al fabricante la posibilidad de registrar **2^24** dispositivos. Otro ejemplo es la OUI de las herramientas de Microchip, que es **00:04:A3**, y que es la que se muestra por defecto en el **Configuration Wizard**.
Una **segunda fuente** desde la cual el stack TCP/IP puede obtener una dirección MAC es una imagen de código programado. La **tercera** es un valor que se almacena en la memoria EEPROM.
En tiempo de ejecución, el stack crea una estructura en memoria **RAM** donde se almacena la dirección MAC válida que se utilizará en el período de sesiones. Entonces, el código revisa si hay una estructura de datos válidos en la **EEPROM**. Si existe, se utilizan estos valores, y se ignoran los que puedan estar programados dentro del código en tiempo de compilación.
Pero si no se encuentran la MAC en la EEPROM, se emplea la que se coloca durante la configuración del stack, es decir, la especificada en el **Configuration Wizard**. Si el valor que colocamos en esa zona del cuadro de diálogo es **00:04: A3**: **00:00:00**, le estamos indicando al stack que la MAC por utilizar es la que está grabada en el módulo de **conectividad ZG2100**. De lo contrario, se utilizará la que le indiquemos en la configuración del stack.

> SOBRE PROTOCOLOS

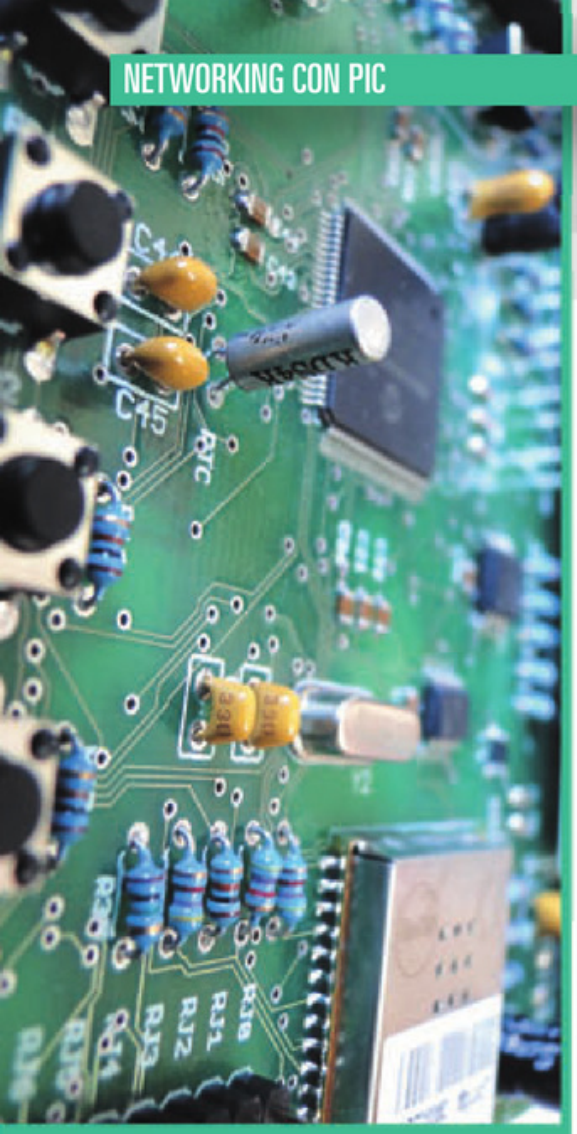
La mayoría de los protocolos que trabajan en la capa 2 del modelo OSI usan una de las tres numeraciones manejadas por el IEEE: **MAC-48, EUI-48 y EUI-64**. Estas han sido diseñadas para ser identificadores globalmente únicos.
No todos los protocolos de comunicación usan direcciones MAC, y no todos los protocolos requieren identificadores globalmente únicos.

Prioridad de las direcciones

En términos de prioridad de las direcciones que se utilizarán para el dispositivo, la **MAC** que se encuentra grabada en la **EEPROM** es la primera, y ningún otro valor tendrá precedencia sobre ella.

>Figura 43. En la imagen vemos la dirección MAC del módulo, especificada en la etiqueta.





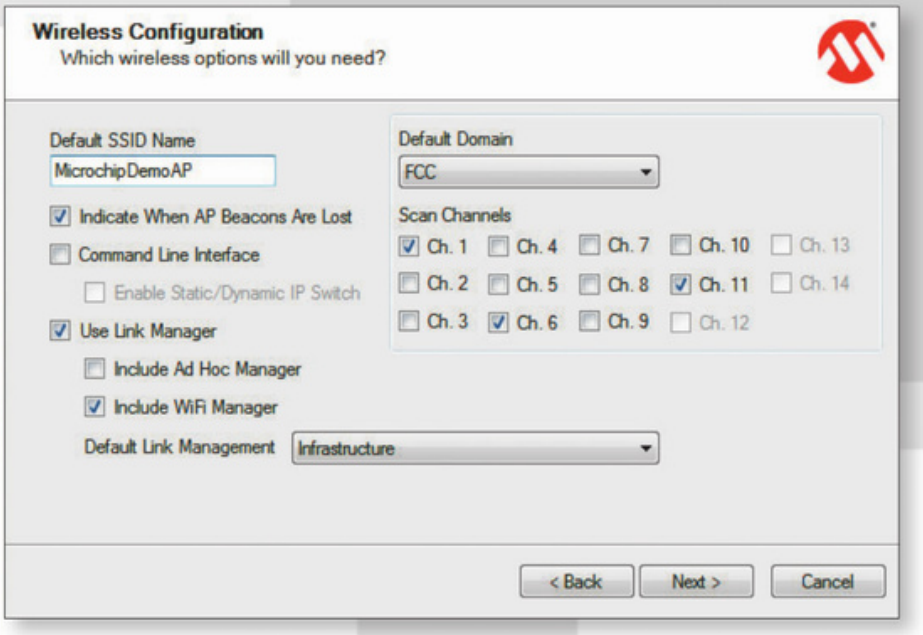


Figura 44. Cuadro de diálogo donde colocaremos el nombre de la red a la que se conectará la aplicación.

Conexión a una red Wi-Fi existente

En el siguiente cuadro de diálogo del Configuration Wizard, tenemos las opciones para la conexión a la red inalámbrica. En **SSID** (*Service Set IDentifier*) colocamos el nombre de la red a la que se conectará el servidor web. Si ya existe una red, debemos indicar aquí el nombre que la identifica, dato disponible desde el **access point** de la red. Si no tenemos una red preexistente, colocamos el nombre que queremos darle. En este lugar, también podemos indicar cuál es la estructura de la red a la cual se va a conectar. Habíamos visto que teníamos, básicamente, dos estructuras para la conexión: ad-hoc y de infraestructura. El módulo permite la conexión en ambos tipos de redes. Por defecto, el stack viene configurado para conectarnos a una red de infraestructura. Si en realidad queremos conectarnos a una red ad-hoc, debemos seleccionar **Include Ad-HOC Manager.**

> CIFRADO DE DATOS

Un muy elevado porcentaje de redes inalámbricas es instalado sin tener en consideración la seguridad, con lo cual quedan abiertas (o completamente vulnerables a los crackers), sin posibilidad de proteger la información que por ellas circula. Los sistemas de seguridad más comunes utilizan protocolos de cifrado de datos, que se encargan de codificar la información transmitida para proteger su confidencialidad.

Canal de radiofrecuencia

En la pantalla (**Figura 44**) podemos elegir el canal de radiofrecuencia que utilizará el módulo. En la imagen vemos que, por defecto, están seleccionados tres canales: 1, 6 y 11. El módulo ZeroG realizará, entonces, el análisis sobre ellos para detectar actividades de RF. Es importante saber que, a mayor cantidad de canales, más se demorará el módulo en conectarse.

Seguridad y encriptación de datos

En el cuadro de diálogo **Wireless Security** podemos adicionar seguridad a la conexión que establecerá nuestra aplicación con el access point. Podemos elegir entre WEP o WPA.

Seguridad WEP

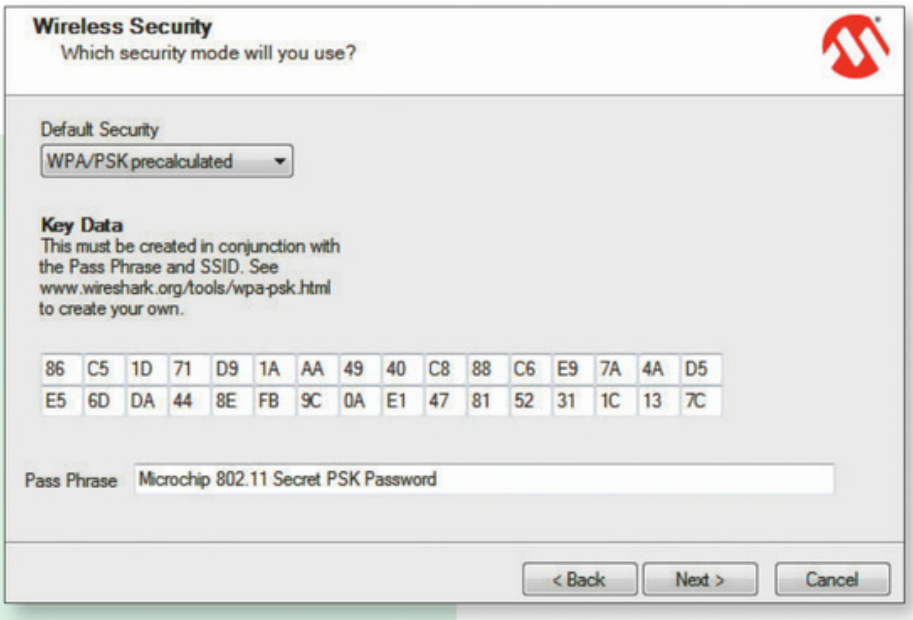
WEP (*Wired Equivalent Privacy*) es un mecanismo de encriptación para las redes inalámbricas. La mayoría de los access points aplican dos tipos de seguridad WEP: la **de 64 bits** utiliza una clave de **40 bits** (WEP-40), en tanto que la de **128 bits** emplea

una clave de **104 bits** (WEP-104). En la forma más básica, las claves WEP son valores hexadecimales de solo 5 bytes para WEP-40 y de 13 bytes para WEP-104. Algunos access points tratan de incrementar el carácter aleatorio de la clave WEP mediante la adición de una capa extra que convertirá una contraseña **ASCII** en una clave **hexadecimal**. El módulo ZeroG requerirá una clave hexadecimal, no importa qué camino se utilice para generarla. Para habilitar la seguridad WEP entre la placa de aplicación y el access point, habrá que configurar este último para dicha seguridad, y el stack de la placa de desarrollo, con las claves WEP correctas.

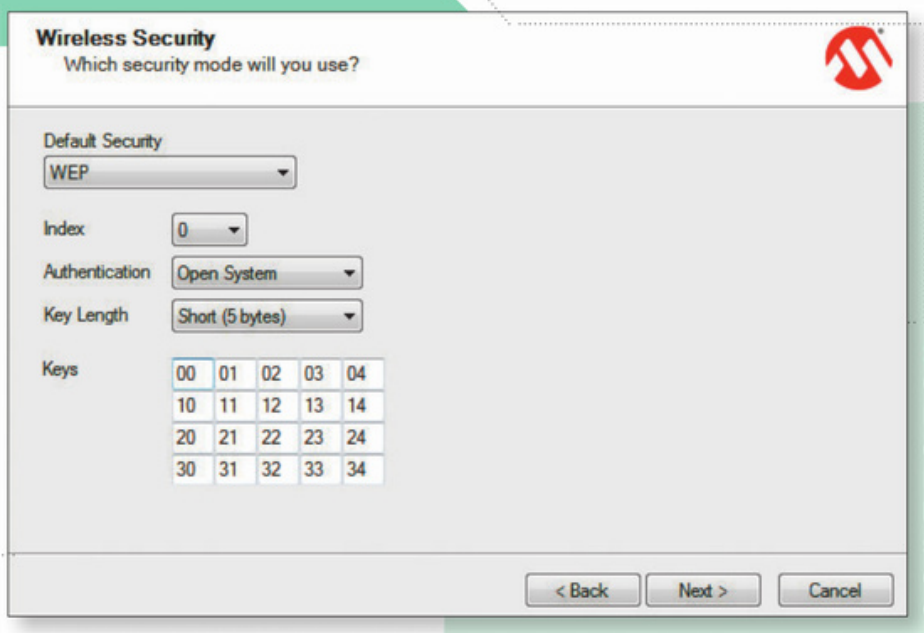
Seguridad WPA/WPA2

WPA y **WPA2** son modos de seguridad que implementan la especificación **802.11i**. Son más seguros que las redes de encriptación WEP. Utilizan el SSID y la contraseña de usuario para generar el **PSK** (*Pre-Shared Key*), donde las estaciones y el access point tienen, previamente, una clave compartida. El módulo **ZeroG** posee dos maneras diferentes de calcular la **PSK** para su uso en WPA/WPA2. En una de ellas, el usuario final puede programar la clave en hexadecimal de **32 bytes** en el código fuente o en el Configuration Wizard. En la otra, el módulo ZeroG puede calcular la clave hexadecimal de 32 bytes internamente. Debemos tener en cuenta que si el módulo ZeroG tiene que calcular la clave, esto puede añadir un tiempo extra de 30 a 45 segundos en el tiempo de conexión, para dar lugar a que el chip calcule el valor. ■

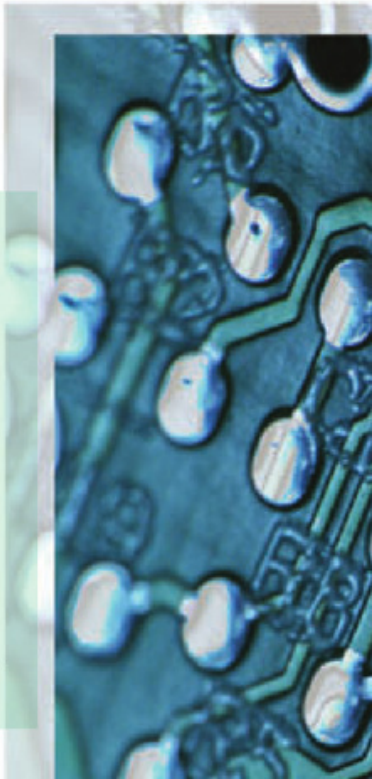
Por defecto, el stack viene configurado para conectarnos a una red de infraestructura.



> **Figura 45. En la seguridad WPA podemos indicarle el valor de la clave o decirle al módulo que lo calcule.**



> **Figura 46. Configuración para la seguridad WEP, donde se definen las claves de 5 bytes. Estas deben ser las mismas que posee el router o access point.**



ACTIVIDADES

TEST DE AUTOEVALUACIÓN

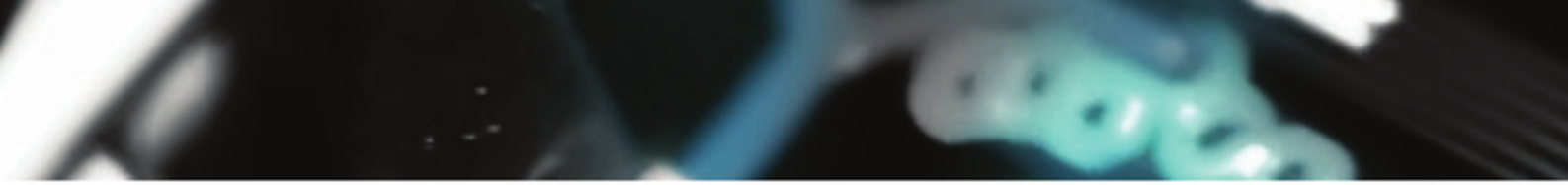
1. ¿Cuáles son las capas que componen a Microchip TCP/IP stack?
2. ¿Qué es un cliente DHCP?
3. Enumere los archivos principales del stack.
4. ¿Para que sirven las funciones callback?
5. ¿Qué es el sistema de archivos MPFS?
6. ¿Qué es Microchip Web Preview?
7. Mencione las características del protocolo SMTP.
8. ¿Cuál es la función de un puente serie-Ethernet?
9. ¿Qué es PIC18F97J60?
10. Caracterice a la telemetría.

EJERCICIOS PRÁCTICOS

1. Agregue nuevos servicios al stack.
2. Deshabilite servicios en forma manual.
3. Realice la configuración de un web server.
4. Utilice una función callback.
5. Envíe un e-mail con SMTP.



ÍNDICE TEMÁTICO



Índice temático

A

Acceso a los bancos 107
Access bank 104
Alarma térmica 144
Algoritmos 206
Aquitectura Hardvard 64
Aritmética binaria 24
Arquitectura Von Neumann 63

B

Banco virtual 107
Binario a decimal 17
Bluetooth 172
Buffers 23
Bus de control 53
Bus de datos 53
Busy flag 147

C

Caracteres del LCD 137
CBLOCK 93
Celda de memoria 38
Ciclo de máquina 58
Circuito de reset 102
Circuitos lógicos 26
Circuitos secuenciales 30
Clases de redes 223
Código RC5 187
Comandos del LCD 136
Compuertas lógicas 19
Comunicación de aplicaciones 176
Concepto de módulo 15
Conectividad inalámbrica 172
Conectividad no inalámbrica 153
Conexión punto a punto 177
Contador de programa 54
Contador de un dígito 85
Contador en anillo 33
Contador Johnson 34
Contador PWM 126
Contadores 31
Control de contraste 205
Control por infrarrojos 186
Control por RF 184
Conversor A/D 70
Conversor de doble rampa 73
Conversor estático 72
Conversor paralelo 72
Cristal líquido 142

D

Demultiplexores 35
Dígitos binarios 19
Dinamic C 195
Directivas 113
Display de 7 segmentos 124
Display LCD 142
División de números binarios 25
DNS 224
Docklight 166

E

EEPROM 41
Ensamblador 57
Escritura de una memoria 39

F

Flip-flop T 29
Flip-flops 27
FT232 169



G

GPS 198
Grabador de PICS 97

I

ICSP 118
Interrupciones 66

L

LCD inteligente 135
Lectura de reportes 183
Lenguajes de alto nivel 59
Lógica combinacional 12
LPV 118

M

Mapa de memoria 55
Matriz activa 142
Memoria de datos 104
Memoria PROM 43, 44
Memoria ROM 42
Memorias DRAM
Memorias Flash 47
Memorias RAM 48
Memorias volátiles 41
Memorias RAM estáticas 49
Micro 6502 65
Microcontroladores 52
Microprocesador 53, 60
Modelo OSI 219
Modo API 182
Modos SPI 157
Módulos prearmados 177
Módulos RABBIT 192
MPLAB 92
MPLAB C18 110
MPLABSIM 94
Multiplexores 35
Mundo RABBIT 200

O

Operaciones binarias 14
Organización matricial 39
Oscilador 79
oscilador externo 79
Oscilador interno 79

P

Perfiles DUN 188
Periféricos de entrada 56
Periféricos externos 123
PIC16 105
PIC16F 78
PIC18 105
Pines de control 39
Placas entrenadoras 134
PLL interno 102
PLOT 206
Programador MCE 117
Protocolo IP 223
Protocolos de conectividad 152
Puerto serie 154
Pulsadores 123
Punto a multipunto 178

R

Rabbit 60
Registro BSR 106
Registro de desplazamiento 37
Registro de estado 84

Registro LAT 108
Registro PORT 108
Registro TRIS 108
Registro W 82
Registros 36
Regla posicional 13
Relay 123
Reloj de tiempo real 199
Representación con signo 14
Resta de números binarios 24
RS-232 152

S

Sensor de temperatura 144
Sensores ZigBee 182
Servidores web 228
Simular protocolo serie 166
Sistema binario 14
Sistema hexadecimal 16
Sistema secuencial 12
Sistemas de numeración 12
Sistemas digitales 12
SPI 156
Status 106
STN negativo 143
Suma de BCD 18
Suma de números binarios 24

T

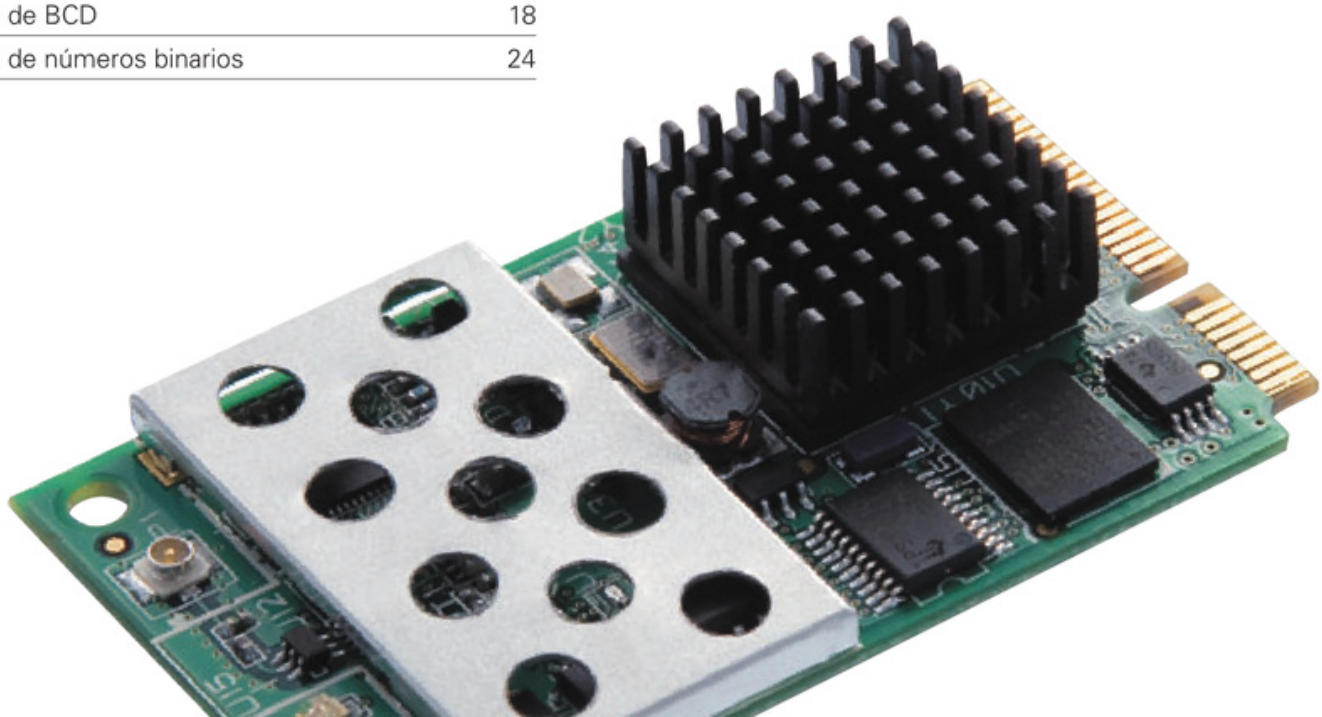
Tabla de verdad 19
Tamaño de punteros 113
TCP/IP 222
Tecnología HID 168
Terminal RBO 109
TFT 142
Timers 132
Tipos de datos enteros 111
Token 218
Topología árbol 217
Topología bus 217
Topología malla 217

U

UDP 226
Unidades de entrada y salida 56
Universal serial bus 167

V

Valores binarios 15
Valores lógicos 19
Variables compartidas 196
Variables protegidas 195



CLAVES PARA COMPRAR UN LIBRO DE COMPUTACIÓN

1 SOBRE EL AUTOR Y LA EDITORIAL

Revise que haya un cuadro "sobre el autor", en el que se informe sobre su experiencia en el tema. En cuanto a la editorial, es conveniente que sea especializada en computación.

2 PRESTE ATENCIÓN AL DISEÑO

Compruebe que el libro tenga guías visuales, explicaciones paso a paso, recuadros con información adicional y gran cantidad de pantallas. Su lectura será más ágil y atractiva que la de un libro de puro texto.

3 COMPARE PRECIOS

Suele haber grandes diferencias de precio entre libros del mismo tema; si no tiene el valor en tapa, pregunte y compare.

4 ¿TIENE VALORES AGREGADOS?

Desde un sitio exclusivo en la Red, un Servicio de Atención al Lector, la posibilidad de leer el sumario en la Web para evaluar con tranquilidad la compra, y hasta la presencia de adecuados índices temáticos, todo suma al valor de un buen libro.

5 VERIFIQUE EL IDIOMA

No sólo el del texto; también revise que las pantallas incluidas en el libro estén en el mismo idioma del programa que usted utiliza.

usershop.redusers.com

VISITE NUESTRO SITIO WEB

» Vea información más detallada sobre cada libro de este catálogo.
» Obtenga un capítulo gratuito para evaluar la posible compra de un ejemplar.
» Conozca qué opinaron otros lectores.
» Compre los libros sin moverse de su casa y con importantes descuentos.
» Publique su comentario sobre el libro que leyó.
» Manténgase informado acerca de las últimas novedades y los próximos lanzamientos.

TAMBIÉN PUEDE CONSEGUIR NUESTROS LIBROS EN KIOSCOS O PUESTOS DE PERIÓDICOS, LIBRERÍAS, CADENAS COMERCIALES, SUPERMERCADOS Y CASAS DE COMPUTACIÓN.



LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com // usershop@redusers.com

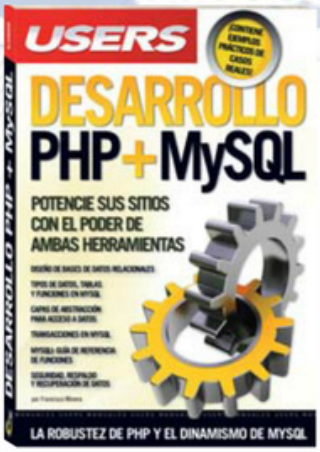
usershop.redusers.com



Windows 7: Trucos y secretos

Este libro está dirigido a todos aquellos que quieran sacar el máximo provecho de Windows 7, las redes sociales y los dispositivos ultraportátiles del momento. A lo largo de sus páginas, el lector podrá adentrarse en estas tecnologías mediante trucos inéditos y consejos asombrosos.

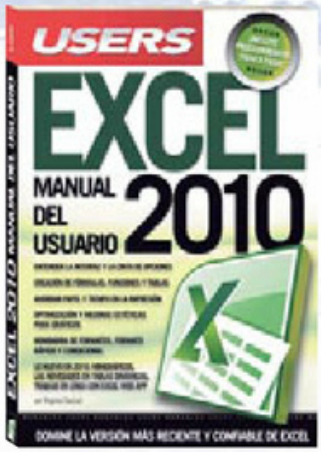
→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-17-6



Desarrollo PHP + MySQL

Este libro presenta la fusión de dos de las herramientas más populares para el desarrollo de aplicaciones web de la actualidad: PHP y MySQL. En sus páginas, el autor nos enseñará las funciones del lenguaje, de modo de tener un acercamiento progresivo, y aplicar lo aprendido en nuestros propios desarrollos.

→ COLECCIÓN: MANUALES USERS
→ 432 páginas / ISBN 978-987-1773-16-9



Excel 2010

Este manual resulta ideal para quienes se inician en el uso de Excel, así como también para los usuarios que quieran conocer las nuevas herramientas que ofrece la versión 2010. La autora nos enseñará desde cómo ingresar y proteger datos hasta la forma de imprimir ahorrando papel y tiempo.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-15-2



Técnico Hardware

Esta obra es fundamental para ganar autonomía al momento de reparar la PC. Aprenderemos a diagnosticar y solucionar las fallas, así como a prevenirlas a través del mantenimiento adecuado, todo explicado en un lenguaje práctico y sencillo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-14-5



PHP Avanzado

Este libro brinda todas las herramientas necesarias para acercar al trabajo diario del desarrollador los avances más importantes incorporados en PHP 6. En sus páginas, repasaremos todas las técnicas actuales para potenciar el desarrollo de sitios web.

→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-1773-07-7



AutoCAD

Este manual nos presenta un recorrido exhaustivo por el programa más difundido en dibujo asistido por computadora a nivel mundial, en su versión 2010. En sus páginas, aprenderemos desde cómo trabajar con dibujos predeterminados hasta la realización de objetos 3D.

→ COLECCIÓN: MANUALES USERS
→ 384 páginas / ISBN 978-987-1773-06-0

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



Windows 7 Avanzado

Esta obra nos presenta un recorrido exhaustivo que nos permitirá acceder a un nuevo nivel de complejidad en el uso de Windows 7. Todas las herramientas son desarrolladas con el objetivo de acompañar al lector en el camino para ser un usuario experto.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1773-08-4



Photoshop

En este libro aprenderemos sobre las más novedosas técnicas de edición de imágenes en Photoshop. El autor nos presenta de manera clara y práctica todos los conceptos necesarios, desde la captura digital hasta las más avanzadas técnicas de retoque.

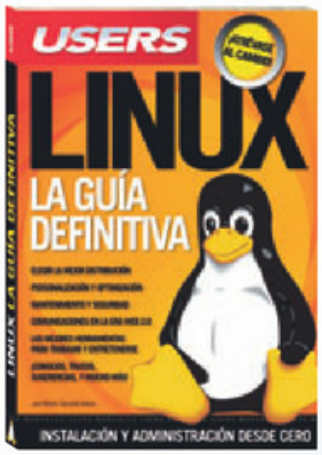
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-05-3



Grabación y producción de música

En este libro repasaremos todos los aspectos del complejo mundo de la producción musical. Desde las cuestiones para tener en cuenta al momento de la composición, hasta la mezcla y el masterizado, así como la distribución final del producto.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1773-04-6



Linux

Este libro es una completa guía para migrar e iniciarse en el fascinante mundo del software libre. En su interior, el lector conocerá las características de Linux, desde su instalación hasta las opciones de entretenimiento, con todas las ventajas de seguridad que ofrece el sistema.

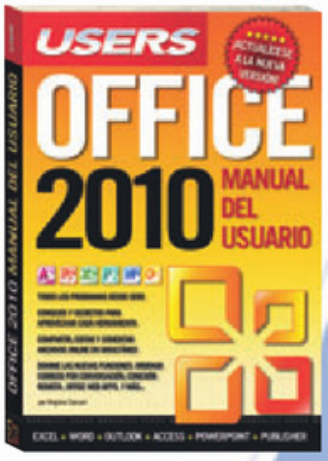
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-8-6



Premiere + After Effects

Esta obra nos presenta un recorrido detallado por las aplicaciones audiovisuales de Adobe: Premiere Pro, After Effects y Soundbooth. Todas las técnicas de los profesionales, desde la captura de video hasta la creación de efectos, explicadas de forma teórica y práctica.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-9-3



Office 2010

En este libro aprenderemos a utilizar todas las aplicaciones de la suite, en su versión 2010. Además, su autora nos mostrará las novedades más importantes, desde los minigráficos de Excel hasta Office Web Apps, todo presentado en un libro único.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-26013-6-2



Excel Paso a Paso

En esta obra encontraremos una increíble selección de proyectos pensada para aprender, mediante la práctica, la forma de agilizar todas las tareas diarias. Todas las actividades son desarrolladas en procedimientos paso a paso de una manera didáctica y fácil de comprender.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-26013-4-8



C#

Este libro es un completo curso de programación con C# actualizado a la versión 4.0. Ideal tanto para quienes desean migrar a este potente lenguaje, como para quienes quieran aprender a programar desde cero en Visual Studio 2010.

→ COLECCIÓN: MANUALES USERS
→ 400 páginas / ISBN 978-987-26013-5-5



200 Respuestas: Seguridad

Esta obra es una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos contestar para conseguir un equipo seguro. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

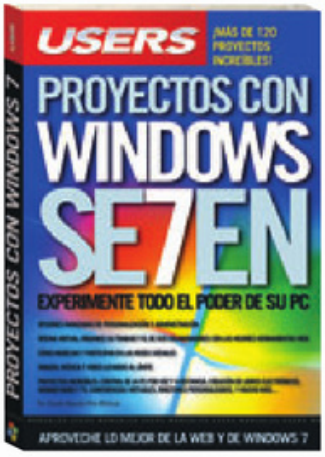
→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-26013-1-7



Funciones en Excel

Este libro es una guía práctica de uso y aplicación de todas las funciones de la planilla de cálculo de Microsoft. Desde las funciones de siempre hasta las más complejas, todas presentadas a través de ejemplos prácticos y reales.

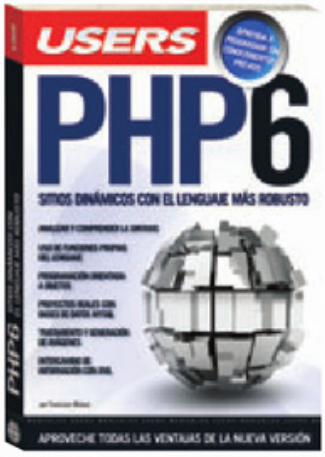
→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-26013-0-0



Proyectos con Windows 7

En esta obra aprenderemos cómo aprovechar al máximo todas las ventajas que ofrece la PC. Desde cómo participar en las redes sociales hasta las formas de montar una oficina virtual, todo presentado en 120 proyectos únicos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-036-8



PHP 6

Este libro es un completo curso de programación en PHP en su versión 6.0. Un lenguaje que se destaca tanto por su versatilidad como por el respaldo de una amplia comunidad de desarrolladores, que lo convierten en un punto de partida ideal para quienes comienzan a programar.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-663-039-9

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



200 Respuestas: Blogs

Esta obra es una completa guía que responde a las preguntas más frecuentes de la gente sobre la forma de publicación más poderosa de la Web 2.0. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-663-037-5



Hardware paso a paso

En este libro encontraremos una increíble selección de actividades que abarcan todos los aspectos del hardware. Desde la actualización de la PC hasta el overclocking de sus componentes, todo en una presentación nunca antes vista, realizada íntegramente con procedimientos paso a paso.

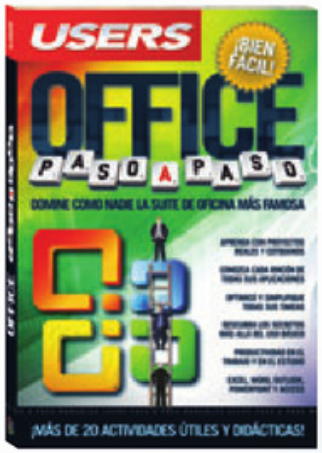
→ COLECCIÓN: PASO A PASO
→ 320 páginas / ISBN 978-987-663-034-4



200 Respuestas: Windows 7

Esta obra es una guía básica que responde, en forma visual y práctica, a todas las preguntas que necesitamos conocer para dominar la última versión del sistema operativo de Microsoft. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-663-035-1



Office paso a paso

Este libro presenta una increíble colección de proyectos basados en la suite de oficina más usada en el mundo. Todas las actividades son desarrolladas con procedimientos paso a paso de una manera didáctica y fácil de comprender.

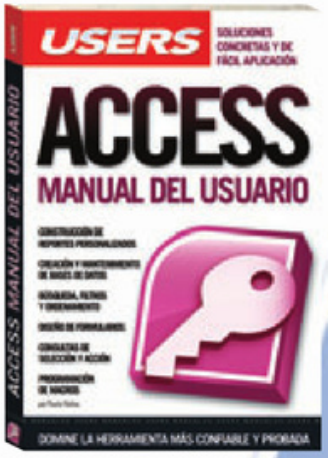
→ COLECCIÓN: PASO A PASO
→ 320 páginas / ISBN 978-987-663-030-6



101 Secretos de Hardware

Esta obra es la mejor guía visual y práctica sobre hardware del momento. En su interior encontraremos los consejos de los expertos sobre las nuevas tecnologías, las soluciones a los problemas más frecuentes, cómo hacer overclocking, modding, y muchos más trucos y secretos.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-029-0



Access

Este manual nos introduce de lleno en el mundo de Access para aprender a crear y administrar bases de datos de forma profesional. Todos los secretos de una de las principales aplicaciones de Office, explicados de forma didáctica y sencilla.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-025-2

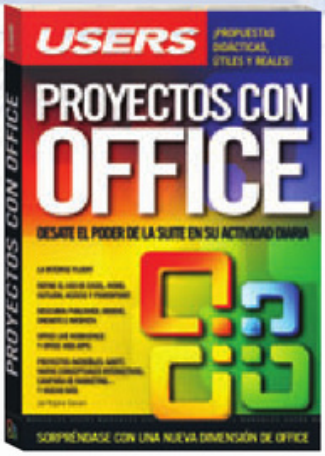
usershop.redusers.com



Redes Cisco

Este libro permitirá al lector adquirir todos los conocimientos necesarios para planificar, instalar y administrar redes de computadoras. Todas las tecnologías y servicios Cisco, desarrollados de manera visual y práctica en una obra única.

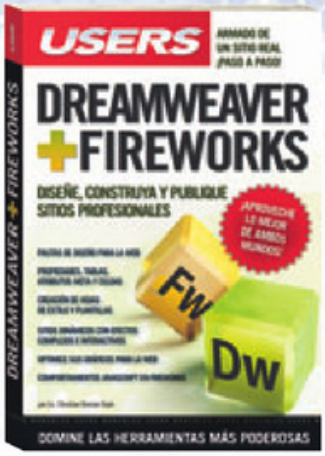
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-024-5



Proyectos con Office

Esta obra nos enseña a usar las principales herramientas de Office a través de proyectos didácticos y útiles. En cada capítulo encontraremos la mejor manera de llevar adelante todas las actividades del hogar, la escuela y el trabajo.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-023-8



Dreamweaver y Fireworks

Esta obra nos presenta las dos herramientas más poderosas para la creación de sitios web profesionales de la actualidad. A través de procedimientos paso a paso, nos muestra cómo armar un sitio real con Dreamweaver y Fireworks sin necesidad de conocimientos previos.

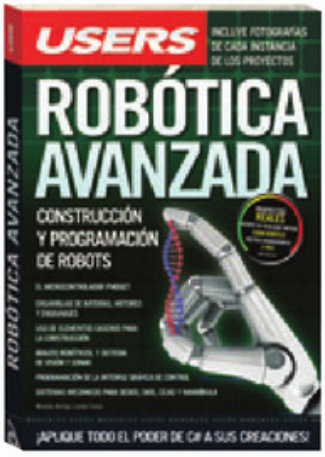
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-022-1



Excel revelado

Este manual contiene una selección de más de 150 consultas de usuarios de Excel y todas las respuestas de Claudio Sánchez, un reconocido experto en la famosa planilla de cálculo. Todos los problemas encuentran su solución en esta obra imperdible.

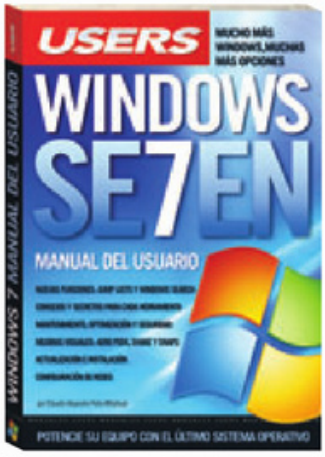
→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-021-4



Robótica avanzada

Esta obra nos permitirá ingresar al fascinante mundo de la robótica. Desde el ensamblaje de las partes hasta su puesta en marcha, todo el proceso está expuesto de forma didáctica y sencilla para así crear nuestros propios robots avanzados.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-020-7



Windows 7

En este libro, encontraremos las claves y los secretos destinados a optimizar el uso de nuestra PC tanto en el trabajo como en el hogar. Aprenderemos a llevar adelante una instalación exitosa y a utilizar todas las nuevas herramientas que incluye esta versión.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-015-3

¡Léalo antes Gratis!

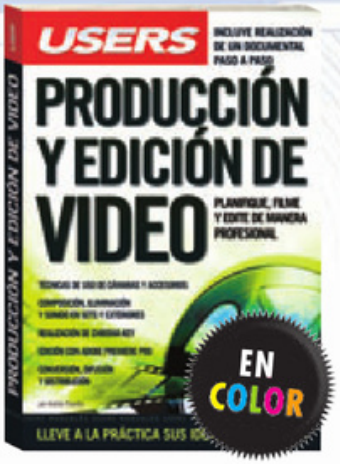
En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



De Windows a Linux

Esta obra nos introduce en el apasionante mundo del software libre a través de una completa guía de migración, que parte desde el sistema operativo más conocido: Windows. Aprenderemos cómo realizar gratuitamente aquellas tareas que antes hacíamos con software pago.

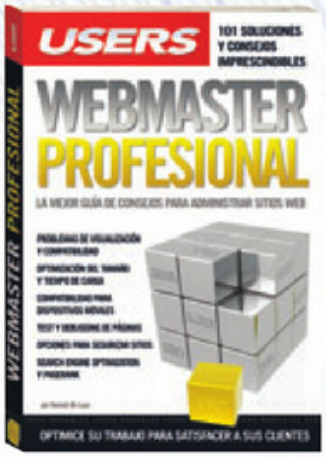
→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-013-9



Producción y edición de video

Un libro ideal para quienes deseen realizar producciones audiovisuales con bajo presupuesto. Tanto estudiantes como profesionales encontrarán cómo adquirir las habilidades necesarias para obtener una salida laboral con una creciente demanda en el mercado.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-012-2



Webmaster profesional

Esta obra explica cómo superar los problemas más frecuentes y complejos que enfrenta todo administrador de sitios web. Ideal para quienes necesiten conocer las tendencias actuales y las tecnologías en desarrollo que son materia obligada para dominar la Web 2.0.

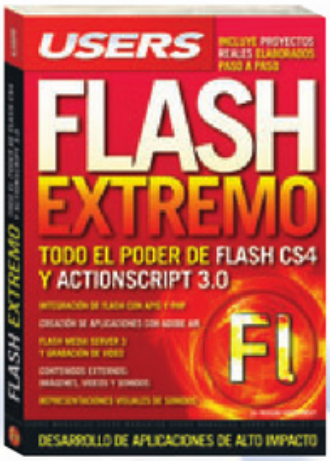
→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-011-5



Silverlight

Este manual nos introduce en un nuevo nivel en el desarrollo de aplicaciones interactivas a través de Silverlight, la opción multiplataforma de Microsoft. Quien consiga dominarlo creará aplicaciones visualmente impresionantes, acordes a los tiempos de la incipiente Web 3.0.

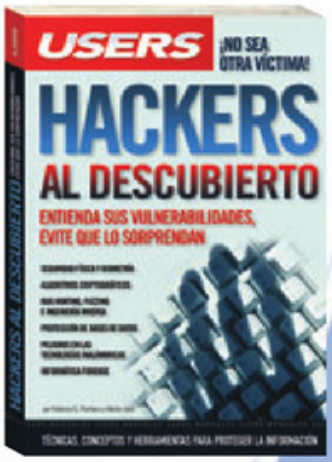
→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-010-8



Flash extremo

Este libro nos permitirá aprender a fondo Flash CS4 y ActionScript 3.0 para crear aplicaciones web y de escritorio. Una obra imperdible sobre uno de los recursos más empleados en la industria multimedia, que nos permitirá estar a la vanguardia del desarrollo.

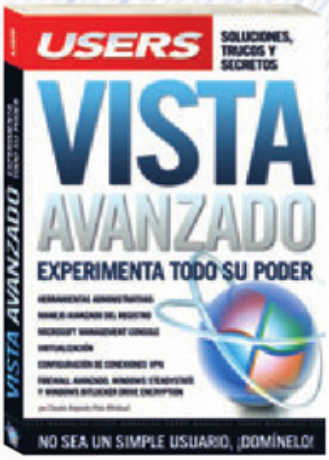
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-009-2



Hackers al descubierto

Esta obra presenta un panorama de las principales técnicas y herramientas utilizadas por los hackers, y de los conceptos necesarios para entender su manera de pensar, prevenir sus ataques y estar preparados ante las amenazas más frecuentes.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-008-5



Vista avanzado

Este manual es una pieza imprescindible para convertirnos en administradores expertos de este popular sistema operativo. En sus páginas haremos un recorrido por las herramientas fundamentales para tener máximo control sobre todo lo que sucede en nuestra PC.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-007-8



101 Secretos de Excel

Una obra absolutamente increíble, con los mejores 101 secretos para dominar el programa más importante de Office. En sus páginas encontraremos un material sin desperdicios que nos permitirá realizar las tareas más complejas de manera sencilla.

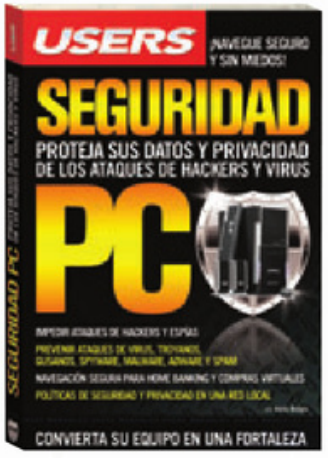
→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-005-4



Electrónica & microcontroladores PIC

Una obra ideal para quienes desean aprovechar al máximo las aplicaciones prácticas de los microcontroladores PIC y entender su funcionamiento. Un material con procedimientos paso a paso y guías visuales, para crear proyectos sin límites.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-663-002-3



Seguridad PC

Este libro contiene un material imprescindible para proteger nuestra información y privacidad. Aprenderemos cómo reconocer los síntomas de infección, las medidas de prevención por tomar, y finalmente, la manera de solucionar los problemas.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-004-7



Hardware desde cero

Este libro brinda las herramientas necesarias para entender de manera amena, simple y ordenada cómo funcionan el hardware y el software de la PC. Está destinado a usuarios que quieran independizarse de los especialistas necesarios para armar y actualizar un equipo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-001-6



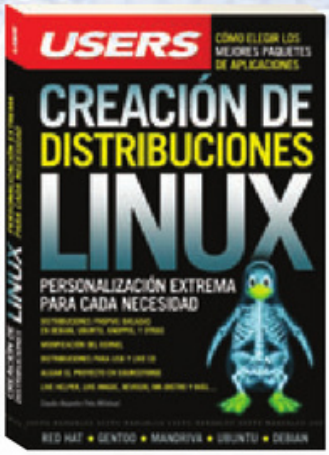
200 Respuestas: Photoshop

Esta obra es una guía que responde, en forma visual y práctica, a todas las preguntas que necesitamos contestar para conocer y dominar Photoshop CS3. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-98-8

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.



Creación de distribuciones Linux

En este libro recorreremos todas las alternativas para crear distribuciones personalizadas: desde las más sencillas y menos customizables, hasta las más avanzadas, que nos permitirán modificar el corazón mismo del sistema, el kernel.

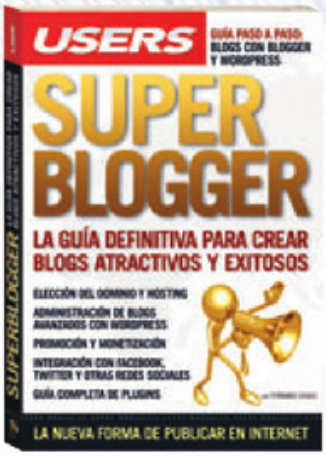
→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-99-5



Métodos ágiles

Este libro presenta una alternativa competitiva a las formas tradicionales de desarrollo y los últimos avances en cuanto a la producción de software. Ideal para quienes sientan que las técnicas actuales les resultan insuficientes para alcanzar metas de tiempo y calidad.

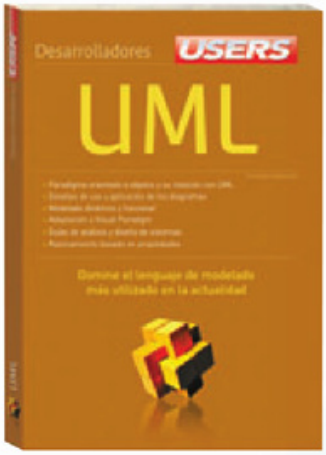
→ COLECCIÓN: DESARROLLADORES
→ 336 páginas / ISBN 978-987-1347-97-1



SuperBlogger

Esta obra es una guía para sumarse a la revolución de los contenidos digitales. En sus páginas, aprenderemos a crear un blog, y profundizaremos en su diseño, administración, promoción y en las diversas maneras de obtener dinero gracias a Internet.

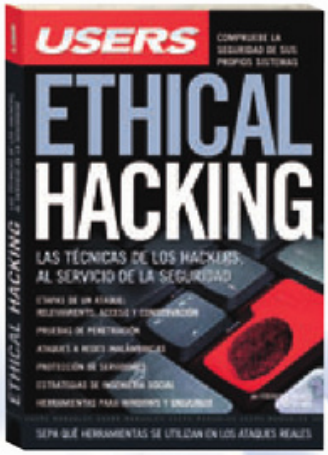
→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-96-4



UML

Este libro es la guía adecuada para iniciarse en el mundo del modelado. Conoceremos todos los constructores y elementos necesarios para comprender la construcción de modelos y razonarlos de manera que reflejen los comportamientos de los sistemas.

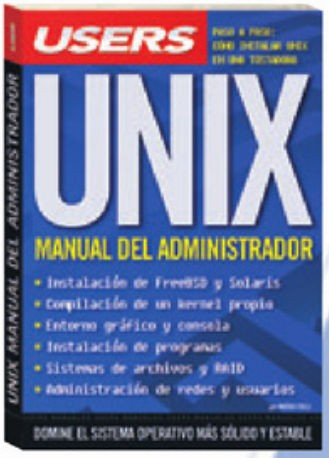
→ COLECCIÓN: DESARROLLADORES
→ 320 páginas / ISBN 978-987-1347-95-7



Ethical Hacking

Esta obra expone una visión global de las técnicas que los hackers maliciosos utilizan en la actualidad para conseguir sus objetivos. Es una guía fundamental para obtener sistemas seguros y dominar las herramientas que permiten lograrlo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-93-3



UNIX

Este manual contiene un material imperdible, que nos permitirá dominar el sistema operativo más sólido, estable, confiable y seguro de la actualidad. En sus páginas encontraremos las claves para convertirnos en expertos administradores de FreeBSD.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-94-0

USERS PRESENTA...

¡EL PRIMER EBOOK USERS!

Sí, ya podés leer Hackers al descubierto en tu PC, notebook, Amazon Kindle, iPad, en el celular...

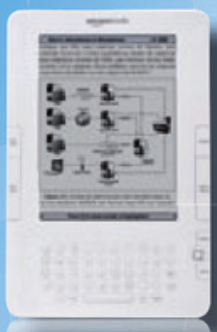
CONSEGUILO DESDE CUALQUIER PARTE DEL MUNDO

A UN PRECIO INCREÍBLE



¿QUÉ ESTÁS ESPERANDO?







¡LEELO DONDE QUIERAS!

INGRESA YA A USERSHOP.REDUSERS.COM Y ENTERATE MÁS

PROGRAME APLICACIONES Y APRENDA LOS MÚLTIPLES USOS DE LOS PICS



Una obra ideal para quienes desean aprovechar al máximo las aplicaciones prácticas de los microcontroladores PIC y entender su funcionamiento. Un material con procedimientos paso a paso y guías visuales, para crear proyectos sin límites.

» DESARROLLO / INTERNET
» 368 PÁGINAS
» ISBN 978-987-663-002-3



LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

*SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // **VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com // usershop@redusers.com

USERS



El contenido de este libro fue publicado en el curso en fascículos Electrónica Digital

CONTENIDO

1 | LÓGICA DIGITAL, MEMORIAS
Sistemas digitales / Sistema binario / Circuitos secuenciales

2 | MICROCONTROLADORES
Entrada y salida / Microcomputadoras / Interrupciones

3 | MICROCONTROLADORES PIC16F
Características / MPLAB / Grabador de PICs / Contador de 1 dígito / Simulador de vela

4 | PIC18F
Familia PIC18F / Placa experimental / Periféricos / Contador y PWM / Manejo de display LCD

5 | CONECTIVIDAD (NO INALÁMBRICA)
Protocolos de conectividad / Serial Peripheral Interface / Memoria SPI / Bus de comunicación I2C / Universal Serial Bus

6 | MÓDULOS RABBIT
Módulos Rabbit / Dynamic C / Mundo Rabbit

7 | DISPLAYS GRÁFICOS LCD
Arquitectura de un display LCD / Dibujar letras y gráficos

8 | NETWORKING CON TCP/IP
Networking / Servidores web / Ethernet con PIC

9 | NETWORKING CON RABBIT
Servidores web SSI y CGI / RabbitWeb / Wi-Fi con Rabbit

10 | NETWORKING CON PIC
Agregar servicios al stack / Envío de mails

NIVEL DE USUARIO

PRINCIPIANTE | INTERMEDIO | AVANZADO | EXPERTO

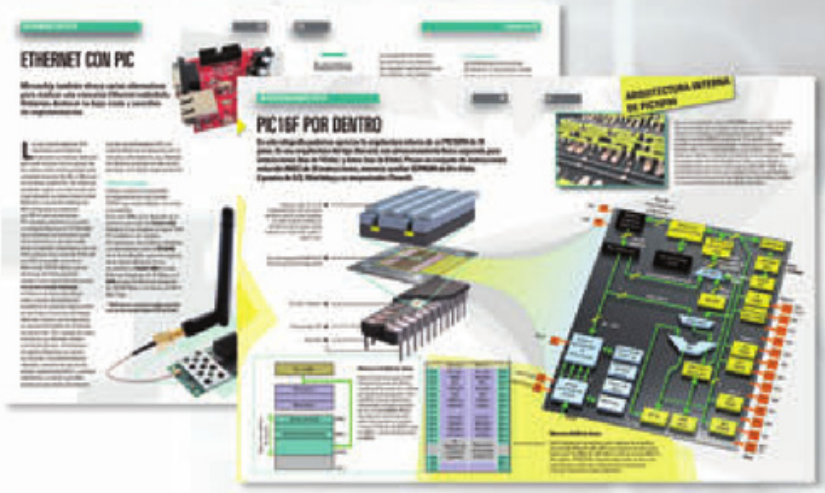
MICROCONTROLADORES

Este manual es ideal para todos aquellos que quieran iniciarse en el mundo de la electrónica digital y la programación de microcontroladores, así como también para los nuevos profesionales que busquen perfeccionarse en el área.

A través de esta obra, podrán conocer los fundamentos de los sistemas digitales, ahondar en el funcionamiento de los microcontroladores, aprender sobre las familias más populares: PIC16 F y PIC18F, hasta llegar a conectar los dispositivos de manera inalámbrica, entre muchos otros proyectos.

Todos los procedimientos son expuestos de forma visual y práctica, con fotografías a todo color, diagramas conceptuales, infografías 3D y la teoría necesaria para comprender en profundidad cada tema presentado.

En definitiva, un extenso y completo recorrido por el maravilloso mundo de los microcontroladores, presentado por los máximos expertos. Una obra única, que nos permitirá estar preparados para el demandante mercado laboral actual.



RedUSERS.com

En este sitio encontrará una gran variedad de recursos y software relacionado, que le servirán como complemento al contenido del libro. Además, tendrá la posibilidad de estar en contacto con los editores, y de participar del foro de lectores, en donde podrá intercambiar opiniones y experiencias.

Si desea más información sobre el libro puede comunicarse con nuestro Servicio de Atención al Lector: usershop@redusers.com

MICROCONTROLLERS

In this book we will learn the basics of digital logic and get involved in depth in the fascinating world of microcontrolers. The expert's tips and pieces of advice contained in this manual, will help us become professional electronic technicians.

ISBN 978-987-1773-56-5

MANUALES USERS MANUALES USERS MANUALES USERS MANUALES USERS MAN

ALCANCE UN NUEVO NIVEL EN EL MUNDO DE LA ELECTRÓNICA