// DESCARGA GRATUITA //

“int año = 1, número 3;” /* MAYO 2008 */

µCONTROL

Electrónica en General Pics en Particular

Matrices de LEDs

Todo lo que tienes que saber para la construcción de tu propio cartel de matriz de diodos LEDs

Detector de Humo y Gases

Fácil montaje basado en el sensor de gases NAP-11AS

Memorias I2C con Protón Lite

La aplicación en BASIC de este sencillo y popular medio de almacenamiento.

"int año = 1, numero 3;" /* MAYO 2008 */

μCONTROL

Electrónica en General Pics en Particular

Matrices de LEDs

Todo lo que tienes que saber
para la construcción de tu propio cartel
de matriz de diodos LEDs

Detector de Humo y Gases

Fácil montaje
basado en el sensor
de gases NAP-1 1AS

Memorias I2C con Protón Lite

La aplicación en BASIC
de este sencillo y popular
medio de almacenamiento.

.indice

Pic BASIC III	0x04
uso práctico del PIC12F675 II	0x09
sensor de temperatura LM35	0x0F
retardador de la red electrica cpn el C1555	0x10
matrices de LEDs	0x13
matriz de LEDs de 8x8	0x1B
memorias I2C con Proton Lite	0x20
módulo ICSP para PIC16F877 y Protoboard	0x25
cálculo de disipadores	0x27
decodificador de protocolo ABA Track2	0x29
usando LCDs II	0x2D
detector de humo y gases microcontrolado	0x34
el relojito III	0x38
paleotrónica: SID6581	0x3E

.staff

hicimos este número...

µCONTROL
Electrónica en General Pics en Particular

número = 3; año = 1;

Dirección y Redacción:
Ariel Palazzesi
Argentina
arielpalazzesi@gmail.com
www.ucontrol.com.ar

Edición, Redacción y Corrección:
Reinier Torres Labrada
Cuba
reiniertl@gmail.com

Diseño:
DCV Verónica C. Lavore
Argentina
azimut.estudio@gmail.com

Consejo Editorial:
Mario Sacco
Argentina
service.servisystem@gmail.com

Carlos Ortega Sabio
España
carlos.ortegasabio@ucontrol.revista.com.ar

Diego Márquez García - Cuervo

Marcos Lazcano
Argentina
marcos.lazcano@gmail.com

Pedro
Venezuela
palitroquez@gmail.com

.editorial

Es increíble lo rápido que pasan dos meses cuando uno tiene la responsabilidad de armar una revista que ve la luz cada 64 días. Hay que robar horas al descanso, e intentar no excederse de la fecha de entrega. Extrañamente, y a pesar de la gran cantidad de horas que insume esta tarea, en ningún momento la hemos sentido como una carga. Todo lo contrario.

Tienes en tus manos el tercer número de esta revista, que poco a poco, se va afianzando y que ya han descargado bastante más de 40.000 personas. Es una sensación muy interesante el sentir que hay, distribuidos por casi todo el mundo, hay miles de lectores esperando que este numero esté listo. Eso nos llena de orgullo, y a la vez, nos compromete aun más con este proyecto.

En cada número de uControl Revista se van sumando colaboradores. Cuando nos sentamos a definir como sería esta publicación, no estamos muy seguros de que los aficionados se animasen a enviarnos sus trabajos. Hoy podemos decir que nos equivocamos: prácticamente todos los días recibimos algún trabajo que merecería ser publicado. Y lo serán, en los ejemplares siguientes.

Éste numero esta dedicado a los carteles de LEDs. El hombre siempre ha sentido una extraña fascinación por las luces de colores (quizás sea una evolución natural de los “espejitos de colores” que deleitaron a los indígenas de América hace 500 años), y si vienen de a centenares, mejor que mejor. Y eso es de lo que se trata un cartel de LEDS: varios cientos de luces organizadas de forma que pueden exhibir un mensaje escrito o una imagen. Hoy puedes comenzar a diseñar el tuyo.

Los que gustan de montar proyectos, sin preocuparse por tener que dedicar horas a su diseño, están de parabienes. Este ejemplar continúa con la publicación de la sección “Circuiteca”, en la que encontrarás algunos circuitos que seguramente llamarán tu atención.

Hemos dedicado el artículo “retro” al emblemático chip de sonido SID6581, que durante años deleitó nuestros oídos generando todos los sonidos de la mítica Commodore 64. Hoy día se ha convertido en un objeto de culto muy buscado por los coleccionistas.

Aquellos que siguen de cerca los tutoriales de programación y la construcción del “relojito”, podrán seguir aprendiendo gracias a los artículos correspondientes.

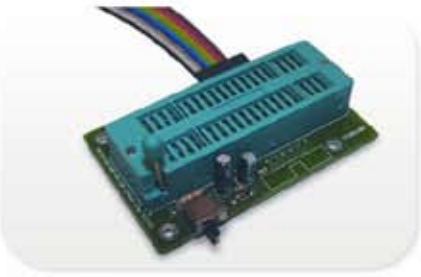
Como siempre, esperamos sus mails con las críticas y sugerencias, para que cada ejemplar de uControl Revista sea un poco mejor que el anterior. ■

GTP-USB+

Grabador de Microcontroladores y Memorias por puerto USB 2.0
Hardware oficial del software Winpic800.

- **SOLO CONECTAR Y USAR.** El puerto USB provee la alimentación y el HID facilita la instalación. Ideal para uso con notebook.
- **ALTA VELOCIDAD DE TRANSFERENCIA DE DATOS.** Con USB 2.0 Full Speed (12 Mb/s), el conjunto GTP-USB+ /Winpic800, puede ser utilizado en equipos con USB 1.1.
- **AMPLIO LISTADO DE DISPOSITIVOS SOPORTADOS.** Soporta numerosos microcontroladores de Microchip (PIC, dsPIC, rfPIC), Atmel (en modo serie, y paralelo con un adaptador) y EEPROM (24Xxx, I2C y 93Xxx, Microwire).
- **IDENTIFICACION AUTOMÁTICA.** El soft Winpic800 identifica automáticamente el dispositivo conectado.
- **ACTUALIZABLE.** El firmware del GTP-USB+ se actualiza con cada nueva versión del Winpic800.
- **GRABACIÓN DIRECTA O EN CIRCUITO.** ICSP, ISP, I2C, SPI.

Módulos ZIF disponibles para todas las familias



mstools

www.mstools.com.ar
ventas@mstools.com.ar
011-4764-2324 15-5158-7306

Electronic Development

//Ariel Palazzesi//
arielpalazzesi@gmail.com

PIC BASIC

capítulo III

Continuamos con nuestro cursillo de programación de microcontroladores en lenguaje PIC BASIC del PIC SIMULATOR IDE. En esta oportunidad veremos como emplear las instrucciones relacionadas con el control del flujo del programa.

Si un programa fuese simplemente una lista de órdenes a ser ejecutadas una detrás de otra, en forma lineal, habría muchos problemas que no tendrían solución. Es la posibilidad de tomar decisiones a lo largo de la ejecución del programa, y la ventaja de repetir grupos de instrucciones cuando es necesario, lo que hace de la programación algo realmente útil.

Todos los lenguajes de programación contienen instrucciones que permiten realizar estas acciones, y PIC BASIC no es una excepción. Hoy veremos cuales son, y como se utilizan.

IF - THEN - ELSE - ENDIF

En cualquier programa medianamente complejo que queramos realizar, seguramente necesitaremos en algún punto tomar alguna decisión basándonos en el estado de una entrada o en el valor de una variable. **PIC BASIC** incorpora instrucciones que nos permiten este tipo de comportamiento, siendo la mas sencilla y frecuentemente utilizada la sentencia **IF - THEN - ELSE - ENDIF.**

Existen varias formas de utilizar esta instrucción. Comenzaremos con los casos mas sencillos y a lo largo de este capitulo iremos agregando complejidad hasta ver todas las posibilidades.

CASO 1: El caso más simple es el siguiente:

IF condición THEN instrucción

"IF" significa "SI....", y "THEN" significa "LUEGO" o "ENTONCES". El caso anterior puede leerse como "SI se cumple la condición, entonces ejecuto la instrucción"

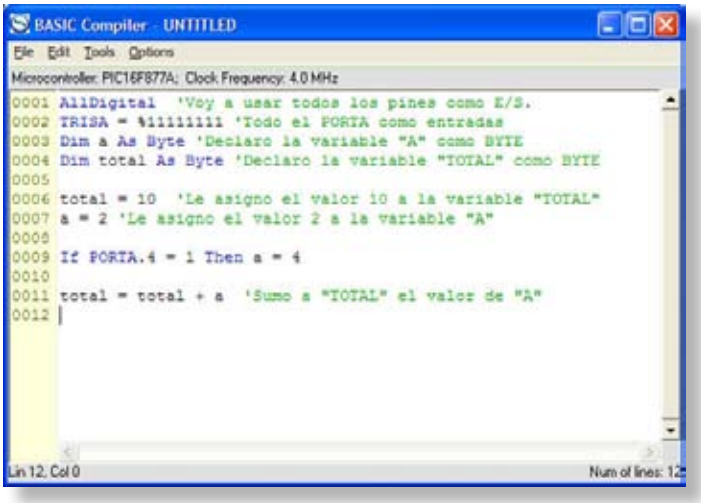
La "condición" es una expresión lógica que puede ser verdadera o falsa. En caso de ser verdadera, la instrucción a continuación del *THEN* será ejecutada. En caso de la condición sea falsa, el programa seguirá su ejecución con la instrucción siguiente al "IF - THEN".

Veamos un ejemplo. Supongamos el siguiente programa:

```
ALLDIGITAL 'Voy a usar todos los pines como E/S.
 TRISA = %11111111 'Todo el PORTA como entradas
 DIM A AS BYTE 'Declaro la variable "A" como BYTE
 DIM TOTAL AS BYTE 'Declaro la variable "TOTAL"
como BYTE
 '
 TOTAL = 10 'Le asigno el valor 10 a la variable "TOTAL"
 A = 2 'Le asigno el valor 2 a la variable "A"
 '
 IF PORTA.4 = 1 THEN A = 4
 '
 TOTAL = TOTAL + A 'Sumo a "TOTAL" el valor de "A"
```

Cuando comienza el programa, se declaran dos variables tipo BYTE (que pueden almacenar valores entre 0 y 255), y a TOTAL se le asigna el valor "0" y a "A" el valor "2". Hasta aquí, no hay nada que no hayamos visto antes.

La línea siguiente realiza la siguiente tarea: evalúa si la condición PORTA.4 = 1 es cierta. En caso de que efectivamente el valor presente en el bit 4 del PORTA sea "1", se ejecuta la instrucción a continuación del *THEN*, la variable "A" toma el valor "4", y se pasa a la instrucción de abajo. Si PORTA es igual a "0", se pasa a la instrucción siguiente sin más.



El valor final de la variable "TOTAL" depende entonces de cual sea el estado de PORTA.4 al momento de hacer la evaluación. Si es igual a "1", "TOTAL" tendrá un valor de 14 (10 + 4). Si PORTA.4 = 0, "TOTAL" tendrá un valor de 12 (10 + 2).

Veamos algunos ejemplos válidos de este caso:

```
IF A = B THEN PORTA.0 = 1

 IF B > A THEN A = B

 IF B = 5 THEN A = 0

 IF (A = 0) OR (B = 5) THEN C = 2

 IF PORTA.0 THEN PORTB.3 = 0
```

En el ultimo ejemplo la condición PORTA.0 equivale a PORTA.0 = 1.

CASO 2: Muchas veces, luego de evaluar la condición necesitamos ejecutar más de una instrucción. En los ejemplos vistos en el **CASO 1** siempre se ejecutaba una sola instrucción cuando la condición era cierta. La manera de ejecutar múltiples sentencias dentro de una estructura *IF-THEN* implica emplear el *ENDIF*:

```
IF condición THEN
   instrucción 1
   instrucción 2
   ...
   instrucción n
 ENDIF
```

No varia prácticamente nada respecto del primer caso, solo que esta vez se van a ejecutar todas las instrucciones que se encuentren entre el THEN y el ENDIF cada vez que condición sea verdadera.

Veamos un ejemplo. Supongamos el siguiente programa:

```
DIM A AS BYTE 'Declaro la variable "A" como BYTE
 DIM B AS BYTE 'Declaro la variable "B" como BYTE
 DIM C AS BYTE 'Declaro la variable "C" como BYTE
 DIM D AS BYTE 'Declaro la variable "D" como BYTE
 DIM TOTAL AS BYTE 'Declaro la variable "TOTAL"
                   como BYTE
 '
 TOTAL = 0 'Le asigno el valor 0 a la variable
           "TOTAL"
 A = 2 'Le asigno el valor 2 a la variable "A"
 B = 5 'Le asigno el valor 5 a la variable "B"
 C = 1 'Le asigno el valor 1 a la variable "C"
 D = 0 'Le asigno el valor 0 a la variable "D"
 '
 IF A = 2 THEN
   A = B + (C * D)
   TOTAL = A * B
 ENDIF
```

El ejemplo anterior, la condición A = 2 es verdadera (puesto que ese es el valor que le asignamos a "A" mas arriba), por lo que las dos instrucciones dentro del *THEN-ENDIF* se ejecutaran. Esto hace que TOTAL tome el valor de 10 (hagan las cuentitas!). Si "A" hubiese tenido otro valor, esas dos sentencias no se ejecutarían y TOTAL seguiría valiendo "0" al terminar el programa.

CASO 3: Hay veces que de acuerdo a la condición, queremos ejecutar un grupo u otro de instrucciones. Para eso, utilizamos el *ELSE:*

```
IF condición THEN
   instrucciónv 1
   instrucciónv 2
   ...
   instrucciónv n
 ELSE
   instrucciónf 1
   instrucciónf 2
   ...
   instrucciónf n
 ENDIF
```

Es decir, si la condición es verdadera, se ejecutan las sentencias entre THEN y ELSE. Y si la condición es falsa, las que estén entre *ELSE* y *ENDIF*. *"ELSE"* puede ser traducido como *"en otro caso"* o *"si no..."*.

Veamos un ejemplo. Supongamos el siguiente programa:

```
ALLDIGITAL 'Voy a usar todos los pines como E/S.
 '
 TRISA = %11111111 'Todo el PORTA como entradas
 DIM A AS BYTE 'Declaro la variable "A" como BYTE
 DIM TOTAL AS BYTE 'Declaro la variable "TOTAL"
                   como BYTE
 '
 TOTAL = 10 'Le asigno el valor 10 a la variable "TOTAL"
 A = 2 'Le asigno el valor 2 a la variable "A"
 '
 IF PORTA.4 = 1 THEN
   A = 4
   TOTAL = TOTAL + 5
 ELSE
   A = 0
   TOTAL = TOTAL + 15
 ENDIF
```

El ejemplo anterior, la condición PORTA.4 = 1 determina que bloque de instrucciones se ejecutan. Si es verdadera, A = 4 y TOTAL = TOTAL + 5 son usadas. Caso contrario se ejecutan A = 0 y TOTAL = TOTAL + 15. Luego, independientemente de cual haya sido el caso, el programa sigue con la sentencia que se encuentre a continuación del ENDIF.

Por ultimo, tenemos que saber que es posible "anidar" instrucciones *IF-THEN-ELSE-ENDIF*, con lo que se pueden tomar decisiones verdaderamente complejas. Por supuesto, tenemos que ser cautos en el uso de esta característica ya que debido a limitaciones en el tamaño de la pila y

cantidad de memoria disponible del PIC podemos ocasionar un desborde y el programa colapsara. Este seria un ejemplo de un anidamiento:

```
IF PORTB.1 = 1 THEN
     IF A = 2 THEN
       A = B + (C * D)
       TOTAL = A * B
   ELSE
       A = 0
   ENDIF
 ELSE
   A = 19
 ENDIF
```

Las sentencias en color negro corresponden a una estructura *IF-THEN-ELSE-ENDIF* y las que están en verde a la otra, que se encuentra dentro ("anidada" en) de la primera.

FOR - TO - STEP - NEXT

Así como la toma de decisiones que vimos antes está presente en casi todos nuestros programas, las estructuras que permiten repetir un grupo de instrucciones un número determinado de veces también son indispensables. En PIC SIMULATOR IDE hay dos de ellas. Veremos ya mismo la primera de ellas: *FOR - TO - STEP - NEXT.*

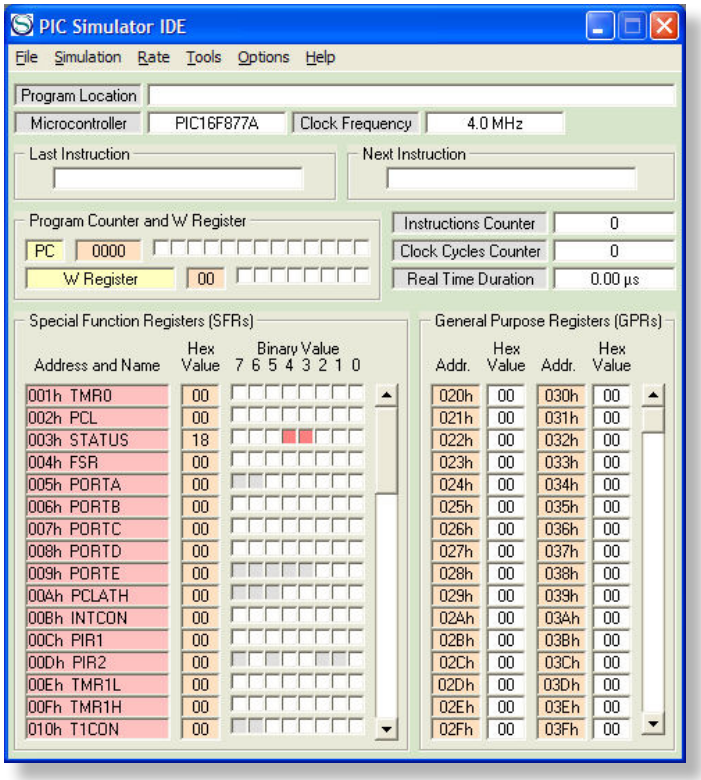
Esta estructura necesita una variable (tipo Byte o Word) para funcionar. En cada iteración del bucle, la variable va cambiando su valor. Cuando el valor de la variable alcanza o supera el valor prefijado, el bucle termina. La forma del bucle es la siguiente:

```
FOR variable = valor_inicial TO valor_final STEP paso
  instruccion1
  instruccion2
  ...
  instruccionn
 NEXT variable
```

Veamos un ejemplo concreto. Supongamos que queremos sumar los números del 1 al 100. El programa quedaría como sigue:

```
DIM A AS BYTE    'Declaro la variable "A" como BYTE
 DIM TOTAL AS WORD    'Declaro la variable "TOTAL"
                      como WORD
 '
 TOTAL = 0     'Asigno "0" a la variable "TOTAL".
 '
 FOR A = 1 TO 100 STEP 1 '"A" va de 1 a 100 de 1 en 1
  TOTAL = TOTAL + A    'Sumo "A" al valor de "TOTAL".
 NEXT A               'fin del bucle.
```

Hemos declarado la variable A como BYTE, ya que su valor va a mantenerse en el rango 0..255. Para TOTAL utilizamos una variable tipo WORD, ya que la suma va a superar el valor máximo de un BYTE. (Recordemos que WORD permite valores en el rango 0..65535)



El bucle se ejecuta 100 veces, la primera de ellas A vale 1, la segunda 2, la tercera 3, hasta la última en la que vale 100. Ese incremento (1 por ves) esta dado por el valor a continuación del STEP. En los casos como este en que *STEP* vale 1, puede omitirse, como veremos en ejemplos posteriores.

TOTAL comienza valiendo 0 (se le asigna ese valor fuera del bucle) y en cada iteración se le suma el valor que tenga A en ese momento. De esa manera, TOTAL va tomando los valores 1, 3, 6, 10, ... 5050.

Tanto *valor_inicial* como *valor_final* y paso pueden ser variables. El siguiente trozo de código hace lo mismo que el anterior, pero usa variables:

```
DIM A AS BYTE            'Declaro la variable "A"
                           como BYTE
 DIM INICIO AS BYTE      'Declaro la variable
                         "INICIO" como BYTE
 DIM FINAL AS BYTE       'Declaro la variable
                         "FINAL" como BYTE
 DIM PASO AS BYTE        'Declaro la variable
                         "PASO" como BYTE
 DIM TOTAL AS WORD       'Declaro la variable
                         "TOTAL" como WORD
 '
 INICIO = 1              'Asigno "1" a la variable
                         "INICIO".
 FINAL = 100             'Asigno "100" a la variable
                          "FINAL".
 PASO = 1                'Asigno "1" a la variable
                         "PASO".
 TOTAL = 0               'Asigno "0" a la variable
                         "TOTAL".
 '
 FOR A = INICIO TO FINAL STEP PASO '"A" va de 1 a
100 de 1 en 1
```

CONTÍNUA EN LA PÁGINA SIGUIENTE

```
TOTAL = TOTAL + A       'Sumo "A" al valor de
                         "TOTAL".
 NEXT A                  'fin del bucle.
```

Y el mismo ejemplo, sin usar STEP:

```
DIM A AS BYTE           'Declaro la variable "A"
                          como BYTE
 DIM TOTAL AS WORD      'Declaro la variable
                         "TOTAL" como WORD
 '
 TOTAL = 0               'Asigno "0" a la variable
                         "TOTAL".
 '
 FOR A = 1 TO 100        '"A" va de 1 a 100 de 1 en 1
  TOTAL = TOTAL + A      'Sumo "A" al valor de
                         "TOTAL".
 NEXT A                  'fin del bucle.
```

Hay casos en que es necesario que el valor de la variable de control del bucle se decremente en lugar de ir aumentando. En ese caso, se puede usar un valor negativo para STEP. El siguiente ejemplo cuenta desde 50 hasta 20, de 5 en 5:

```
DIM A AS BYTE          'Declaro la variable "A"
                  como BYTE
 '
 FOR A = 50 TO 20 STEP -5  '"A" va de 50 a 20
                            de 5 en 5
  instruccion1
  instruccion2
  ...
  Instrucción n
 NEXT A                    'fin del bucle.
```

De la misma manera que ocurría con I*F-THEN-ELSE-ENDIF*, pueden anidarse diferentes bucles *FOR-TO-STEP-NEXT* , uno dentro de otro:

```
FOR variable1 = valor_inicial1 TO valor_final1
                 STEP paso1
  FOR variable2 = valor_inicial2 TO valor_final2
                   STEP paso2
     instruccion1
     instruccion2
     ...
     Instrucción n
  NEXT variable2
 NEXT variable1
```

La única condición es que un bucle este completamente dentro del otro. El siguiente anidamiento daría un error en el compilador:

```
FOR variable1 = valor_inicial1 TO valor_final1
           STEP paso1
  FOR variable2 = valor_inicial2 TO valor_final2
                 STEP paso2
     instruccion1
     instruccion2
     ...
     instruccionn
  NEXT variable1
 NEXT variable2
```

Para terminar, veamos el siguiente código:

```
AllDigital
 TRISB = 0

 Dim a As Byte

 For a = 0 To 15
  PORTB = a
 Next a
```

Si se lo corre en el PIC SIMULATOR IDE, puede verse como los primeros 4 bits del PORTB cuentan en binario de 0 a 15.

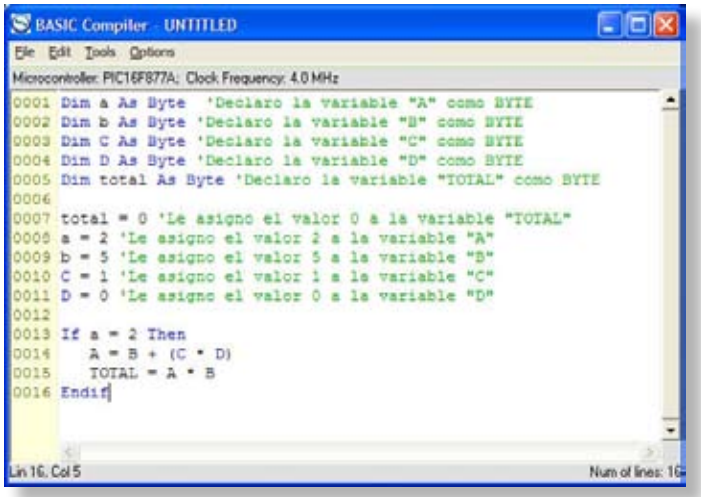
WHILE - WEND

La segunda estructura de control que proporciona PIC BASIC es *WHILE - WEND*. Su propósito es el mismo que la que vimos en el capitulo anterior, y su estructura es la siguiente:

```
WHILE condición
  instruccion1
  instruccion2
  ...
  Instrucción n
 WEND
```

Mientras que la condición sea verdadera, el grupo de instrucciones dentro del cuerpo del *WHILE-WEND* se ejecuta. Las características de la condición son las mismas que vimos antes para *IF-THEN-ELSE-ENDIF.*

Por supuesto, si no somos cuidadosos al momento de elegir la condición, puede darse el caso de que el numero de repeticiones del bucle sea infinito, y nunca salgamos de el. De hecho, esta circunstancia se aprovecha en algunos programas para repetir indefinidamente un grupo de instrucciones. También hay que tener presente que si la condición



no es cierta al momento de ejecutar la primera vez el *WHILE*, el flujo del programa pasara directamente a la instrucción posterior al *WEND* y las instrucciones dentro del bucle no se ejecutaran ninguna vez.

No hay mucho mas para decir de *WHILE-WEND* , solo analizar algunos ejemplos:

Ejemplo 1: El siguiente es un bucle infinito. Como dentro del cuerpo del *WHILE-WEND* no se cambia el valor de la variable A, esta siempre vale "0" y la condición del *WHILE* nunca es falsa, por lo que se repite eternamente:

```
DIM A AS BYTE
 A = 0
 ...
 WHILE A = 0
   instruccion1
   instruccion2
   ...
   Instrucción n
 WEND
 ...
```

Ejemplo 2: Las instrucciones dentro del siguiente *WHILE-WEND* no se ejecutan nunca, dado que la condicion siempre es falsa:

```
DIM A AS BYTE
 A = 0
 ...
 WHILE A > 0
   instruccion1
   instruccion2
   ...
   Instrucción n
 WEND
 ...
```

jemplo 3: Las instrucciones dentro del siguiente *WHILE-WEND* se ejecutan 10 veces, y al terminar la variable B contiene la suma de los números del 0 al 10 naturales:

```
DIM A AS BYTE
 DIM A AS BYTE
 A = 0
 B = 0
 '
 WHILE A < 10
    A = A + 1 'Incremento la variable A
    B = B + A 'Sumo a B el valor de la variable A
 WEND
```

Cuando A = 10, se suma su valor a A, y al llegar al WEND el control del programa se transfiere al *WHILE*, donde se evalúa la condición A < 10, se determina que es falsa, y el programa pasa el control a la línea que exista después del *WEND*.

Conclusión

Hemos visto como hacer para que nuestros programas sean capaces de tomar decisiones, y como lograr que un grupo de instrucciones se repita un numero determinado de veces. Estas dos características de PIC BASIC nos permitirán crear programas mucho más eficientes y compactos.

//Carlos Ortega Sabio//
carlos.ortegasabio@ucontrol.revista.com.ar

uso práctico del PIC12F675

parte II

En este nuevo número podremos construir y practicar con una mini-entrenadora de muy bajo coste basada en el PIC12F675 o similares. Será una excelente herramienta para comenzar a experimentar con estas pequeñas maravillas de la técnica moderna.

.Descripción del circuito

Con el capitulo anterior pudimos ver una aplicación practica del PIC12F675. El circuito que hoy les presentamos pondrá a nuestra disposición una mini-tarjeta entrenadora de reducidas dimensiones y de muy bajo coste, aunque sin resignar interés o posibilidades. Con ella podemos realizar prácticas con PWM, I/O, ADC y RS-232.

La placa esta compuesta de varios puertos de entrada salida. El CN2 nos permite descargar nuestros programas al micro directamente sin necesidad de extraerlo de su zócalo (ICSP, In Circuit Serial Programming). Esto nos aportara una gran comodidad y un gran ahorro de tiempo. Por otro lado podremos conectar un servomotor Futaba S3003 (o compatible) en CN1 para aprender a controlarlo por modulación de ancho de pulso (PWM, por pulse-width modulation en inglés). También podremos colocar un sensor de temperatura LM35 y varios dispositivos más. El puerto serie RS232 trabajará en modo Tx, de esta forma se enviara información serial al PC lo que nos facilitara la tarea de la depuración de los programas que estemos ensayando y nos permitirá enviar datos para poder ser procesados por nuestro ordenador para, por ejemplo, mostrar una grafica de temperatura. La entrenadora tiene incorporado un diodo led que junto con la tecla miniatura alojada en la placa nos permitirá hacer practicas con el modulo de entradas y salidas digitales (I/O) del PIC y por supuesto una resistencia ajustable que nos permite interactuar con el convertidor analógico digital que es un modulo interno conocido como ADC (Analog to Digital Converter).

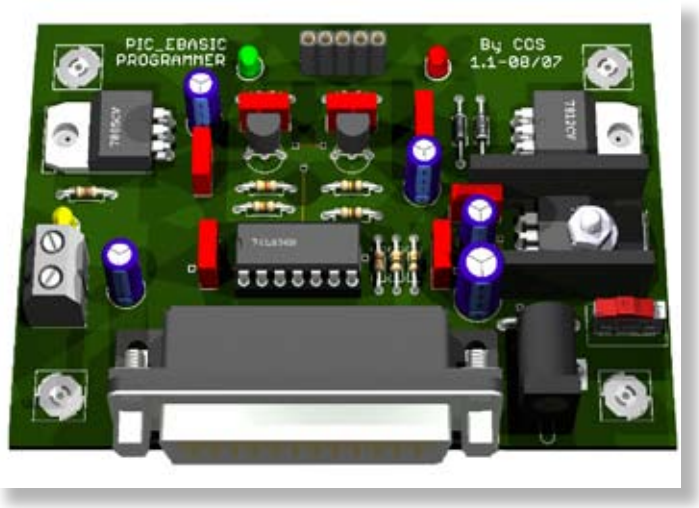


Imagen 3D de un programador.

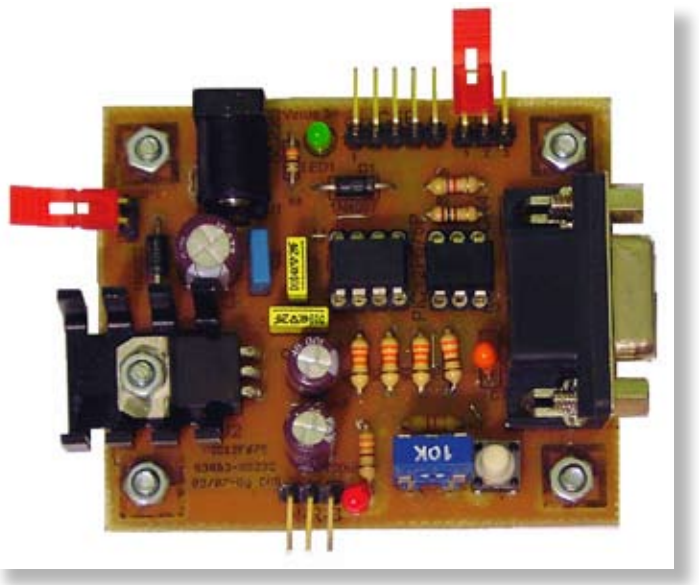


Imagen de la Mini-Entrenadora montada empleando el método de fabricación de circuitos impresos explicado en la revista.

.Memoria:

Antes de seguir haremos una pequeña reseña de las características de la memoria de programa (FLASH) del PIC. Este PIC en particular tiene una capacidad de

memoria FLASH de 1024 posiciones, cada una con una longitud de 12 bits. Cada posición de memoria contendrá una instrucción completa en código nemotécnico, y esta instrucción necesitará cuatro ciclos de reloj para poder ser ejecutada. La única excepción a esta regla es la instrucción de salto, que necesita el doble.

Pasando esto a números obtenemos lo siguiente: OSC/4, siendo OSC la velocidad del oscilador principal en este caso el interno trabajando a 4Mhz, 4.000.000Mhz/4=1.000.000Mhz y pasando a tiempo la frecuencia obtenida nos queda 1/1.000.000MhZ=0.000001Seg. = 1uSeg. Y esto significa que nuestro microcontrolador ejecutara 1 instrucción maquina por cada uSeg.

Para terminar con la memoria indicaremos que ***el Pic12F675 es un microprocesador de 8bit ya que generalmente se clasifican según la longitud de dato que maneja su juego de instrucciones maquina.***

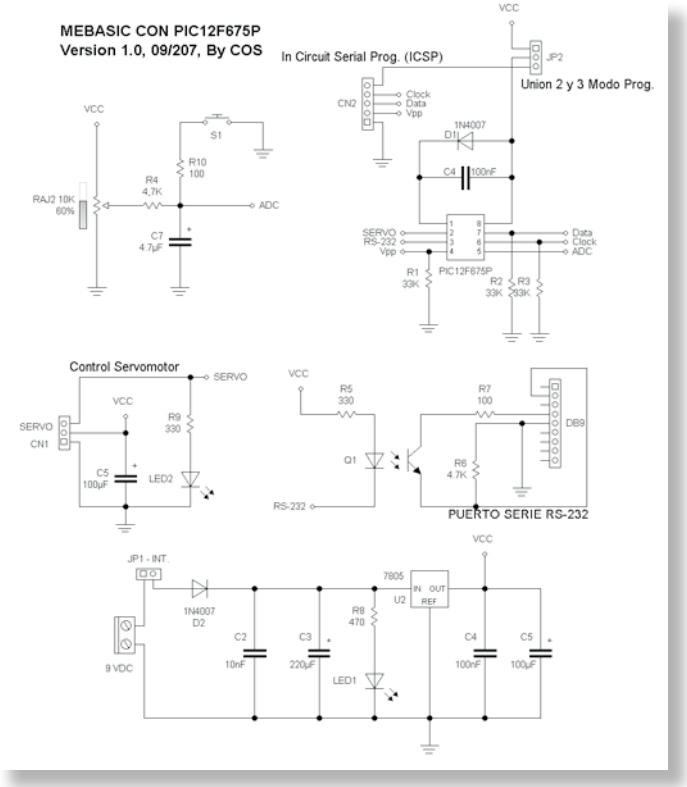
.Lenguajes de programación:

Para que nuestro hardware basado en el PIC12F675 pueda llevar a cabo alguna función predefinida por nosotros tendremos que utilizar un lenguaje de programación, mediante el que confeccionaremos una lista de instrucciones a ejecutar por el procesador interno de nuestro PIC y que posteriormente volcaremos en su memoria flash quedando residente en ella incluso después de desconectar la alimentación a nuestro circuito, en este caso nuestra placa entrenadora. Generalizando, podemos dividir los lenguajes de programación en dos grupos:

Lenguajes de bajo nivel o código maquina, llamado "Assembler" (assembly language), que es el lenguaje natural del microcontrolador. Este es el lenguaje mas rápido y los programas ocupan menos memoria, pero la opinión general es que es el mas difícil de aprender. Con tiempo se pueden preparar bloques de rutinas especializadas para insertar en los programas y facilitar el trabajo, pero a partir de cierta longitud de programa no se suele utilizar o se mezcla con otro lenguaje de alto nivel, ya que el tiempo de programación y depuración a ciertas longitudes de programa lo hacen solo factible para verdaderos expertos en él. Como opinión personal recomiendo que todo aquel que esté interesado en el desarrollo de hardware basado en microcontroladores o miprocesadores lo estudie sino a nivel experto si por lo menos a nivel básico, ya que implica comprender el modo de funcionamiento del microcontrolador y sus módulos internos de una forma muy eficaz, que luego se podrá reflejar en nuestros diseños.

Lenguaje de alto nivel. A diferencia del assembler se aleja del lenguaje nativo del procesador y se acerca mas al nuestro, así que cuanto mas se parece en su sintaxis al nuestro de mas alto nivel es. En general, el lenguaje aceptado por los programadores para los microcontroladores es el "C", un lenguaje cuyo compilador nos genera un código rápido y compacto, aunque su estructura puede crearnos alguna dificultad al principio. Es aceptado por la mayoría de los programadores para uso profesional aunque hay otros lenguajes no menos importantes. Y por ultimo llegamos al siempre polémico BASIC. Aunque la mayoría de los modernos lenguajes BASIC (hay muchos dialectos) no tienen nada que ver con sus antiguas versiones de apenas una década o menos, ha quedado clasificado como lenguaje de segundo orden. La característica principal del lenguaje BASIC es que tiene una sintaxis muy similar a la nuestra, por lo que es rápido de aprender y de depurar su código. Estas ventajas a menudo se pagan programas algo más lentos y largos que sus equivalentes en C o assembler.

En este caso usare el BASIC del PIC SIMULATOR IDE (PSI), que nombraré como BASIC PSI. El PSI es un entorno de trabajo que nos permite crear y editar programas tanto en BASIC como Assembler. Además contiene una serie de herramientas y componentes que nos permiten simular la mayoría de los programas generados con sus dos lenguajes. Una característica que lo hace interesante es que genera un código bastante compacto, lo que permite trabajar con cierta libertad con micros de menos de 1024 Word de memoria de programa. El compilador BASIC del PSI no tiene bug por lo que no tenemos que preocuparnos de que se produzcan fallos en nuestras rutinas. Y por ultimo, la mas importante cualidad (que en este caso hace al carácter didáctico de la revista), y por experiencia después de llevar 3 años publicando los programas que controlan mis proyectos con él, tiene un ca-



Esquema de la Mini-Entrenadora.

rácter universal que lo hace comprensible por aficionados y profesionales no importando cual sea el lenguaje usado habitualmente.

.Programadores:

Para poder continuar necesitamos unos conocimientos básicos sobre programadores de PIC ya que necesitaremos de hacernos con uno de ellos para poder volcar nuestros programas desde el PC a nuestra placa entrenadora. El programador o también conocido como "quemador" esta compuesto generalmente de dos partes de un hardware que contiene la circuiteria necesaria para poder conectarse a nuestro PIC y poder transferir nuestro programa a el, el "hard" del programador se complementa con un software que se ejecuta en nuestro ordenador, este software nos transfiere el archivo generado por el compilador de nuestro lenguaje respetando un protocolo determinado, este archivo pasa por el "hard" del programador que lo convierte a señales comprensibles por nuestro PIC.

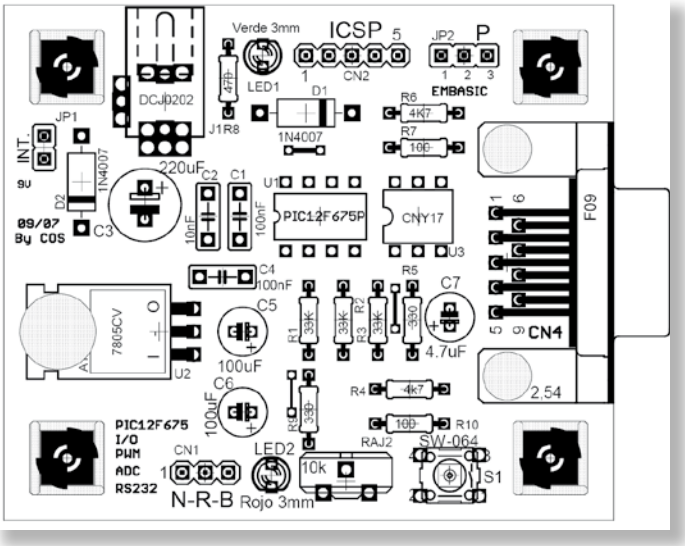
Así que para transferir nuestro programa tenemos que conectar el programador físicamente al PIC, esto se puede hacer de varias formas ya sea que el programador tenga un zócalo en su circuito impreso para poder insertar el PIC hasta ser programado y vuelto a colocar en nuestro circuito o por ejemplo como la entrenadora que tratamos en este capitulo, en el que conectamos el programador mediante un cable de cinta plana con unos conectores a ella, y de esta manera no tenemos que extraer el micro de nuestra placa. El "soft" de nuestro programador tiene que ser configurado como mínimo para indicarle que modelo de PIC estamos utilizando. Hay muchos programadores completos de uso libre que circulan por la red, por lo que no suele ser un grave problema de fabricar o comprar alguno.

Por comodidad utilizo una versión adaptada por mí de uno de los varios que se pueden obtener en la página del PSI (http://www.oshonsoft.com/picprog.html). Desde el software del programador puedo activar o desactivar la alimentación del micro de la entrenadora así como enviarle un Reset. Por supuesto que hay que tener en cuenta que hay muchos tipos de programadores ya estén integrados en la misma placa de nuestro proyecto o ya sean programadores de un nivel mas profesional que suelen llevar un gran numero de funciones ya que están gobernados por un microcontrolador ellos mismos, de todas formas a saber que ya usemos un tipo u otro ambos nos programaran nuestro PIC.

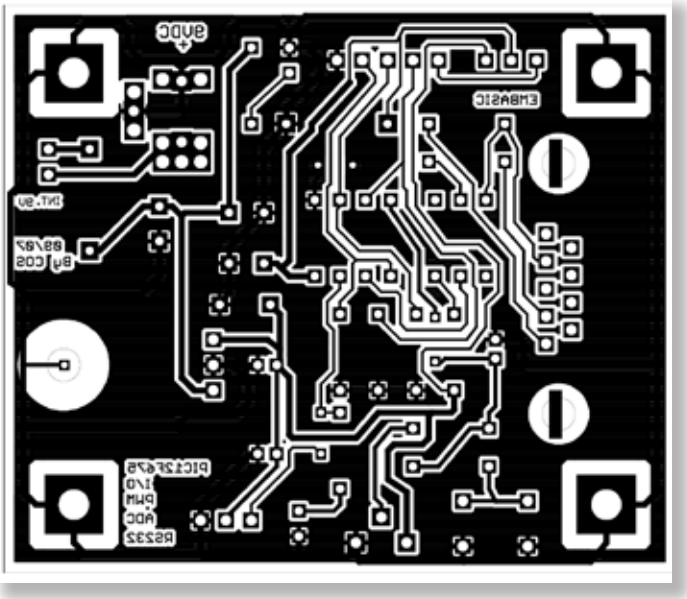
Nuestra placa entrenadora se conecta al "hard" de nuestro programador mediante 5 hilos dicha conexión se realiza mediante CN2, que corresponde: Vpp (5Vdc) esta tensión de alimentación la controla el programador y para esto cambiaremos de posición JP2 (uniones 2 y 3) en la placa, tenemos Vss que corresponde a GND del circuito, Data por donde se transfieren los datos al PIC, Clock que sincroniza la información que fluye entre programador y PIC, por ultimo y no menos importante la señal de un relativo alto voltaje (en este caso superior a 13V, en otros programadores puede ser inferior, a partir de 11V) que le indica a nuestro micro que entre en modo programación y siendo controlándola por nosotros desde el PC provoca un bloqueo del PIC o Reset según se utilice.

.Descripción general del circuito:

Comenzaremos con la descripción del circuito desde la fuente de alimentación que está compuesta por un jack de alimentación para circuito impreso J1, por donde entra la alimentación de 9Vdc a nuestro circuito. Continúa pasando (en serie) por el puente JP1 y D1, donde JP1 hace de interruptor y D1 nos protege de una posible inversión de polaridad. Los condensadores C2 y C3 ayudan al filtrado de la alimentación.R8 es la resistencia limitadora del LED1 de color verde 3mm, que cumple la función de testigo de la alimentación.



Posición de los componentes y puentes sobre el PCB.



Este es el PCB que albergara los componentes.

Luego os al popular estabilizador de 1Amp. LM7805CV (U2),que nos reduce y estabiliza la tensión de entrada a 5VDC,estando su salida filtrada por C4 y C5. Además, C5 (al igual que C3) ayuda a cumplir con la demanda de corriente instantánea de nuestra placa.

El resto de los componentes van asociados directamente al microcontrolador PIC12F675, siendo D1 la protección contra inversión de polaridad proveniente del conector CN2 que es el encargado de dar conexión a nuestro micro con el programador. JP2 nos permite seleccionar si la alimentación de nuestro circuito será suministrado por el programador o por la fuente interna de nuestra placa. Los resistores R3 y R2 polarizan las líneas de Datos y Clock del PIC, que podrían eliminarse del esquema siempre y cuando programásemos como salidas sus respectivos pines del micro en nuestros programas.

R1 no puede ser eliminada del circuito porque el pin del PIC asociado a Vpp no puede ser programado como salida digital, aplicando la norma de no dejar sin conexión (o “al aire”) ninguna entrada CMOS.

C1 es el condensador de desacople de la alimentación del PIC. Para trabajar con el ADC usaremos el RAJ2 con el que podremos variar la tensión en el pin asociado. Mediante R4 y C7 constituimos un circuito atenuador de las pequeñas variaciones de resistencia inherentes a la película de carbón de RAJ2. Hay que tener en cuenta que otra función importante de R4 es la de proteger al micro en el caso que olvidemos de configurar este pin (GP2) como entrada, ya que si lo configuramos como salida esta tendrá que estar en estado alto o bajo y hay que tener en cuenta que el cursor de RAJ2 puede llegar también a estar en uno de estos dos estados, y si se diera la coincidencia de que el cursor quedara en un estado contrario habría una “lucha de niveles” entre RAJ2 y el PIC llegando seguramente al deterioro de uno de los dos componentes. Esto solo pasaría si no estuviera R4 para impedirlo.

S1 es una tecla miniatura para soldar directamente en circuito impreso que comparte pin con R4 y C7 mediante R10, en este caso la función de R10 es similar a la de R4, tanto para atenuar la diferencia de niveles entre GP2 (en caso de ser programada como salida en estado alto y al mismo tiempo S1 estando pulsada). Además,

Lista de materiales:

Part	Value	Device
C1	100nF	CONDENSADOR CERAMICO, MKP, MKT
C2	10nF	CONDENSADOR CERAMICO, MKP, MKT
C3	220uF/16V	ELECTROLITICO
C4	100nF	CONDENSADOR CERAMICO, MKP, MKT
C5	100uF/16V	ELECTROLITICO
C6	100uF/16V	ELECTROLITICO
C7	4.7uF/16V	ELECTROLITICO
CN1		HEADER, MACHO ACODADO 3 ELEMENTOS
CN2		HEADER, MACHO ACODADO 5 ELEMENTOS
CN4		DB9, HEMBRA ACODADO PARA CIRCUITO IMPRESO
D1	1N4007	DIODO
D2	1N4007	DIODO
J1		JACK DE ALIMENTACION PARA CIRCUITO IMPRESO
JP1		HEADER, MACHO ACODADO 2 ELEMENTOS
JP2		HEADER, MACHO ACODADO 3 ELEMENTOS
LED1		LED, Verde 3mm
LED2		LED, Rojo 3mm
R1	33K	RESISENCIA 1/8W o 1/4W
R2	33K	RESISENCIA 1/8W o 1/4W
R3	33K	RESISENCIA 1/8W o 1/4W
R4	4k7	RESISENCIA 1/8W o 1/4W
R5	330	RESISENCIA 1/8W o 1/4W
R6	4K7	RESISENCIA 1/8W o 1/4W
R7	100	RESISENCIA 1/8W o 1/4W
R8	470	RESISENCIA 1/8W o 1/4W
R9	330	RESISENCIA 1/8W o 1/4W
R10	100	RESISENCIA 1/8W o 1/4W

Además necesitaremos:

- 1 placa de circuito impreso simple cara de 5.5x6.5mm

- 2 puentes hembra, como los utilizados en los discos duros (config. Master/esclavo).

- 1 clavija jack aerea de alimentacion complementaria a j1.

- 4 separadores m3 con sus correspondientes tuercas.

- 1 broca 0.6mm para los puentes.

- 1 broca 0.7mm componentes.

- 1 broca 1mm header y diodos.

- 1 broca 3.5mm para los agujeros de los separadores

otra función añadida es la de atenuar la descarga de C7 a través de S1. Otra utilidad de RAJ2 es la de polarizar a estado alto al micropulsador S1, llevando su cursor a positivo para que de esta forma al ser pulsado S1 pueda variar el estado de su pin asociado y que programaremos como entrada.

Pasando a otra parte del circuito tenemos R9, que es la resistencia limitadora del LED2 de color rojo 3mm al que podremos encender o apagar a voluntad en nuestros experimentos, el pin que lo controla esta compartido con CN1 donde podremos conectar directamente un servomotor Futaba 3003 entre otros dispositivos. C6 nos permitirá atender las demandas instantáneas de corriente del servomotor.

Pasando finalmente a la descripción de nuestro adaptador optoacoplado de señal TTL a niveles de RS232 funcionando solo como TX, tenemos a R5 que es la resistencia limitadora del LED interno de U3. Mediante GP5 se controla al transistor(también interno en U3) que se encarga de acoplar nuestro PIC al puerto serie RS-232 de nuestro ordenador. R7 suministra el estado alto RS-232 desde el mismo puerto serie (DTR) del ordenador y R6 suministra el estado bajo RS-232 desde GND_RS del ordenador, quedando el puerto total mente aislado de nuestro circuito.

.Montaje de la placa:

En general no soy partidario de seguir ninguna regla en particular, ya que esto varía según el usuario, los materiales y las herramientas disponibles. Pero recomiendo comenzar una vez terminado el taladrado de la placa colocando los puentes que sustituyen a las pistas de la cara superior. Posteriormente se puede revisar la integridad de todas las pistas del circuito con un polímetro configurado en modo conductividad.

Más tarde, podemos comenzar con la colocación y soldado de los componentes. En primer lugar todos los pequeños, como son resistores, diodos, LED, etc, continuando por los condensadores de pequeño tamaño, transistores, jumper, zócalos; y terminado con los componentes de gran tamaño, como pueden ser condensadores electroliticos, conectores, etc.

Colocar unos buenos separadores, lo suficientemente largos, a ambos lados de la placa facilita mucho el montaje de nuestro circuito. Como paso final antes de colocar los circuitos integrados (exceptuando el de alimentación U2), conectaremos alimentación a nuestro proyecto y verificaremos que las tensión 5V llega correctamente a los correspondientes pin de alimentación de los zócalos y conectores, sin olvidar comprobar que la tensión en el pin GP2 varía según movemos el cursor de RAJ2. Una vez terminada esta prueba dejaremos ajustada la resistencia para que se pueda leer el valor mas próximo a la tensión de alineación (+5VDC). Acto seguido pulsando S1 comprobaremos que dicho pin cambia de estado lógico.

Links:

Página del PIC Simulator IDE:
http://www.oshonsoft.com/picprog.html

//A. Palazzesi/C. Ortega Sabio//
arielpalazzesi@gmail.com
carlos.ortegasabio@ucontrol.revista.com.ar

sensor de temperatura LM35

El LM35 es un sensor de temperatura con una precisión calibrada de 1ºC. Puede medir temperaturas en el rango que abarca desde -55º a + 150ºC. La salida es muy lineal y cada grado centígrado equivale a 10 mV en la salida. Veremos sus características y forma de utilizarlo en nuestros proyectos.

A pesar de la existencia de otros sensores de temperatura que funcionan de forma analógica o incluso los del tipo DS1820 con interfaz 1-wire, el LM35 es uno de los más utilizados en los proyectos de los aficionados. Gran parte de su éxito se debe a la precisión que posee, y a su bajo costo. Este sensor es fabricado por Fairchild y National Semiconductor

Características:

- Precisión de ~1,5ºC (peor caso), 0.5ºC garantizados a 25ºC.
- No linealidad de ~0,5ºC (peor caso).
- Baja corriente de alimentación (60uA).
- Amplio rango de funcionamiento (desde -55º a + 150ºC).
- Bajo costo.
- Baja impedancia de salida.

Su tensión de salida es proporcional ala temperatura, en la escala Celsius. No necesita calibración externa y es de bajo costo. Funciona en el rango de alimentación comprendido entre 4 y 30 voltios.

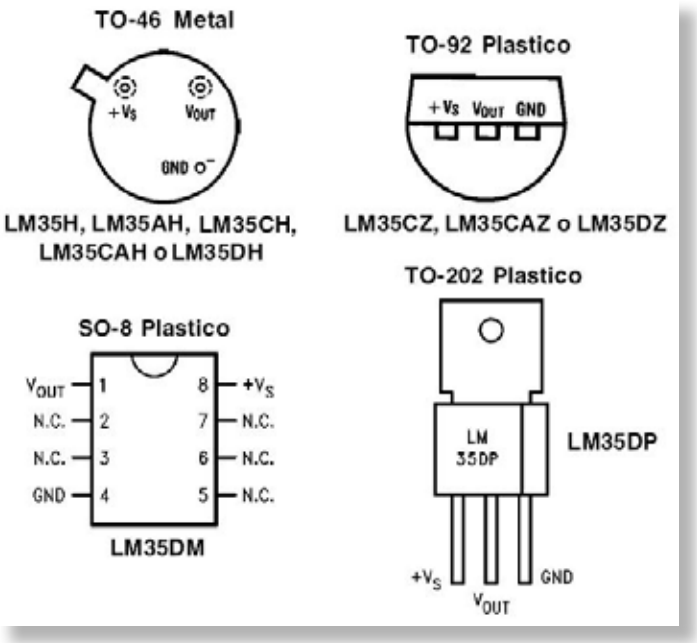
Como ventaja adicional, el LM35 no requiere de circuitos adicionales para su calibración externa cuando se desea obtener una precisión del orden de ±0.25 ºC a temperatura ambiente, y ±0.75 ºC en un rango de temperatura desde 55 a 150 ºC.

La baja impedancia de salida, su salida lineal y su precisa calibración inherente hace posible una fácil instalación en un circuito de control.

Debido a su baja corriente de alimentación (60uA), se produce un efecto de autocalentamiento reducido, menos de 0.1 ºC en situación de aire estacionario.

.Encapsulado

El sensor se encuentra disponible en diferentes encapsulados pero el mas común es el TO-92, una cápsula comúnmente utilizada por los transistores de baja potencia, como el BC548 o el 2N2904. La figura 1 nos muestra la disposición de sus pines, que son tres: alimentación (VCC), tierra (GND) y salida (OUT).



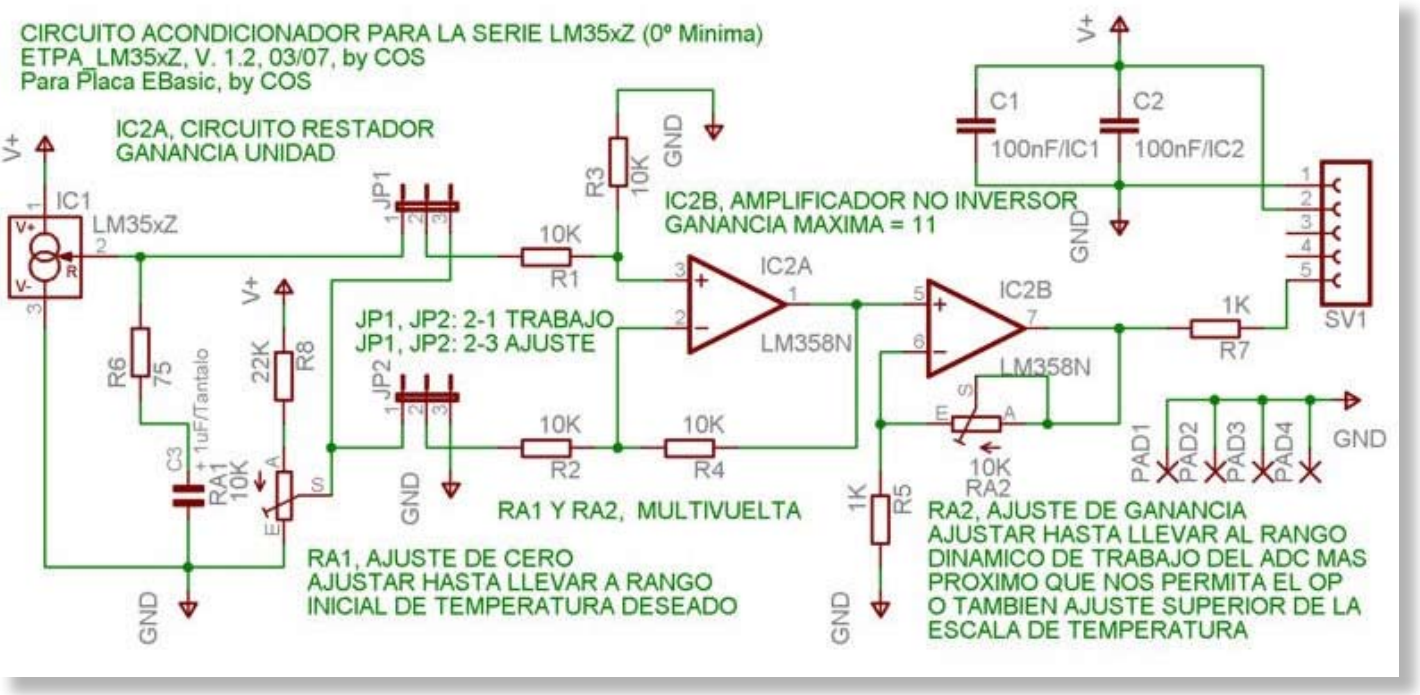
Cápsulas posibles y su pinout

Circuitos de aplicación: Acondicionador de señal para LM35x

El LM35 permite una precisión importante, pudiendo leerse fracciones de grado. Pero para ello es necesario hacer un adecuado tratamiento de la señal, ya que al trabajar con tensiones tan pequeñas, cualquier ruido o interferencia puede hacernos tomar una lectura errónea, o a veces, errática. Carlos Ortega Sabio ha desarrollado este circuito, que facilita la lectura del sensor mediante un microcontrolador.

.El circuito

El circuito acondicionador esta pensado para poder elegir el rango de trabajo del LM35, aunque teniendo en cuenta que la temperatura mínima que podremos leer será 0º, aunque fácilmente podría modificarse, y con la ayuda de la hoja de datos del sensor, trabajar con todo el rango de temperaturas disponible. Este es el circuito propuesto por Carlos:



Este es el esquema del acondicionador de señal para LM35x"

.Funcionamiento y ajuste

El circuito queda ajustado mediante RA2 a ganancia 10, pero para poder llegar a un valor mas próximo al rango dinámico de trabajo del ADC, se podría sustituir este potenciómetro (RA2) por uno de 20K, con lo que la ganancia máxima llegaría a 21 = (R5 + RA2) / R5. Este cambio permitiría llegar a los 3 voltios.

No conviene llevar la tensión de salida (pin 7) del IC2B a un valor muy próxima al de la alimentación, ya que este dejará de trabajar linealmente.

El valor de R5 parece redundante en la formulita, pero en realidad no lo es. No hay más que cambiar el valor de R5 por 2K y veréis que todo cambia. Hay que tener en cuenta que "el que reparte se lleva la mejor parte", en este caso el que diseña se reserva colocar los valores de los componentes, y en este caso ganancia de IC2B es igual a 1 + 10 = 11.

La ganancia del circuito restador se calcula teniendo en cuenta que siempre se cumpla lo siguiente: R1 = R2 y R3 = R4, con lo que la ganancia será igual a R4/R2. De esta forma podríamos simplificar el circuito realizando todo con un único amplificador operacional, por ejemplo con el CA3140. En este caso, para que el circuito funcionara igual que el otro, tendríamos que darle ganancia 10, y esto se haría cambiando el valor de R3 y R4 por 100K (R3 = R4 = 100K).

La salida del circuito se encuentra en el pin 5 del conector SV1, que vamos a suponer se conecta al pin RA4 de un microcontrolador con conversor ADC, como el PIC16F88.

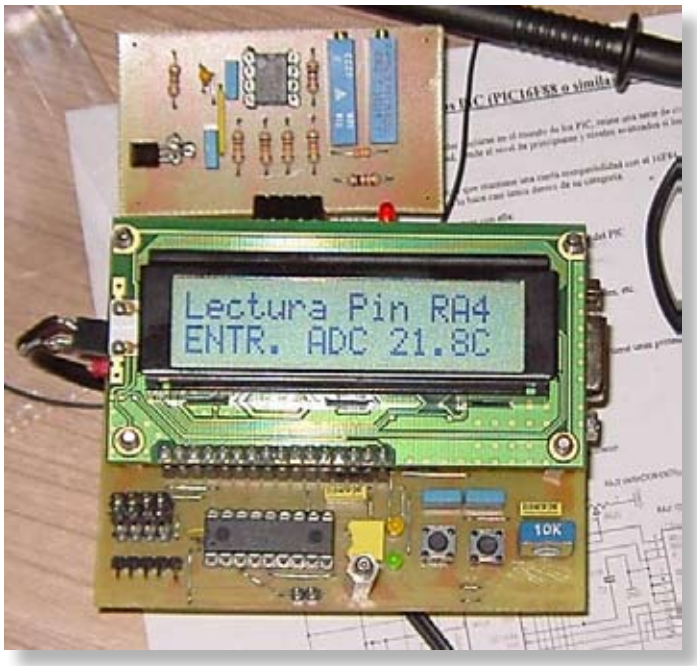
Si trabajamos sobre una placa para desarrollos, es fácil que el pin RA4 (que es nuestra entrada analógica), por un despiste u olvido, lo dejemos configurado como salida, creando una "lucha de niveles" entre la salida del IC2B y el pin del PIC. R7 evita este problema.

R8 cierra el entorno de voltaje de ajuste del RA1. Solo necesitamos unos pocos cientos de milivoltios para la vida más fácil ya que el RA1, de esta forma, nos dará un grado de precisión muy elevado.

El circuito acondicionador esta pensado para poder elegir el rango de trabajo del LM35

Teniendo esto en cuenta procedemos a ajustar el circuito para trabajar con un rango de temperatura, por ejemplo, de 15º a 30º centígrados.

Links:
Fairchild : http://www.fairchildsemi.com/
National Semiconductor : http://www.national.com/



//Carlos Ortega Sabio//
carlos.ortegasabio@ucontrol.revista.com.ar

retardador de la red eléctrica con el C.I. 555

Con este verdadero clásico de la electrónica construiremos un circuito práctico. El proyecto cumple la función de retardar la entrada de la red después de que la compañía eléctrica nos restablezca el servicio impidiendo que nos afecten los tan molestos y peligrosos cortes de electricidad que se producen antes de que quede totalmente restablecido el fluido eléctrico.

El circuito integrado conocido por todos como "555" lleva con nosotros más de 30 años. De hecho, ya estaba en el mercado antes de que casi todos nosotros entrásemos en el mundo de la electrónica.

Integra una parte analógica con otra digital que lo ha hecho durante muchos años indispensable en multitud de montajes, ya sea como protagonista o como circuito asociado. No es la misión de este articulo explicar el funcionamiento de este chip, ya se podría (y se ha hecho) escribir un libro sobre él.

El proyecto que nos ocupa en esta ocasión es una verdadera configuración de oscilador astable, pero trabajando como monoestable gracias al control del pin de Reset del 555. Es un temporizador de la conexión a la red eléctrica. No detecta los cortes eléctricos muy rápidos, aunque en general estos no producen perjuicio en nuestros equipos. Detecta los cortes con duraciones entre unos 100 y 200 milisegundos, aunque esto puede variarse modificando el valor de los componentes asociados y debido a la tolerancia de los mismos.

Tendremos especial cuidado en su manipulación, ya que trabajamos con tensiones peligrosas

Como puede verse en el esquema de la figura 1, el circuito posee un relé (RL1), que es el elemento utilizado para impedir que pase la corriente eléctrica durante un tiempo después de la restauración del servicio por

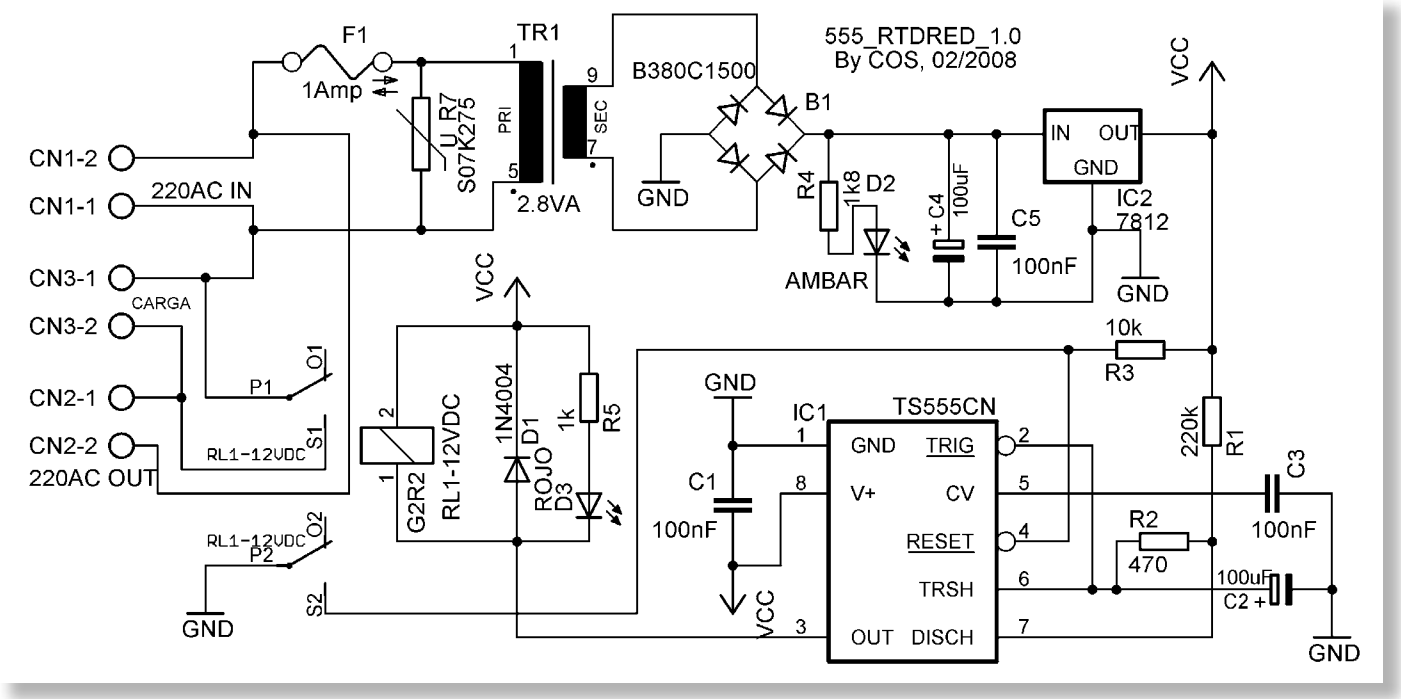


Figura 1: Esquema eléctrico del circuito, en su versión 1.0

parte de la compañía de electricidad mediante uno de sus dos juegos de contactos. El otro juego controla el pin de Reset del 555.

Cuando el circuito termina su temporización activa el relé, que restablece el suministro eléctrico y al mismo tiempo "congela" el circuito en este ultimo estado, dejándolo preparado para un nuevo ciclo en el caso de una segunda caída de la tensión de la red eléctrica.

La red se aplica al circuito mediante la bornera CN1 y la salida a controlar se conecta a CN2. Estañaremos todas las pistas asociadas a la ruta de 220 AC en nuestra placa de circuito impreso (figura 2). Con las mismas tendremos especial cuidado en su manipulado ya que trabajamos con tensiones peligrosas.

Aunque los contactos del relé no sufrirán desgaste haciendo maniobras continuas, hay que tener en cuenta que a pesar de soportar 5 Amperes por circuito, debemos mantener su corriente de trabajo en un valor menor, estando ésta limitada también por la superficie de las pistas de la placa del circuito impreso y por los mismas borneras. Para controlar corrientes más elevadas se puede utilizar este circuito para gobernar elementos de control de mayor potencia.

Para aumentar la inmunidad a los cortes de tensión incrementaremos la capacidad de C4, y mediante la red RC compuesta por R1 y C2 podremos variar el tiempo de espera previo a la reactivación del circuito después de ser restaurada la red eléctrica por la compañía.

Seguidamente paso a explicar su segunda utilidad, en la que los profesionales verán realmente una aplicación interesante. Mediante CN3 podemos conectar una carga de una potencia a elegir, en este caso 200Watios (dos lámparas de filamento de 100 Watios en paralelo), que permitirán limitar la corriente de entrada mientras dure el periodo de temporización. Esto permite que en los lugares donde hay fuentes de alimentación compuestas por filtros con condensadores de elevada capacidad estos obtengan una limitación a su demanda instantánea de corriente de carga, evitando tener que colocar costosos automatismos eléctricos de protección y rearme de la red eléctrica precisamente por estas elevadas corrientes instantáneas.

.Montaje

Una vez que tengamos la placa de circuito impreso taladrada y verificada se procederá a soldar los componentes, desde los más pequeños a los de mayor tamaño. Terminado este proceso procederemos con el estañado de las pistas de potencia (220AC), siendo generosos con el estaño.

Terminado todo esto comprobaremos que la distribución de las tensiones en la placa sean correctas, prestando especialmente atención la salida del estabilizador 7812 (aprox. 12VDC) y la presente entre los pines 1 y 8 (alimentación) del 555, que debe ser la misma tensión presente a la salida del estabilizador.

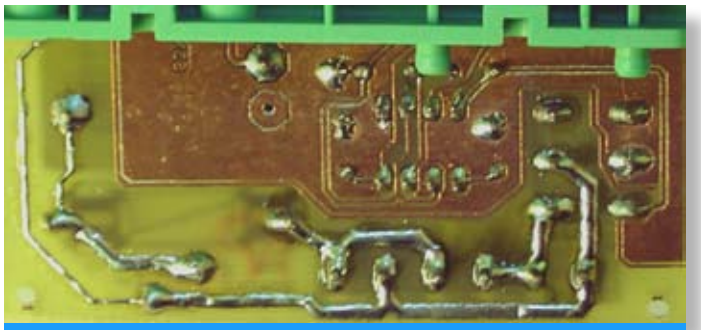


Figura 2: Detalle de las pistas de potencia recién estañadas, quedando la placa lista para su limpieza y barnizado.

Figura 3: Placa terminada, con todos sus componentes colocados.

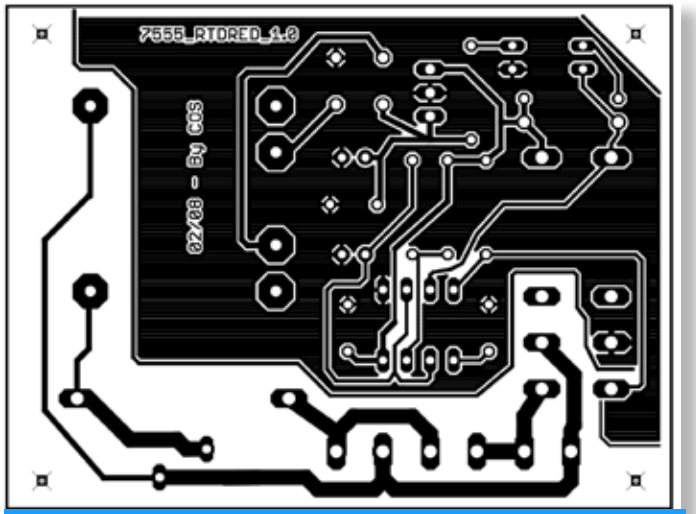


Figura 4: Vista de la parte inferior del circuito impreso.

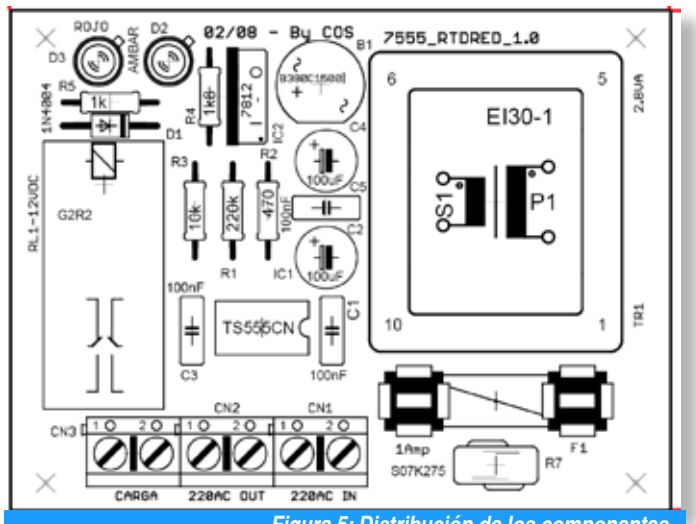


Figura 5: Distribución de los componentes.

Si la tensión que suministra la fuente de alimentación es superior en 4 o 5 voltios a la del regulador (12VDC) procederemos a colocar un disipador miniatura al mismo. Tendremos especial cuidado al orientar los componentes con polarización en la placa, como los diodos, diodos LED, condensadores electrolíticos y el propio circuito integrado.

Una vez comprobado el buen funcionamiento del circuito, procederemos a la limpieza de las pistas y soldaduras usando una brocha plana no muy grande con las cerdas cortadas a unos 3 o 4cm de su base e impregnada ésta en disolvente universal. Una vez secas las pistas, se procederá al barnizado de las mismas con una ligera (pero consistente) capa de barniz en spray para uso en circuitos impresos.

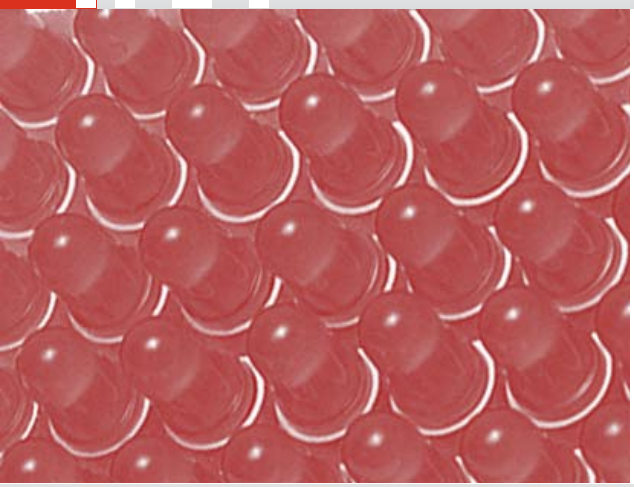
Lista de materiales:

Ref.	Descripción
B1	PUENTE RECTIFICADOR B380C1500
C1	CONDENSADOR 100nF CERAMICO, MKT, MKP
C2	CONDENSADOR 100uF/25V ELECTROLITICO
C3	CONDENSADOR 100nF CERAMICO, MKT, MKP
C4	CONDENSADOR 100uF/25V ELECTROLITICO
C5	CONDENSADOR 100nF CERAMICO, MKT, MKP
CN1	BORNERA SEPARACION 5mm PIN, TIPO AK500/2
CN2	BORNERA SEPARACION 5mm PIN, TIPO AK500/2
CN3	BORNERA SEPARACION 5mm PIN, TIPO AK500/2
D1	DIODO 1N4004
D2	LED 3MM AMBAR
D3	LED 3MM ROJO
F1	PORTAFUSIBLE PARA CIRCUITO IMPRESO CON FUNDA PROTECTORA Y FUSIBLE DE 1AMP
IC1	CIRCUITO INTEGRADO CMOS TS555CN
IC2	CIRCUITO INTEGRADO LM7812CV
R1	RESISTENCIA 220k, 1/4W
R2	RESISTENCIA 470, 1/4W
R3	RESISTENCIA 10k, 1/4W
R4	RESISTENCIA 1k8, 1/4W
R5	RESISTENCIA 1k, 1/4W
R7	VARISTOR 275V
RL1-12	VDC RELE OMRON G2R-2
TR1	TRANSFORMADOR ARISTON TR-4112 2.4VA 230V/12V

Además necesitaremos:

- PLACA DE CIRCUITO IMPRESO SIMPLE CARA
- BROCA 0.7mm Y 1mm
- BASE PARA MONTAR EN CARRIL

//Ariel Palazzesi//
arielpalazzesi@gmail.com

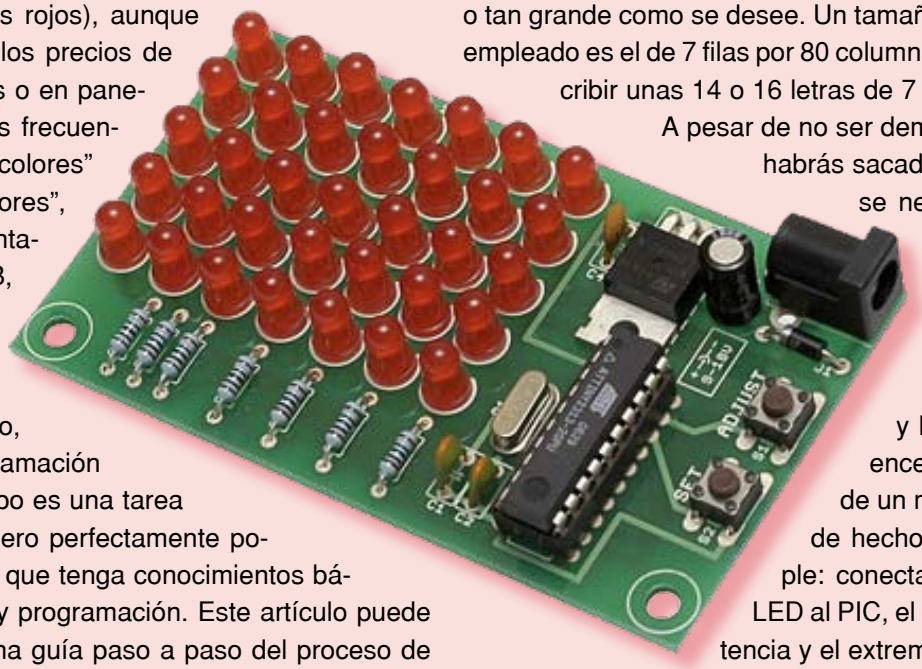


matrices de LEDs

La gran mayoría de los aficionados a la electrónica, tarde o temprano, se propone la construcción de un cartel basado en una matriz de diodos LEDs. El propósito de este artículo es explicar, de forma clara y sencilla, la forma de hacerlo.

Un cartel formado por varias filas y columnas de LEDs, convenientemente programado, puede servir para pasar mensajes publicitarios, decorar nuestra habitación, ordenador o lo que se nos ocurra. No solo se trata de un proyecto más que interesante para llevarlo a cabo como hobbysta, sino que puede resultar interesante como un producto comercializable. Es que estas matrices, que en algunos países se las conoce como "cartel de LEDs" o "Publik", son un recurso muy frecuentemente utilizado con fines publicitarios o informativos.

Desde el punto de vista del hardware, básicamente consiste en una matriz de píxeles similar a los de la pantalla de un ordenador, generalmente de un solo color (la mayoría de las veces rojos), aunque con el descenso de los precios de los LEDs individuales o en paneles, es cada vez más frecuentes ver carteles "bicolores" o incluso "multicolores", aprovechando la ventaja del los LEDs RGB, que pueden mostrar cualquier color.



Como es de suponer, el desarrollo, construcción y programación de un cartel e este tipo es una tarea bastante compleja, pero perfectamente posible para cualquiera que tenga conocimientos básicos de electrónica y programación. Este artículo puede ser utilizado como una guía paso a paso del proceso de creación de un cartel de este tipo. Y aunque no construyas uno, leyéndolo aprenderás algún truco útil que podrás emplear en otro proyecto.

Para mantener el nivel de la explicación dentro de lo razonable, y para no gastar una fortuna en nuestro cartel, lo diseñaremos monocromático, utilizando LEDs de color rojo únicamente. Las dimensiones de la matriz utilizada para mostrar los textos la decidirá cada uno de los lectores, pudiendo ser tan pequeña (7 filas y 5 columnas) o tan grande como se desee. Un tamaño razonable y muy empleado es el de 7 filas por 80 columnas, que permite escribir unas 14 o 16 letras de 7 "pixeles" de altura. A pesar de no ser demasiado grande, ya habrás sacado la cuenta de que se necesitan 560 LEDs individuales para armar el cartel.

En el número 2 de la **revista uControl**, en "PICs y LEDs" vimos como encender un LED desde un microcontrolador. Y de hecho es algo muy simple: conectamos el ánodo del LED al PIC, el cátodo a una resistencia y el extremo de la resistencia a +V. Cuando el pin del microcontrolador está en "1", el LED enciende. Pero lamentablemente este esquema no sirve para la construcción de un cartel matricial como este, ya que al utilizar cientos de LEDs necesitaríamos tener un microcontrolador que tenga como mínimo ese número de pines de salida (y por supuesto, no existe).

El secreto, por supuesto, está en el multiplexado. Esta técnica permite utilizar unos pocos pines de E/S del microcontrolador para manejar una serie de circuitos integrados que se encarguen de excitar los LEDs. Hay varias maneras, y muchos modelos diferentes de circuitos para hacer esto.

Pueden usarse un tipo de integrado digital llamado "LATCH", que básicamente es una memoria en la que escribimos un valor, y lo mantiene en sus salidas hasta que nosotros lo indiquemos. De esta manera, usando varios latches podríamos encender los LEDs por turnos, rápidamente para que no se note el parpadeo, y de esa manera formar una palabra en el cartel.

Matriz de LEDs RGB, de 8x8

Otra forma es utilizar un registro de desplazamiento como los analizados en el primer número de la **revista uControl**. Y de hecho, es de esta forma cómo vamos a diseñar nuestro cartel. Como vimos en esa oportunidad, un registro de desplazamiento funciona de la misma manera en que funciona una cola de gente que espera para entrar en un cine. Por un extremo de la cola van ingresando las personas que llegan, y por el otro van saliendo de la fila. En un registro de desplazamiento, en lugar de personas tenemos "0" y "1". Lo bueno de esto es que para "meter" datos ("0"s y "1"s) en el registro de desplazamiento solo hacen falta tres pines del microcontrolador, independientemente de lo largo que sea.

Estos pines se encargan de tres tareas: Uno de ellos, al que denominaremos "DATA" es el encargado de decirle al registro de desplazamiento que lo que introduciremos es un "0" o un "1". El segundo se encarga de avisar al registro que el dato ya está listo para ser ingresado, y lo llamaremos "CLOCK". Y el ultimo, que no es indispensable, es el "RESET", que se encarga de "vaciar" la fila escribiendo "0"s en todas las salidas del registro.

Para desarrollar nuestro ejemplo utilizaremos el circuito integrado **74HC164N**, que es un registro de desplazamiento de 8 bits. Es decir, con el se puede armar una "fila" de 8 "personas". Para construir un cartel de 80 columnas, necesitaríamos utilizar 10 de estos integrados, uno a continuación del otro. Afortunadamente, este integrado cuesta solo centavos.

En la figura 1 podemos ver la función de cada uno de los pines del 74HC164N y en la figura 2 de que forma podemos conectar uno a continuación del otro para obtener un registro de desplazamiento de cualquier longitud.

Bien, con el esquema explicado podemos encender los LEDs que queramos de una fila de 80 bits de largo. Si en el registro de desplazamiento introducimos "11111…111", los 80 LEDs estarán encendidos. Si queremos encender uno por medio, escribiremos "10101…01". Por supuesto, cuando lleguemos a la parte de la programación veremos cómo se ingresan uno a uno los "0" y "1" en el registro.

desarrollar nuestro ejemplo utilizaremos el circuito integrado 74HC164N

En este punto puede ser necesario analizar el tema de las filas. Si tenemos, por ejemplo, un cartel tiene 7 filas, y lo explicado recién sirve para manejar solo una de ellas ¿debemos utilizar un registro de desplazamiento para cada una de las filas restantes? Afortunadamente, la respuesta es no. Si bien podríamos utilizar 7 registros de este tipo, la cantidad de circuitos integrados necesarios (56 de ellos), la complejidad del circuito impreso y el costo implicado lo hacen poco aconsejable. Nosotros aprovecharemos un "defecto" del ojo humano, que mantiene la imagen vista durante unos 20 o 30 milisegundos, para "dibujar" una fila a la vez, pero muy rápidamente, de forma que todo el car-

Reloj gigante basado en una matriz de LEDs

tel parezca estar encendido a la vez. Si, se trata de un sistema similar al empleado en el cine o en la televisión.

Si seguimos pensando en un cartel de 7 filas y 80 columnas, sin utilizar registros de desplazamiento necesitaríamos 560 pines de entrada/salida. Con el esquema propuesto solo necesitamos 7 de ellos para seleccionar la fila a escribir, y tres para manejar el registro de desplazamiento. Es decir, un PIC de 3 euros y 18 pines serviría perfectamente para realizar el proyecto.

.¿Cómo funciona la matriz?

Como dijimos antes, la pantalla está formada por una serie de filas y columnas. La intersección entre ambas contiene un LED. Para que este encienda, tiene que recibir simultáneamente un "0" en la fila, y un "1" en la columna. Cuando se dan estas condiciones, la electrónica de la placa se encarga del encendido del LED en cuestión. La forma de generar un mensaje sobre el display es relativamente sencilla, si nos atenemos al siguiente algoritmo:

1) Apagar todas las filas.
2) Escribir los valores correspondientes a la primer fila en el registro de desplazamiento, teniendo en cuenta que el primer digito binario colocado corresponde al último LED de la fila, y el ultimo en poner al de la primer columna.
3) Encenderla primer fila, esperar un tiempo, y volver a apagarla.
4) Repetir los pasos 2 y 3 para las filas restantes.

El tiempo de la demora debe ser tal que permita una visualización correcta, sin molestos parpadeos y con los LEDS brillantes. Hay que tener en cuenta que si utilizamos tiempos mayores para el encendido de cada fila, el brillo de los LEDS será mayor, pero también aumentará el parpadeo. La forma de transformar este algoritmo en un programa funcional depende de cada programador, y puede ser más o menos complejo según se permitan diferentes tipos de caracteres, animaciones, etc.

Un punto a tener en cuenta es la intensidad del brillo que puede proporcionar el tipo de LED que utilicemos. Un LED, utilizado en aplicaciones "normales", se alimenta con unos 3V y requiere unos 15mA (varia ligeramente de un modelo a otro) para brillar con una buena intensidad. En caso de un típico cartel de 7 filas, a pesar de que las veremos encendidas al mismo tiempo, cada LED solo estará encendido la séptima parte del tiempo, por lo que su brillo será siete veces inferior al normal, y nuestro cartel apenas será visible.

Afortunadamente esto también tiene solución: dado que los tiempos que permanecerá encendido cada LED no superará los 20 o 30 milisegundos, podremos hacerles circular una corriente mayor a la nominal sin que lleguen a dañarse, con lo que brillarán mucho más intensamente, dando como resultado un cartel perfectamente visible.

Respecto de los LEDs, podremos utilizar LEDs discretos (y soldar 1120 terminales) o comprar "paneles" de 7x5 LEDs que tienen unos 14 o 16 terminales (según el modelo), estando ya interconectados en forma de matriz. Quizás sea esta la mejor alternativa.

.El hardware

Dado que nuestro hipotético cartel tiene fines meramente educativos, y la intención mantener su costo lo más bajo posible para que cada lector pueda construirlo, por lo que intentaremos realizarlo en base a un microcontrolador pequeño, como el **PIC16F628A**. Si el lector necesita un cartel de mayor tamaño o con capacidad para almacenar textos o imágenes más extensos, deberá utilizar algún micro con mayor capacidad y velocidad.

La utilización de una memoria EEPROM externa de un tamaño bastante grande, como la **24C256**, nos brinda la posibilidad de almacenar mucho texto en ella. Por supuesto, esto también puede ser ampliado con mucha facilidad.

Dividiremos el esquema electrónico del cartel en dos partes: en primer lugar veremos toda la lógica de control, y en segundo, la "pantalla" con el registro de desplazamiento. A la hora de llevarlo a la práctica se puede incluso hacer dos circuitos impresos por separado. Esto le permitiría al lector experimentar con otros controladores sin necesidad de volver a montar la placa de los displays, o viceversa.

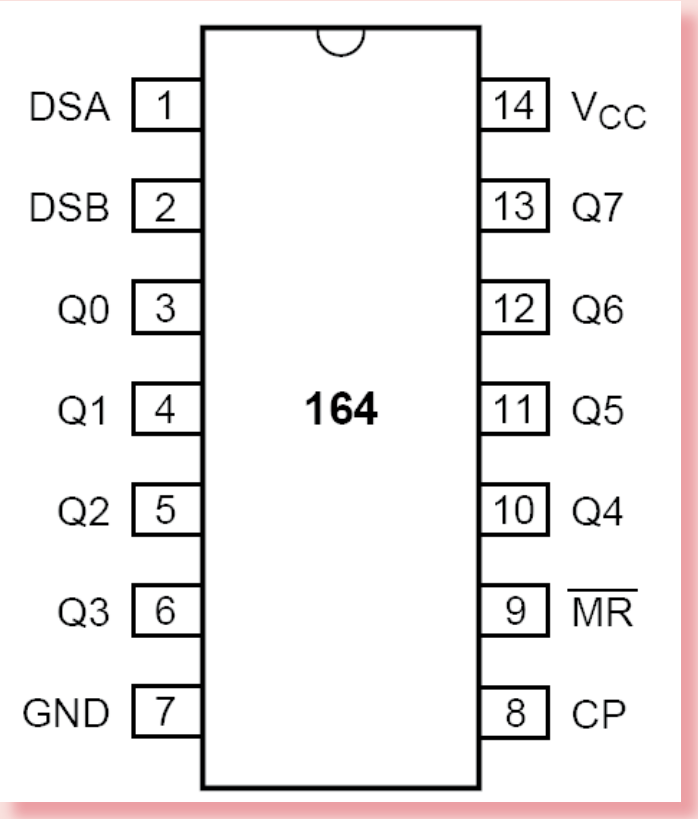


Figura1: Pinout del 74xx164N

.El circuito controlador

Este es el cerebro de nuestro cartel. Será el encargado de gestionar el encendido de cada LED mediante órdenes enviadas a las columnas mediante el registro de desplazamiento y a las filas.

Como una fila tendrá muchos LEDs (80, por ejemplo)y existe la posibilidad que en algún momento puedan estar todos encendidos, no podemos conectarlas directamente a pines de E/S del PIC, porque la corriente que demandarían haría que el puerto del microcontrolador pase a mejor vida. Para evitar esto, utilizaremos en medio un transistor capaz de manejar la corriente requerida.

Analicemos el circuito de la figura 3. El centro de todo es el microcontrolador **PIC16F628A,** que tiene su pin de **RESET** conectado a un pulsador y una resistencia de 10K. Este pulsador permite reiniciar el cartel cuando lo necesitemos. También se ha implementado un circuito de reloj externo, basado en un cristal de 4 MHz y dos condensadores de 22 nF. Esto le permite al PIC ejecutar un millón de instrucciones por segundo, más que suficientes para este proyecto.

Los pines 1 y 2, correspondientes a los terminales **A2** y **A3** del microcontrolador, se han utilizado para acceder a una memoria EEPROM del tipo **24C256**. Esta memoria es de acceso serial (por eso necesitamos solo dos pines del PIC para usarla) mediante el protocolo **I2C**, y tiene capacidad para almacenar 32.768 Bytes. Si nuestro programa hace uso de ella, podemos guardar allí 32.768 caracteres (con el display en modo texto) o más de 450 pantallas de 7x80 píxeles en modo gráfico. Si resultara insuficiente, puede ponerse una memoria de mayor capacidad, siempre consultando la hoja de datos de la misma para asegurarnos su compatibilidad con la del ejemplo.

Todo el puerto B del PIC está dedicado a controlar las filas del cartel. Como ya habrán notado, tenemos 8 salidas para filas, y nuestro cartel tiene solo 7 filas. Efectivamente, la fila 8 no se utilizará si nuestra "pantalla" está construida con módulos LED de 7x5, pero el circuito de control está preparado para el uso (en caso de que alguien los prefiera) de módulos de 8x8 o bien para crear un cartel de 8 filas mediante el uso de LEDs sueltos. Quienes utilicen módulos de 7x9 pueden ahorrarse el transistor de la fila 8.

Por último, los pines 17 y 18, correspondientes a los terminales **A0** y **A1** del microcontrolador se encargan de la gestión del registro de desplazamiento. El programa deberá generar los pulsos de reloj necesarios por el pin 18, y "meter" los datos en el registro por el pin 17.

Como cada fila tendrá muchos LEDs, no podemos conectarlas directamente a pines de E/S del PIC.

No hemos incluido una fuente de alimentación. Cualquier fuente comercial (o construida en casa) que sea capaz de entregar 5V y 2A será suficiente. Esos 5V deben estar bien regulados, y por supuesto, el software deberá estar escrito correctamente, es decir, no encender varias filas al mismo tiempo, ya que el consumo de todo el cartel encendido sería de unos 80 x 70 x 20mA = 11.2 A, lo que podría destruir la fuente en caso de que no cuente con protecciones adecuadas.

.El display

Esta es la parte del proyecto que todo el mundo va a mirar, así que debemos ser prolijos al montarlo. Como puede verse en el esquema eléctrico de la figura 4, hemos utilizado un total de 10 circuitos integrados **74HC164N** para construir el registro de desplazamiento de 80 bits de largo, uno para cada columna. Como explicamos, si alguien quiere hacer un cartel más largo o más corto, deberá poner más o menos integrados.

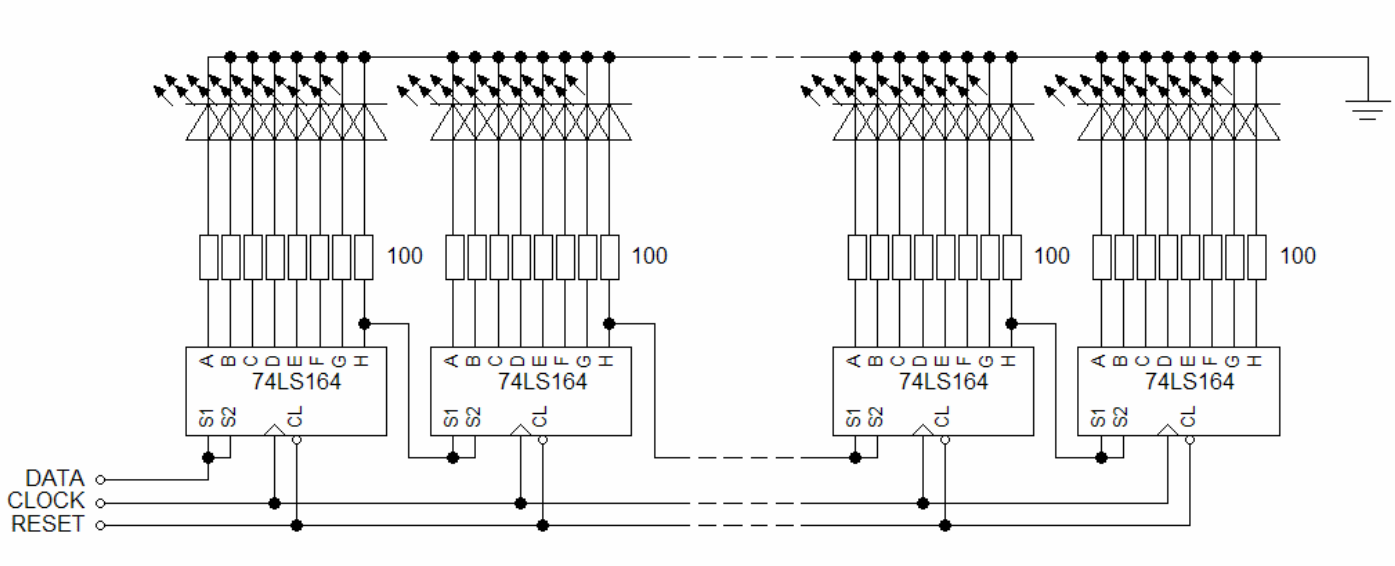


Figura2: Varios 74xx164N trabajando en conjunto

Si miramos el esquema del display, veremos que en la parte superior se muestra como está conectado cada LED dentro de la matriz de 5x7. Esto es importante tenerlo en cuenta a la hora de comprar los módulos, ya que hay una gran cantidad de modelos, y algunos de ellos tienen los LEDs conectados en el sentido inverso.

Cada display también difiere en la función de cada terminal, por lo que se debe estar a atento a la hoja de datos para diseñar el circuito impreso apropiado, y conectarlos como corresponda.

En el dibujo del circuito no hemos representado los 16 módulos ni los 10 circuitos integrados, por una cuestión de espacio, pero es fácil darse cuenta de qué forma se conectan las filas y columnas de los demás displays a cada **74HC164N.**

No utilizaremos el pin de **RESET** de los **74HC164N.** En lugar de ser controlados desde el microcontrolador, cada **RESET** está puesto a +5V, de forma que el integrado funcione continuamente. Si por algún motivo se desea borrar la pantalla, basta con enviar 80 “0”s al registro de desplazamiento y listo. El tiempo empleado para esa tarea es despreciable, ya que el microcontrolador estará ejecutando 1 millón de instrucciones por segundo. El utilizar una línea de control menos nos permitirá tener una placa de circuito impreso ligeramente más sencilla.

Cada salida de los **74HC164N**, como dijimos, se conecta a una columna de la serie de displays. Esta conexión se efectúa mediante un resistor de 1/8 de Watt, que en el esquema se ha dibujado con un valor de 330 ohm. Ese fue el valor adecuado para el tipo de módulos que conseguimos para hacer el prototipo, pero su valor variará de un modulo a otro. Se puede montar solo un display con resistores de 330 ohms, y ver como es el brillo de los LEDs. Si es escaso, se puede bajar el valor a 220 o 100 ohms. Con eso debería ser suficiente

.El software

Ahora nos toca abordar la programación del hardware propuesto. El cartel del LEDs que estamos construyendo puede adoptar diferentes tamaños de acuerdo a las necesidades o componentes que cada uno consiga. Esto hace que sea imposible proporcionar un programa específico que funcione en cualquier versión de cartel que se haya construido, pero sin embargo podemos hacer algo mucho mejor: ver de qué manera se escribe un programa

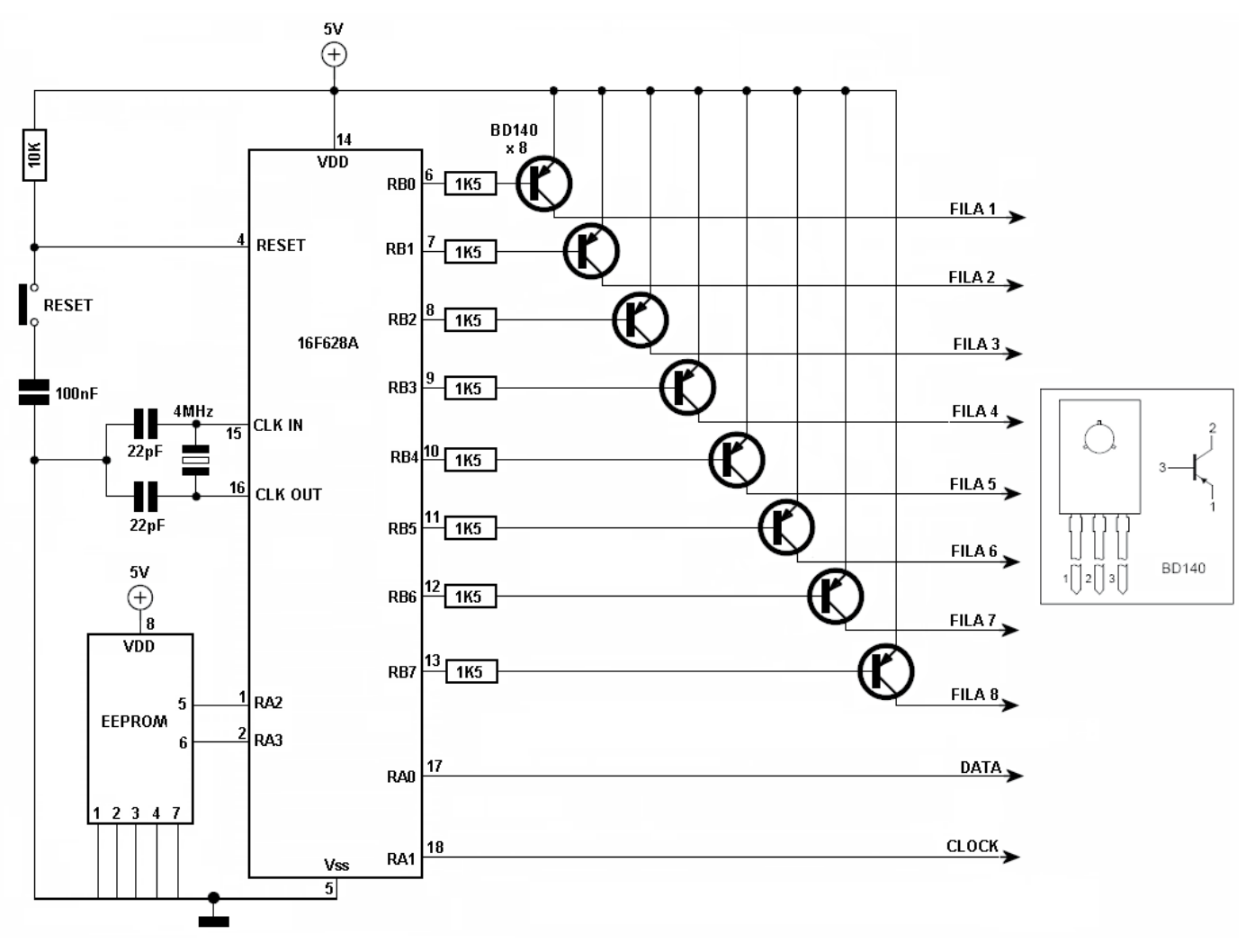


Figura3: Este será el cerebro del cartel

de este tipo en **BASIC** (del **PIC SIMULATOR IDE**) para que cada uno lo adecue a su proyecto.

Debemos pensar en un programa que nos permita mostrar píxeles individuales representados sobre la pantalla de nuestro cartel. Sigamos con el ejemplo del cartel de 80 columnas y 7 filas de altura, recordando que todo lo que expliquemos puede ser adecuado para carteles de otro tamaño.

Lo primero que necesitamos saber es que el "barrido" del cartel debe hacerse por filas. Es decir, mostraremos el contenido de la primera fila, esperamos un tiempo determinado (unos pocos milisegundos), mostramos el de la segunda fila, esperamos nuevamente, y así hasta llegar a la última fila, tal como se expresa en el algoritmo visto mas arriba.

Dividiremos el cartel en dos partes: la lógica de control, y la "pantalla.

El motivo de no emplear las columnas para realizar el barrido es que como son más numerosas, el tiempo total que se necesita para "escribir" por filas es mucho menor que el necesario para escribir por columnas, y en la práctica eso significa que el brillo de nuestro cartel será mucho mayor si lo hacemos por filas, ya que cada LED permanecerá encendido 1/7 del tiempo. Si lo hiciésemos por columnas, cada LED estaría (en este ejemplo) encendido solo 1/80 del tiempo, por lo que su brillo seria unas 10 veces menor.

Ahora bien, el primer problema a resolver es ¿Cómo escribo los datos de una fila del cartel? Esto tiene una solución más que simple: solo debemos introducir en el registro de desplazamiento los "0" y "1" necesarios para que los LEDs que queremos estén encendidos en esa fila tengan +V en sus ánodos. Por supuesto, mientras hacemos esto todos los pines del microcontrolador que controlan las filas deberán estar apagadas, para que no se perciba una débil luminosidad en todos los LEDs de la fila que estamos escribiendo a medida que pasan los datos a través del registro.

El primer valor que se debe "meter" en el registro de desplazamiento es el que corresponderá a la última columna. A medida que vamos ingresando los siguientes, se van desplazando hacia el final del cartel. Cuando hayamos introducido el valor número 80 (que corresponderá a la primera columna) el primer valor que metimos habrá llegado a su posición. En ese momento tenemos todo el registro escrito, y ya podemos activar la salida del PIC que corresponde a esa fila en particular.

El tiempo que debe estar encendida la fila se puede determinar empíricamente, pero por lo generan unos 10 milisegundos es suficiente. Si tenemos 7 filas, 10 milisegundos de demora permitirían escribir todo el cartel en unos 70 milisegundos, por lo que obtendríamos un máximo de 1000/70 = 14 "frames" por segundo. Este es un muy buen valor para una pantalla de este tipo, ya que solo estamos mostrando un texto y no un video.

En los cálculos anteriores no tuvimos en cuenta el tiempo que se demora en escribir los 80 valores en el registro de desplazamiento. Veamos porque: cada valor ingresado en el registro de desplazamiento demora unos 2 microsegundos. Es decir, demoramos 2 x 80 = 160 millonésimas de segundo en escribir toda la fila. Si multiplicamos este valor por 7 tendremos en tiempo que nece-

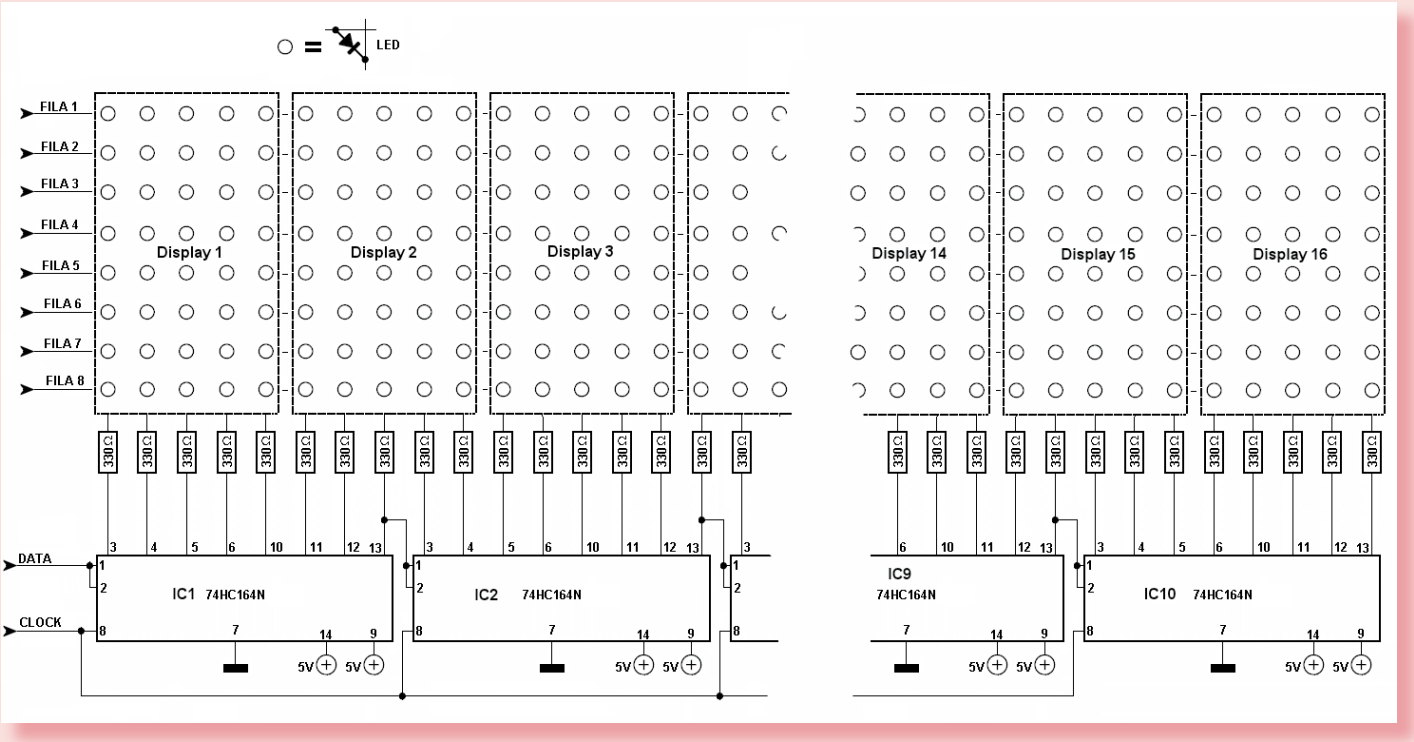


Figura4: Circuito de un panel de LEDs de 7x80

sitamos para escribir las 7 filas del cartel, lo que nos da 1136 millonésimas de segundo, es decir, poco más de 1 milésima. Este es un tiempo despreciable frente a las 70 milésimas que nos tomamos para mostrar la imagen completa, y podemos despreciarla.

Ahora vamos a ver, en **BASIC**, como hacer para escribir un valor en el registro de desplazamiento. Recordemos que el dato ingresa al registro en el momento que se produce la transición de "0" a "1" del pulso de **CLOCK**, por lo que se deberán seguir los siguientes pasos para ingresar cada uno de los 80 valores correspondientes a cada fila:

1) Fijar el valor del dato a escribir (si DATA es 1, hacer PORTA.0 = 1, si no PORTA.0 = 0)
2) Poner la línea de CLOCK en estado bajo (PORTA.1 = 0).
3) Esperar un 1 microsegundo (WaitUs 1)
4) Poner la línea de CLOCK en estado alto (PORTA.1 = 1). En este punto el dato entra efectivamente en el registro de desplazamiento.
5) Esperar un 1 microsegundo (WaitUs 1)
6) Fin

En BASIC, si hemos declarado que

```
Symbol clock = PORTA.1
Symbol data = PORTA.0
```

Un "0" se escribiría así:

```
data = 0
clock = 0
WaitUs 1
clock = 1
WaitUs 1
```

Y un "1" de la siguiente manera:

```
data = 1
clock = 0
WaitUs 1
clock = 1
WaitUs 1
```

Para escribir los 80 valores de la fila necesitamos hacer una subrutina que, tomando 10 bytes de la memoria EEPROM (10 bytes x 8 bits = 80 bits, es decir, una fila completa) los vuelque al registro de desplazamiento.

Si repetimos 7 veces este procedimiento, tendríamos una pantalla de 7x80 completa. Eso significa que en la EEPROM cada pantalla va a necesitar de 70 bytes (10 bytes por fila, 7 filas) para almacenar el mapa de bits correspondiente.

Veamos un ejemplo de cómo podría ser la subrutina encargada de escribir un byte tomado de la EEPROM en el registro de desplazamiento, a la que hemos llamado *escriboByte:*

```
escriboByte:
     For columna = 1 To 8
         If dato.7 = 0 Then
data = 0
clock = 0
WaitUs 1
clock = 1
WaitUs 1
         Else
data = 1
clock = 0
WaitUs 1
clock = 1
WaitUs 1
         Endif
         aux = ShiftLeft(dato, 1)
     Next columna
Return
```

Esta función debe ser llamada 10 veces para escribir la fila completa, con el valor a escribir guardado en la variable "dato". El motivo por el cual el bucle FOR-NEXT toma los bits del byte desde el 7 hasta el 0 se debe a que justamente el último bit es el que debe ingresar primero al registro de desplazamiento, tal como explicamos antes.

Debemos partir de la base de que la información de la EEPROM la vamos a grabar desde un ordenador, y que seguramente crearemos un programa que permita, a partir de un texto determinado, generar los bits individuales que componen el bitmap de cada pantalla del cartel. Esto simplifica muchísimo la programación del microcontrolador, ya que solo debe dedicarse a leer la EEPROM y volcar su contenido al registro de desplazamiento, sin tener que "dibujar" las letras a partir de una tabla ni nada por el estilo.

.Textos animados

Para animar el texto mostrado en el display hay dos opciones. La primera de ella es que, una vez que el bitmap de la EEPROM ha sido mostrado en la pantalla, comencemos a redibujarlo continuamente (si no lo hacemos, el texto desaparecerá de la pantalla) pero cada un tiempo determinado (1 segundo por ejemplo) escribimos un bit "0" más en cada fila. Es decir, escribimos 81 bits en el primer segundo, 82 en el segundo, etc. Esto hará que el texto se desplace de izquierda a derecha, y es la animación más fácil de implementar. Sin embargo, lo normal es que los textos de desplacen en sentido contrario, por lo que nuestro programa debería hacer lo siguiente: comenzar escribiendo 80 "0"s en el registro **antes** de enviar la información de la fila, luego escribir 79 "0"s, y así sucesivamente. De esa manera, el texto al principio no será visible (estará "dibujado" a la derecha, fuera del registro de desplazamiento), y luego a medida que el numero de "0"s

escritos va disminuyendo, comenzara a ser visible, entrando desde la derecha.

La segunda manera es que el software que escribe los datos en la EEPROM guarde cada "cuadro" de la animación, uno a continuación del otro, y que el PIC se limite a escribir cada cuadro leído durante (por ejemplo) un segundo. Esto vuelve a facilitar mucho la programación del PIC, a la vez que permite animaciones mucho más complicadas. Por supuesto, el precio a pagar es el espacio de memoria EEPROM requerido para implementar esta técnica.

.Conclusión

Si bien no se trata de un proyecto completo y concreto, es muy posible que este artículo haya servido para que te animes a encarar el diseño y construcción de tu propio cartel de LEDs. Solo hemos intentado exponer la forma (o al menos una de ellas) en que se puede atacar este problema, de modo que cualquier hobbysta o estudiante se anime y pueda hacer realidad su propio cartel animado. No se trata de un proyecto sencillo, pero una vez terminado seguramente los llenara de orgullo. Desde aquí los animamos a que encaren este proyecto, y que por supuesto, nos comenten los resultados obtenidos.

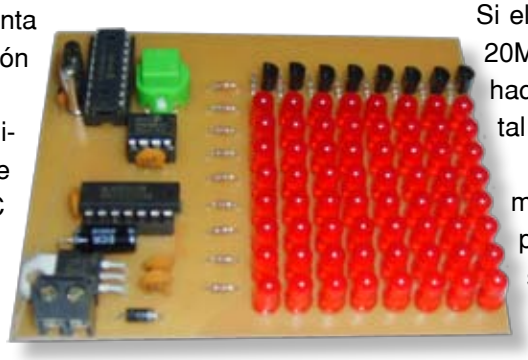
//Ariel Palazzesi//
arielpalazzesi@gmail.com

matriz de LEDs de 8x8

Como complemento a la nota de tapa, desarrollaremos un display formado por una matriz de LEDs de 8 filas y 8 columnas. Para hacer más interesante el circuito, hemos incluido una memoria EEPROM externa del tipo 24C256, en la que podremos almacenar bloques de píxeles que luego serán representados por la pantalla.

El proyecto que desarrollaremos es una buena herramienta para el aprendizaje. El microcontrolador empleado es de los más pequeños, un PIC16F628A con 2 KBytes de memoria FLASH. Sin embargo, se puede utilizar un PIC16F627A (que tiene la mitad de memoria) o bien un PIC16F648A (que tiene el doble).

Este microcontrolador puede parecer pequeño, pero es más que suficiente para poder manejar el display y la memoria EEPROM externa. Por poco dinero podemos disponer de una buena herramienta para experimentar con la programación de las matrices de LEDs.



Como decíamos, hemos incluido una memoria EEPROM 24C256 de 32 KBytes, a la que se accede vía I2C y que puede utilizarse para almacenar el contenido de 4096 "pantallas", de 8x8 píxeles cada una, que el microcontrolador podría desplegar una a continuación de otra, a modo de video de baja resolución.

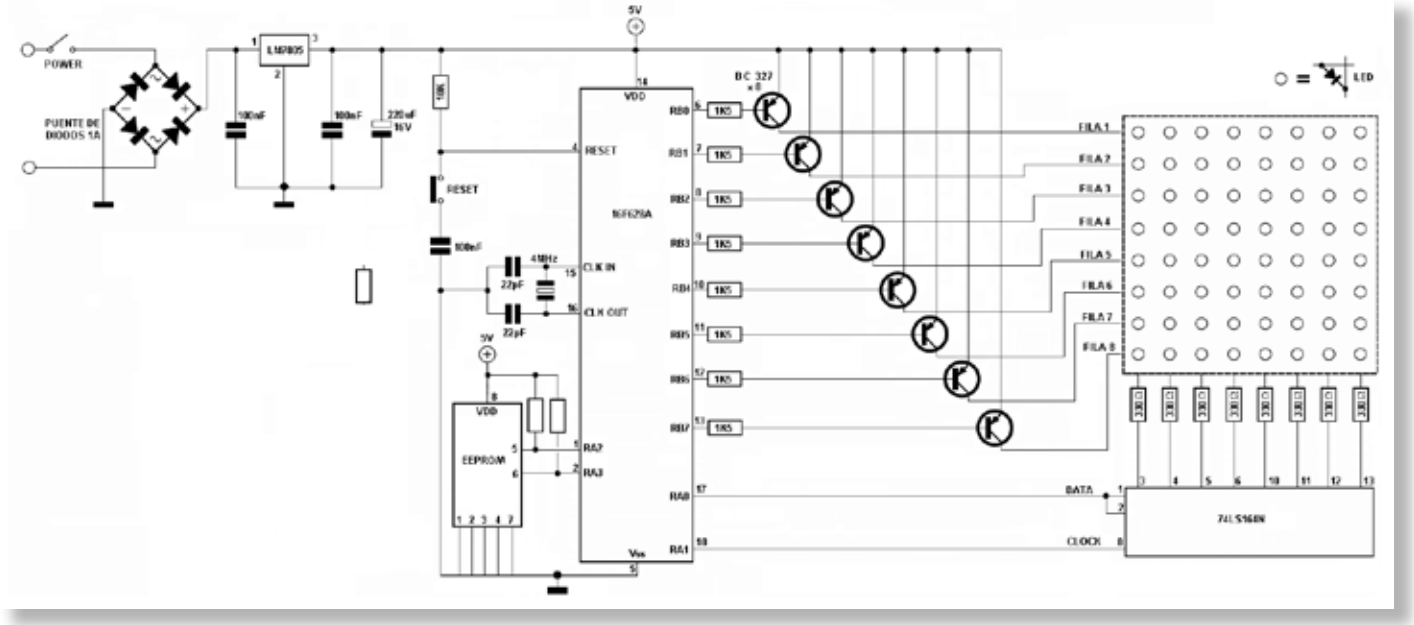
Las columnas de la matriz, tal como se explica en el articulo de tapa, se seleccionan mediante un registro de desplazamiento de 8 bits de longitud, implementado a partir de un circuito integrado 74HC164N.

.El circuito

El circuito eléctrico de este proyecto es muy sencillo, y se basa en el microcontrolador PIC16F628A. El oscilador del PIC esta construido a partir de un cristal de 4 MHz y los dos condensadores de 22 picofaradios de rigor. Si el lector prefiere utilizar la versión de 20MHz de este microcontrolador, puede hacerlo simplemente cambiando el cristal por uno de ese valor.

La pantalla esta construida mediante una matriz basada en 8 filas por 8 columnas de diodos LEDs, con sus ánodos controlados desde los pines del PORTB del microcontrolador, mediante un transistor 2N3906 que se encarga de proporcionar la corriente suficiente para encender los 8 LEDs de la fila.

La fila inferior corresponde al pin B0, la siguiente



Esquema eléctrico del proyecto

al B1 y así, hasta la fila superior, que esta conectada al pin B7. Cuando queramos programar el microcontrolador, deberemos configurar todo el puerto B como salidas.

El puerto A tiene a su cargo el control del 74HC164N, que a su vez se encarga de seleccionar las columnas que deben estar activas en cada momento. Entre cada salida del 74HC164N y los LEDs hemos colocado un resistor para limitar la corriente que circula por ellos. Si el brillo de los LEDs es muy bajo, puede probarse con valores más pequeños para estos resistores. El pin DATA del 74HC164N es controlado desde A1 y los pulsos de CLOCK los proporciona el pin A0.

También hemos incluido una memoria EEPROM externa del tipo 24C256

La memoria EEPROM también depende de los buenos oficios del puerto A, con la línea SCL conectada al pin A2 y la línea SDA conectada al pin A3. Ambas líneas están puestas a +V mediante resistores de 10K.

La alimentación se ha resuelto mediante un regulador de voltaje tipo LM7805 y sus componentes asociados. Un diodo se encarga de proteger el circuito por si involuntariamente conectamos la alimentación con la polaridad invertida. La bornera es la encargada de vincular la fuente de corriente continua de entre 9V y 12V encargada de alimentar la placa.

Un pulsador, junto a una resistencia de 10K forma un circuito de reset, que tiene la capacidad de volver el circuito a su estado inicial en cualquier momento.

La figura 1 muestra el circuito que hemos descrito. Solo debemos hacer una aclaración importante: en el no figura la conexión del PIN 9 (RESET) del 74HC164N a +5V, aunque si está contemplado en el diseño del PCB. Ese pin DEBE estar a +5V para que el circuito funcione.

En aras de mantener la complejidad de este proyecto lo mas baja posible, no hemos incluido la posibilidad de la programación ICSP.

Lista de componentes

La lista de componentes es más bien pequeña:

```
* 1 microcontrolador PIC16F628A,
    con su zócalo.
* 1 memoria EEPROM 24C256,
    con su zócalo.
* 1 circuito integrado 74HC164N,
    con su zócalo.
* 1 regulador de voltaje LM7805
* 4 condensadores cerámicos de 0.1 uF.
* 2 condensadores cerámicos de 22 pF.
* 1 cristal de 4 MHz.
    (o de 20MHz si el lector lo prefiere)
* 1 condensador electrolítico
    de 220uF/16V.
* 1 diodo 1N4007.
* 8 transistores 2N3906.
* 8 resistores de 100 ohms.
* 1 resistor de 10K.
* 8 resistores de 1.5K.
* 1 bornera de dos tornillos.
* 64 diodos LED de 5mm, color rojo.
```

.Circuito impreso

El circuito impreso necesario para montar este proyecto es relativamente pequeño, y mide unos 80x95 milímetros, con una sola cara. Esto ha obligado a realizar una buena cantidad de puentes en el lado trasero de la placa, pero es mucho mas sencillo construir en casa una PCB de simple cara. Si no sabes como hacerlo, puedes aprender con el tutorial publicado en el numero 1 de la revista uControl. La figura 2 muestra el diseño del PCB, cuyo archivo en formato PDF puedes descargar de nuestra Web.

.Montaje

El montaje no requiere de ninguna técnica en especial. Una vez que tengamos el PCB listo y agujereado, procedemos a soldar los componentes. Podemos comenzar por los resistores y los LEDs. Al hacerlo, hay que tener en cuenta que los LEDs deben tener la muesca que indica el cátodo hacia el lado de los circuitos integrados. Si no lo hacemos así, el proyecto no funcionará.

Más tarde soldaremos los zócalos, el pulsador de reset, los condensadores, el diodo 1N4007 (cuidando su orientación) y el LM7805. Por ultimo, soldaremos el cristal y pasaremos al otro lado del PCB.

Aquí es donde este montaje puede diferir un poco de otros que hayas realizado. Sin embargo, tampoco es tan complicado lo que resta por hacer. Lo primero es soldar las los dos resistores pull-up que requiere el bus I2C para funcionar. La figura 3 muestra donde van soldados.

Una vez hecho esto, debemos realizar una serie de puentes para unir los ánodos de cada fila de LEDs al colector del transistor correspondiente. Debería quedar tal como se puede ver en la figura 4.

.El software

Como decíamos, esta es una excelente herramienta para “jugar” un poco con la programación, y aprender un poco más sobre el lenguaje que utilicemos. En nuestra Web hay algunos ejemplos, y pronto subiremos más. De hecho, nos gustaría que nuestros lectores nos envíen el código que hayan escrito para también incluirlo en la página correspondiente.

Solo publicaremos un ejemplo en CCS, que se en-

carga de mostrar encendidas las filas desde la 0 a la 7, y luego enciende rectángulos de LEDs cada vez más grandes, comenzando desde la esquina superior derecha.

Como el dato se ingresa dentro del registro de desplazamiento cuando el pulso de CLOCK pasa de estado bajo a alto, hemos escrito dos funciones que se encargan de enviar un cero o un uno al registro. La siguiente envía el “0”:

```
//-------------------------------------------
//---Envía un 0 al registro de desplazamiento:
//-------------------------------------------
void fRDD_send_data0(void){
  output_low(RDD_DATA);
  delay_us(2);
  output_low(RDD_CLOCK);
  delay_us(2);
  output_high(RDD_CLOCK);
}
```

y esta otra envía un “1”:

```
//-------------------------------------------
//---Envia un 1 al registro de desplazamiento:
//-------------------------------------------
void fRDD_send_data1(void){
  output_high(RDD_DATA);
  delay_us(2);
  output_low(RDD_CLOCK);
  delay_us(2);
  output_high(RDD_CLOCK);
}
```

En ambas funciones pueden eliminarse los retardos de 2 us.

Este es el código completo del ejemplo:

```
//Device/Fuses/Etc.----------------------------
#INCLUDE <16F628A.H>
#INCLUDE <MATH.H>
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES XT
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD                    //No EE protection
#use delay(clock=4000000)
//Defines--------------------------------------
#BYTE PORTA = 0x05
#BYTE PORTB = 0x06
#BYTE PORTA_TRIS = 0x85
#BYTE PORTB_TRIS = 0x86
#DEFINE RDD_DATA  PIN_A1
#DEFINE RDD_CLOCK PIN_A0
#DEFINE EEPROM_SCL PIN_A2
#DEFINE EEPROM_SDA PIN_A3
//
void fConfigurar_puertos(void);
void fRDD_send_data0(void);
void fRDD_send_data1(void);
//-------------------------------------------
//Main-----------------------------------------
```

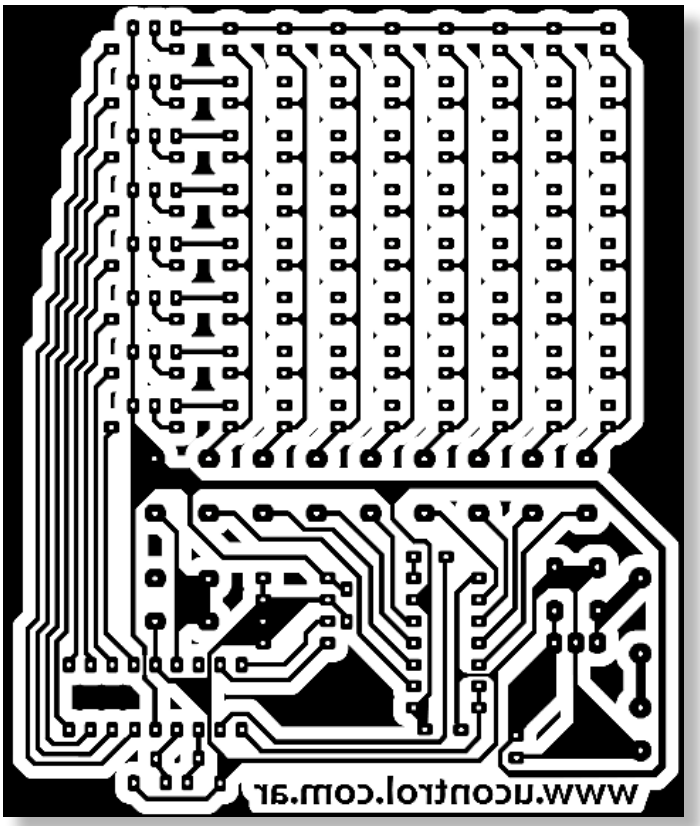


Figura2: El PCB propuesto de una sola cara

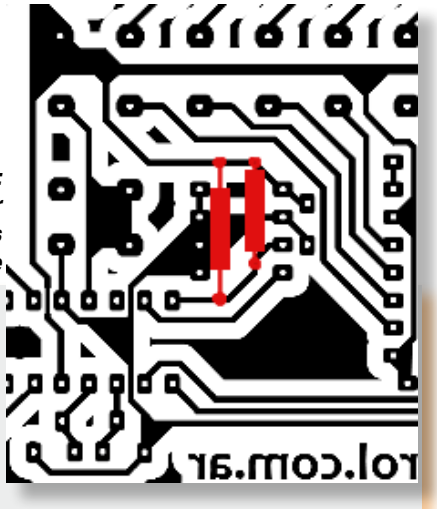


Figura3: Debemos soldar dos resistores del lado del cobre

Un diodo se encarga de proteger el circuito por si involuntariamente conectamos la alimentación con la polaridad invertida.

```
//---------------------------------------------
void main(){
  int8 i,j;
  fConfigurar_puertos();
  output_low(RDD_CLOCK); //RELOJ = Bajo
  output_low(RDD_DATA);
//
//------Dibujo las filas una a una -----
//
  while (TRUE){
     PORTB = 0b11111110; //Filas 1 encendida, las demas apagadas.
     for (i=0;i<8;i++) {
        for (j=0;j<8;j++) {
            fRDD_send_data0();
           }
        delay_ms(200);
        PORTB = (PORTB <<1) +1;
     }
//
//------Dibujo rectangulos... -----
//
     PORTB = 0b11111111; //Filas  apagadas.
     for (i=0;i<8;i++) {fRDD_send_data1();}
     for (i=0;i<8;i++) {
        fRDD_send_data0();
        PORTB = PORTB /2 ;
            delay_ms(200);
         }
  }//Fin while
}//Fin main
//---------------------------------------------
//- FUNCIONES-
//---------------------------------------------
void fConfigurar_puertos(void){
  PORTA_TRIS = 0b11000000; //1=ENTRADA, 0=SALIDA
  PORTB_TRIS = 0b00000000; //1=ENTRADA, 0=SALIDA
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
}
//---------------------------------------------
//---------------------------------------------
void fRDD_send_data0(void){
  output_low(RDD_DATA);
  delay_us(2);
  output_low(RDD_CLOCK);
  delay_us(2);
  output_high(RDD_CLOCK);
}
//---------------------------------------------
//---Envia un 1 al registro de desplazamiento:
//---------------------------------------------
void fRDD_send_data1(void){
  output_high(RDD_DATA);
  delay_us(2);
  output_low(RDD_CLOCK);
  delay_us(2);
  output_high(RDD_CLOCK);
}
```

Un pulsador, junto a una resistencia de 10K forma un circuito de RESET.

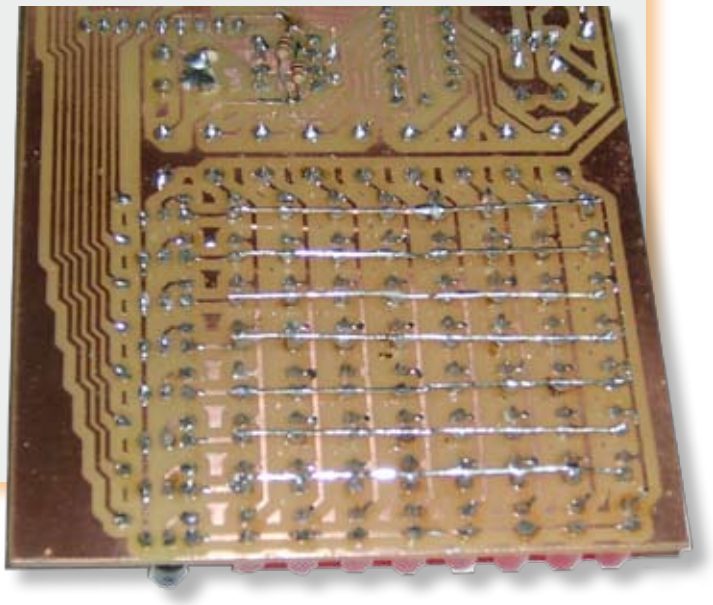
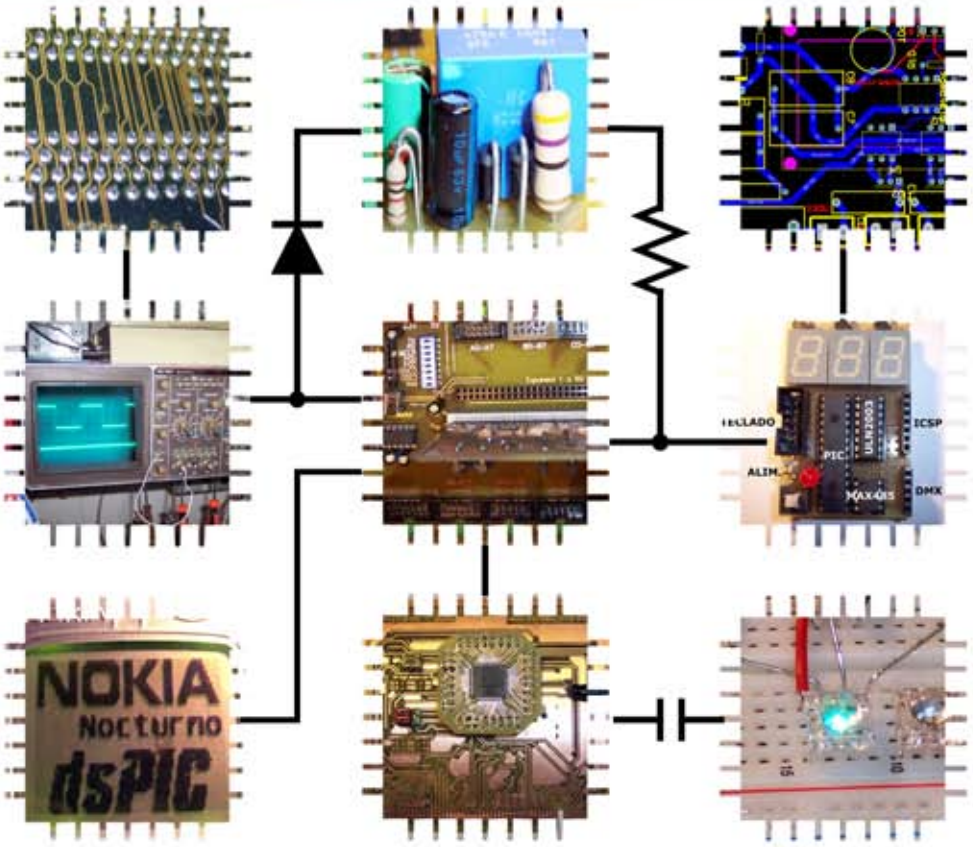


Figura4: Forma de soldar los puentes necesarios

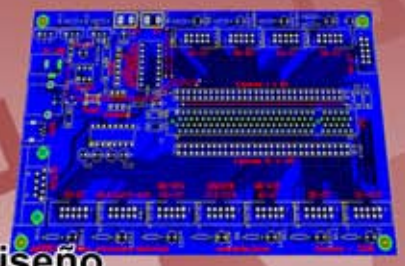
Los lectores que lo deseen pueden ver videos de este programa funcionando en la Web de uControl.

Todo para ti, aficionado a la electrónica y microcontroladores PIC

- Proyectos
- Artículos
- Asistentes
- Tutoriales
- Técnicas
- Trucos



Y si necesita nuestros Servicios Profesionales



Diseño

Programación

Prototipos

Fabricación

MicroPIC

www.micropic.es

Tu sitio favorito sobre electrónica y microcontroladores PIC

//Mario Sacco//
service.servisystem@gmail.com

memorias I2C con Protón Lite

Poco a poco fueron ganando espacio hasta transformarse en uno de los medios de almacenamiento de información más populares por su practicidad y sencillez. Y en BASIC, utilizar mmemorias EEPROM I2C es más fácil aún.

Estos diminutos circuitos integrados, poseen la capacidad de almacenar y recuperar datos de forma organizada, o leer los que han sido previamente grabados en ellos y poseen particularidades que los hacen sobresalir y destacar dentro de su género.

Algunas de las características dignas de mencionar pueden ser las que a continuación enumeramos:

•Pueden ser borradas de forma eléctrica. De ahí su nombre: EPROM es el acrónimo de Electrically Erasable Programable Read Only Memory

•Están garantizadas para 1 millón (a veces más) de ciclos de escritura/lectura.

•Pueden llegar a retener la información sin ser alimentadas, durante cientos de años.

•Se organizan por páginas para facilitar su direccionamiento y almacenamiento de información.

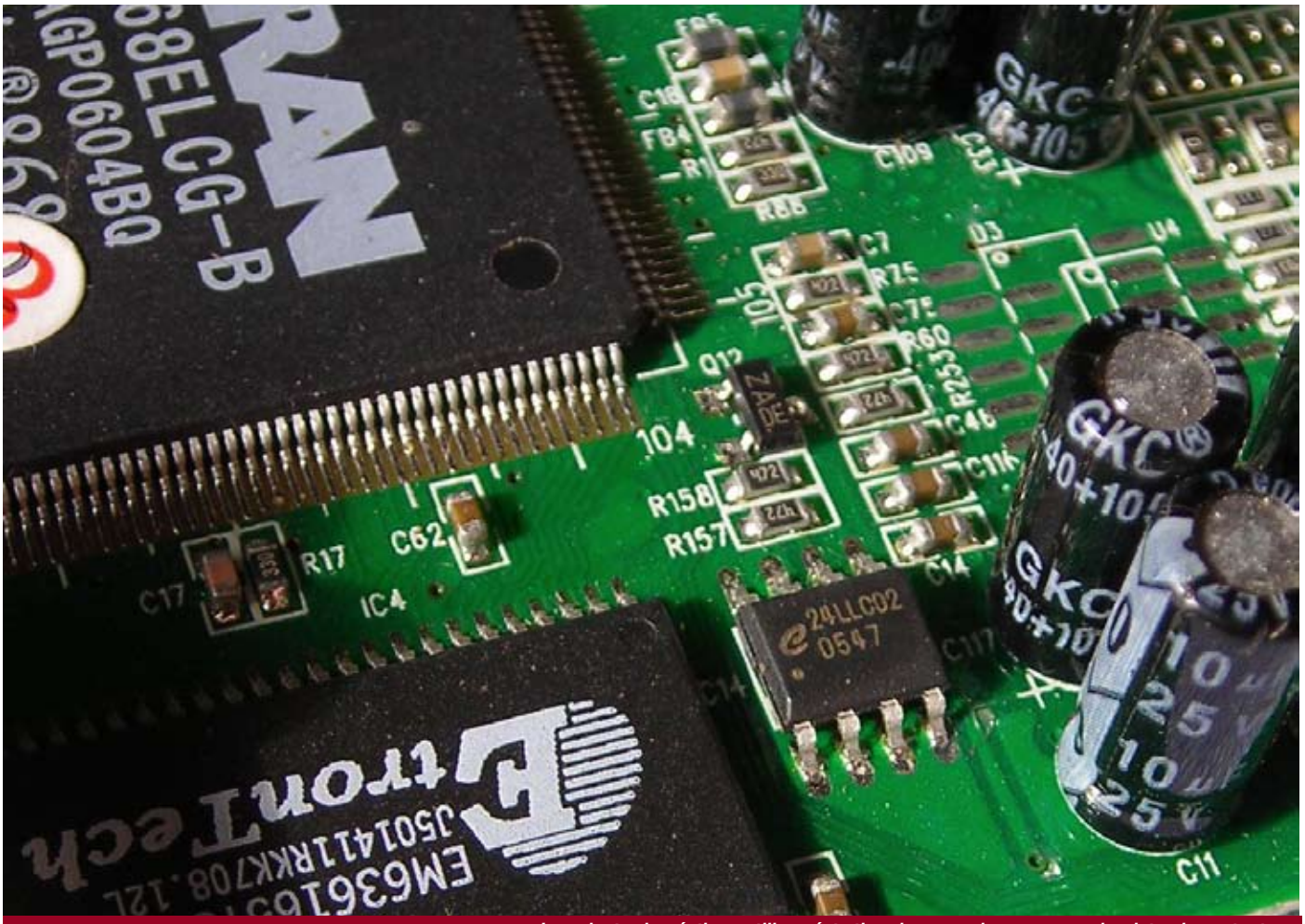
•Utilizan para su funcionamiento una tensión única de 5 Volts.

•Son compatibles con el protocolo serial I2C (Marca Registrada de Philips Corporation)

•Tienen un costo bajísimo.

.Repasemos el Concepto I2C

El Bus I2C (Inter-Integrated Circuit), es un sistema de comunicación que utiliza solo dos cables, con una



Los electrodomésticos utilizan éste tipo de memorias para guardar datos importantes.

velocidad de transmisión de datos de media a baja (400 Khz. a 100 Khz.), que cuál fue desarrollado por Philips Semiconductors a comienzos de la década del '80.

Originalmente fue creado para reducir los costos de los equipos electrónicos, y tuvo entre sus primeras aplicaciones los controles de contraste, brillo y volumen en aparatos de televisión. Actualmente encontramos conexiones por bus I2C en una gran variedad de computadoras, equipos industriales, de entretenimiento, medicina, en sistemas militares y un ilimitado abanico de aplicaciones e importantes usos potenciales.

Antes de la aparición del I2C, las transferencias de datos desde las memorias a los microprocesadores (o microcontroladores), eran realizadas con buses que trabajaban en forma paralela, requiriendo de ésta forma encapsulados con una importante cantidades de pines (24, 28, ó más pines).

El gran número de los pines era indispensable para el direccionamiento de la memoria, selección, control y transferencia de datos. Esta última solamente requería de 8 pines más otros ocho para el direccionamiento, por mencionar un ejemplo.

En cambio, I2C permite la comunicación "chip-to-chip" usando solo dos cables en una conexión serial, permitiendo a los ICs, comunicarse con muy pocas vías. Estos dos "cables" son llamados **Clock (SCL)** y **Data (SDA)** y son los encargados del direccionamiento, selección, control y transferencia de datos, de a un **BIT** por vez.

SDA está encargado del intercambio de datos, mientras que SCL se encarga de sincronizar al transmisor y al receptor durante la transferencia de los mismos desde un circuito integrado al otro.

Dentro del sistema de comunicación I2C, los dispositivos están identificados como **Maestro (Master)** y **Esclavo (Slave)**, por lo que al dispositivo que inicia el contacto y "abre" el bus, se lo denomina **Master**, mientras que al que contesta el llamado se lo denomina **Slave.**

Los dispositivos conectados al bus, pueden ser Master solamente, Slave solamente, ó intercalar las funciones de Master y Slave de acuerdo como el sistema lo requiera, tal como es el caso que veremos de las memorias EEPROM I2C.

Este sistema puede interconectar a muchos IC sobre el bus (hasta 255 dispositivos), todos conectados a los mismos dos cables **SDA** y **SCL**. Cada dispositivo esclavo posee una única dirección y cuando el **Master** transmita el llamado, todos los ICs conectados al bus lo escucharán, pero solo le contestará aquel que posea la dirección que el transmisor incluyó en su llamada y será con éste único **Slave**, con quien iniciará la transferencia de datos hasta que decida "cerrarla".

.Comenzando a comunicarnos

La condición de **START** o Inicio, ocurre únicamente en la transición de un estado **ALTO** a un estado **BAJO** en la línea **SDA**, mientras la línea **SCL** se encuentre en un nivel **ALTO**.

En cambio, la transición de un estado BAJO a un estado **ALTO** en la línea **SDA**, mientras la línea **SCL** se encuentre en un nivel **ALTO**, indicará una condición de **STOP** ó Parada.

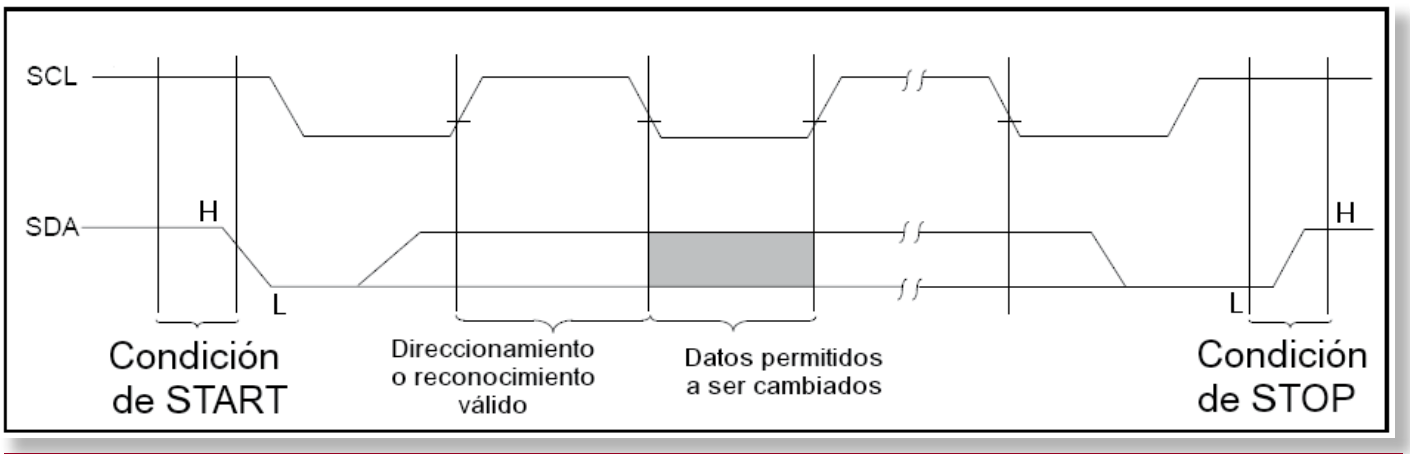
Las condiciones de **Start** y **Stop**, son siempre generadas por el dispositivo que se asigne la condición de **Master** dentro del bus.

El bus se considerará ocupado después de la condición de **Start** y se considerará nuevamente libre, cierto tiempo después de la transmisión de la condición de **Stop**. Tiempo, que será determinado por el **Master** y vendrá especificado en la hoja de datos del mismo.

Las memorias se componen de páginas de 256 Bytes, a excepción de la 24C00 y 24C01.

.Ahora vamos a programar

En Protón debemos indicarle al programa y/o al microcontrolador cuáles son los pines que cumplirán las funciones de **SDA** y **SCL**, en el caso en que no conectemos el bus a los pines dedicados a tal fin del microcon-



Transición de SDA para generar las condiciones de START y STOP

trolador, o bien cuando el microcontrolador no disponga conexiones por defecto de bus I2C.

Esto se realiza al comienzo del programa, antes de la declaración de variables, y se formaliza mediante un comando **DECLARE**, quedándonos la secuencia de la siguiente forma:

```
DECLARE SDA_PIN PortA.1
DECLARE SCL_PIN PortA.0
```

En el ejemplo propuesto, hemos ordenado que SDA sea el pin 1 del puerto A, mientras que SCL será el pin 0 del mismo puerto.

Otra de las cosas que debemos indicarle al bus, es si vamos a trabajar con sistemas de oscilador de más de 8 Mhz. Si utilizamos una frecuencia menor, incluiremos el siguiente comando:

```
DECLARE SLOW_BUS=ON
```

Luego de esto, ya estamos listos para "abrir" el bus, cosa que haremos con el sencillo comando:

```
BSTART
```

Y de ésta forma, el bus ya habrá sido abierto por nuestro microcontrolador, asignándose éste la función de **Master**, y pasando a estar todo listo para la transferencia de datos hacia el **Slave** que éste decida, transmitiendo al bus la dirección pertinente.

Para leer datos, en nuestro caso alojados en una memoria, lo haremos de la siguiente forma:

```
BUSIN Control, Dirección, [Variable]
```

La sintaxis expresada nos indica que el **Master** recibirá **(BUSIN)** un dato, el que colocará dentro de una **Variable**, luego de haberlo sacado de uno de los dispositivos "colgados" del bus I2C, al que habrá seleccionado a través de la palabra d**e Control**, indicándole a éste que va a extraer datos de él, y que dicho dato se encuentra en la **Dirección** apuntada

Si por el contrario lo que desea hacer es escribir un dato en el dispositivo Slave seleccionado, lo hará mediante la siguiente forma:

```
BUSOUT Control, Dirección, [Variable]
```

Con una sintaxis muy similar al caso anterior, el **Master** transmitirá **(BUSOUT)** un dato, el que tomará de una **Variable**, y lo grabará en uno de los dispositivos conectados al bus y seleccionado con la palabra de **Control,** indicándole que grabará en él y que a este proceso lo hará en la **Dirección** definida por el **Master**

Protón y la mayoría de los fabricantes de memorias, recomienda efectuar una rutina de demora ó espera, luego de haber grabado un dato a través de **BUSOUT**, para asegurar la grabación del dato, la que se efectiviza mediante la expresión:

```
DELAYMS 10
```

Indicándonos con esto, que efectuará un Delay (retardo, demora) de 10 milisegundos.

Luego, solo nos queda "cerrar" el bus y lo haremos así:

```
BUSTOP
```

Eso es todo. Ya tenemos la forma de abrir el bus, de leer ó de grabar en un dispositivo **Slave** y de cerrar nuevamente el bus.

Es momento de saber cómo manejar la palabra de **Control** y la **Dirección** de lectura/escritura.

.Organización Interna de las Memorias

No todas las memorias EEPROM I2C se direccionan ni controlan de la misma forma, por lo tanto, haremos un breve resumen de los datos más relevantes a tener en cuenta al momento de emplear los comandos **BUSIN** y **BUSOUT**.

Palabra o Byte de Control

La palabra o Byte de Control, nos indicará la dirección que posee la memoria dentro del bus y si vamos a leer o a escribir en ella

24C00

1	0	1	0	X	X	X	R/W

R/W significa Read ó Write, que en español significa Leer ó Escribir, respectivamente, según nos dispongamos a leer ó a escribir en la memoria.

Si vamos a leer la memoria (Read), este bit adoptará el valor 1, en cambio al grabar un dato en la misma (Write) el valor será 0

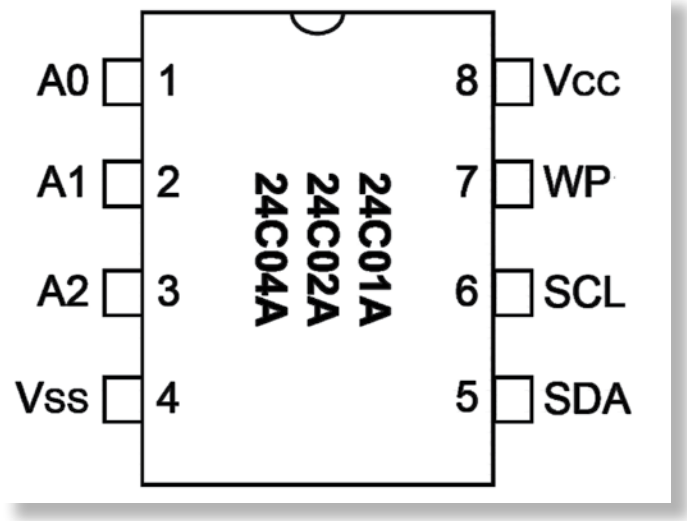
X, significa que no tiene relevancia el valor que adopte, por lo que podemos adoptar para éste lugar, un cero.

La memoria **24C00** posee una capacidad de **128** Bits, organizada en 16 Bytes de 8 Bits

24C01/24C02/24C04

1	0	1	0	A2	A1	A0	R/W

Para éste grupo de memorias, tenemos la misma dirección en los cuatro bits iniciales (1010) que en el caso anterior, lo mismo para el bit final de R/W, pero encontramos el cambio en las posiciones A0, A1 y A2 que coinciden con los pines uno, dos y tres respectivamente, según las hojas de datos de las memorias.



Los pines A2, A1 y A0 se utilizarán para seleccionar el chip al que direccionaremos la transferencia de datos

Con éstos tres bits, podemos direccionar hasta ocho memorias conectadas al mismo bus, trabajo que puede realizar el microcontrolador.

La única que se diferencia de éste grupo es la **24C04** que debe tener siempre conectado **A0** a **GND** ó a **VCC**, pudiendo por lo tanto, direccionarse sólo cuatro unidades de la misma, a través de **A1** y **A2**.

A0, en éste caso, servirá para direccionar el "puntero" de escritura ó lectura, hacia la primer página de 256 Bytes ó hacia la segunda.

Aquí empezaremos a ver ya, cómo la estructura interna de estas memorias está organizada en "páginas".

Por último destacamos que en éste grupo de memorias, encontramos las siguientes características de almacenamiento:

24C01 = 128 Bytes	24C02 = 256 Bytes	24C04=2 X 256 Bytes= 512 Bytes

Vemos entonces, que las 01 y 02 poseen una sola página, por lo que la palabra de control para ellas será igual a **1010000X**, siendo X el bit que defina la lectura o escritura en la memoria, mientras que la 24C04 tendrá **DOS** palabras de control, según la página donde decidamos trabajar.

Esto es, **1010000X** para la primer página y **1010001X** para la segunda.

24C08/24C16

1	0	1	0	B2	B1	B0	R/W

Repetimos los primeros cuatro bits (1010) y R/W, y nos encontramos con B2, B1 y B0 en estos modelos.

En el caso de estas dos memorias, no podremos colocar en el bus más de una de ellas, a diferencia de las anteriores vistas hasta aquí. A pesar de poseer la misma disposición de pines que los modelos anteriores A2, A1 y A0 no poseen conexión interna, colocándose generalmente estos a GND.

B2, B1 y B0 sirven para identificar la página (ó bloque) dentro de la memoria, o sea que para la 24C08 que posee una disposición de cuatro páginas de 256 Bytes (4 X 256 X 8 Bits = 1 KByte = 8 KBit); mientras que la 24C16, dispondrá de 8 páginas de 256 Bytes.

Dicho esto y deduciendo desde el ejemplo anterior, tendremos 4 palabras de control para la 24C08 y 8 Palabras de control para la 24C16; una por cada página de 256 Bytes.

Dispositivo	*Palabra*	*Página*
	1010 B2 B1 B0 R/W	
Ambas Memorias	1010000X	1
	1010001X	2
	1010010X	3
	1010011X	4
24C16	1010100X	5
	1010101X	6
	1010110X	7
	1010111X	8

.La palabra Dirección

Habiendo llegado hasta aquí, nos queda resolver solamente cómo estará compuesta ésta palabra.

Tal cómo hemos visto aquí, las memorias se componen de páginas de 256 Bytes, a excepción de la 24C00 y 24C01.

En el caso de la 24C00 teníamos una página de 16 Bytes, por lo que tendremos sólo 16 "espacios" para ubicar Bytes de información.

Esto puede ser expresado en forma Binaria, Decimal o Hexadecimal.

Protón Lite acepta cualquiera de las tres notaciones para las Palabras de Control y de Dirección, por lo que podemos escribir para facilitar el trabajo, un comando de la siguiente forma:

```
BUSIN 161, 8, [Dato]
```

Tenemos la palabra de control 161, lo que sería 10100001 en binario, lo que nos indica la primera página (página cero) de cualquier memoria de las vistas y que la misma será leída.

Luego viene el valor 8, que sería 00000100 en binario, lo que nos indica que se leerá el octavo de todos los casilleros que tenga ésta página.

Por último, el dato extraído será volcado en una variable de tamaño **BYTE** a la que hemos denominado "Dato"

Escribir en ésta misma dirección sería:

```
BUSOUT 160, 8, [Otro_Dato]
```

El último bit de la palabra de control a pasado a ser cero, y el valor a grabar será el que exista en ese momento dentro de la variable “Otro_Dato”, también de tamaño **BYTE** y declarada al inicio del listado del programa.

Un listado elemental de lectura de éste tipo de memorias sería el siguiente:

```
DEVICE 16F84A                       ‘DECLARO EL PIC A USAR
DECLARE XTAL = 4                    ‘DECLARO EL CRISTAL A USAR

SYMBOL LEER %10100001               ‘LA PALABRA DE CONTROL LA DENOMINO ‘LEER

DECLARE SDA_PIN PORTA.1             ‘DECLARO LOS CUALES PINES DEL PIC ‘VAN A SER SDA Y SCL CON UN
BUS DE ‘MENOS DE 8 MHZ
DECLARE SCL_PIN PORTA.0
DECLARE SLOW_BUS=ON

DIM A AS BYTE                       ‘DECLARO  LA VARIABLE “A”
DIM DATO AS BYTE                    ‘DECLARO  LA VARIABLE “DATO”

CLS                                 ‘INICIAMOS CON PANTALLA LIMPIA

BSTART                              ‘ABRIMOS EL BUS

FOR A = 0 TO 255                    ‘CONTAMOS DE 0 A 256
BUSIN LEER, A, [DATO]               ‘LEEMOS EN UNA PÁGINA CERO
PRINT AT 1, 1, DEC DATO             ‘MOSTRAMOS EL DATO LEÍDO EN UN LCD
DELAYMS 500                         ‘LO DEJAMOS QUE SE VEA MEDIO ‘SEGUNDO
NEXT                                ‘AVANZA UN PASO LA CUENTA

CLS                                 ‘AL TERMINAR LA CUENTA DE LAS 256 POSICIONES, LIMPIAMOS LA PANTALLA

BSTOP                               ‘CERRAMOS Y LIBERAMOS EL BUS

PRINT AT 1, 1, “FIN DE LECTURA”     ‘INDICAMOS QUE LA LECTURA TERMINÓ
DELAYMS 2000                        ‘MOSTRAMOS EL MENSAJE 2 SEGUNDOS

CLS                                 ‘LIMPIAMOS LA PANTALLA
END                                 ‘FIN DEL PROGRAMA
```

Con este sencillo programa, habremos leído la primera página de una memoria EEPROM I2C y habremos colocado en un display LCD los valores leídos durante medio segundo, para pasar al próximo valor, hasta leerlos a todos. Para otras páginas, variaremos la palabra de control a nuestras necesidades.

24C08 y 24C16 son memorias muy utilizadas en los equipos de TV y DVD para almacenar información

//Raul Sandoval Amesty//
rauljsand@hotmail.com

módulo ICSP para PIC16F877 y Protoboard

A la hora de programar un microcontrolador, y sobretodo cuando se necesitan hacer pruebas por medio de ensayo y error, la programación ICSP (In-Circuit Serial Programming, o Programación Serial en el Circuito) es la opción mas adecuada y eficiente para hacerlo. En este articulo, veremos como crear un modulo ICSP específico para el PIC16F877, que calza directamente en los pines correctos del integrado en el protoboard.

La facilidad que nos provee la programación ICSP de los microcontroladores PIC es una herramienta que se debe conocer para mejorar la efectividad y rapidez a la hora de hacer pruebas con un circuito con estos integrados. Se puede aprovechar la programación ICSP para que, después de armada alguna placa, modificar, borrar y mejorar el código guardado en el PIC.

En este caso emplearemos este sistema para crear un modulo de programación orientado al uso en un circuito con microcontrolador armado sobre un protoboard, olvidándonos así de tener que remover el PIC del mismo cada vez que necesitamos hacerle cambios al programa, evitando de esta forma los posibles daños que podríamos ocasionarle al circuito integrado.

Pero antes de utilizar esta herramienta, presente en prácticamente la totalidad de los microcontroladores PIC, debemos saber como funciona.

En principio, existen dos maneras de hacer que el microcontrolador entre en estado de programación. La primera de ellas es utilizando la HVP (High Voltaje Programming, o Programación por Alto Voltaje) que consta de aplicar un voltaje VIHH, especificado en la hoja de datos, al pin Vpp/MCLR. La segunda es mediante la LVP (Low Voltaje Programming o Programación por Bajo Voltaje), que se logra por medio de la activación del bit LVP de la palabra de configuración del microcontrolador. Cabe destacar que en este ultimo caso, no se necesita llevar el voltaje aplicado al pin Vpp/MCLR hasta VIHH.

Con la programación ICSP grabamos el PIC directamente en el protoboard sin necesidad de quitarlo.

Luego de llevar al PIC al estado de programación, se comienza la transmisión serial por medio de los pines PGC (Señal de reloj para la programación serial) y PGD (Señal de datos para la programación serial).

Aparte de las patillas ya mencionadas, se necesita energizar el microcontrolador por medio de los pines VDD y VSS. Esta tensión puede provenir directamente del programador o de una fuente de alimentación externa.

ICSP para el PIC16F877 en acción

Se debe saber que los pasos descritos arriba para la programación ICSP son seguidos automáticamente por el software y el hardware de todos los grabadores de PICs. Nosotros solo nos debemos preocupar por la conexión del programador al PIC y por avisar al grabador y al software que se utilizara programación ICSP. Este paso varía de acuerdo a cada hardware y software de programación.

Como se puede observar en el esquema, se trata de un circuito extremadamente sencillo cuyos únicos componentes son unos pines SIL para conectarlo directamente en el protoboard. Estos pines están ubicados unos del lado superior de la placa (para conectar el modulo al programador) y otros del lado inferior de la placa (para conectar el modulo al protoboard).

En estos últimos pines se debe tener una consideración especial ya que debido a la altura del circuito integrado, los pines SIL rectos no son los suficientemente largos como para entrar en el protobard. Para solucionar este problema se emplearan unos pines SIL en ángulo

recto, que posteriormente se enderezaran hasta quedar en forma recta (como se muestra en la foto). También se debe correr el plástico que mantiene unidos entre si los postes de bronce individuales hacia el extremo de los mismos, para aprovechar su máxima longitud.

El PCB recomendado es el que se ve en la foto. Observe como los pines que provienen del programador se conectan directamente con los pines que corresponden en el PIC, evitándonos así hacer el cableado en el protoboard cada vez que queramos utilizar ICSP.

Debido a que se trata de una placa muy sencilla, no amerita el empleo de un PCB doble cara, por esto a la hora de soldar los pines SIL en ángulo que ya se doblaron en forma recta, se debe atender a que estos deben ser soldados como se ve en la figura.

Al finalizar la placa solo nos queda probarla y utilizarla en nuestros proyectos con el PIC16F877, olvidándonos de los dolores de cabeza que causan los pines doblados o incluso rotos.

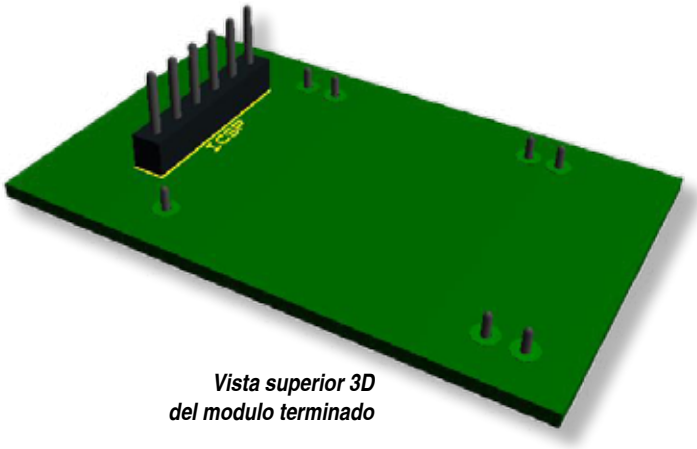
Bibliografia:

- Hoja de datos del PIC16F877, en http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf

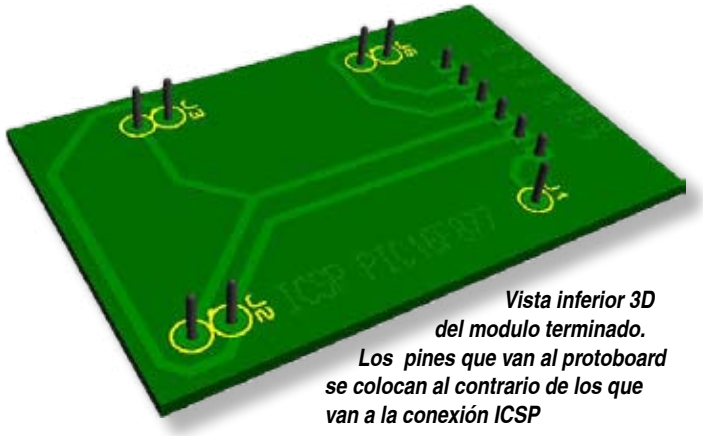
- PIC Programming using ICSP and ICPROG http://www.best-microcontroller-projects.com/pic-programming.html

.Pines empleados para el ICSP en un PIC 16F877

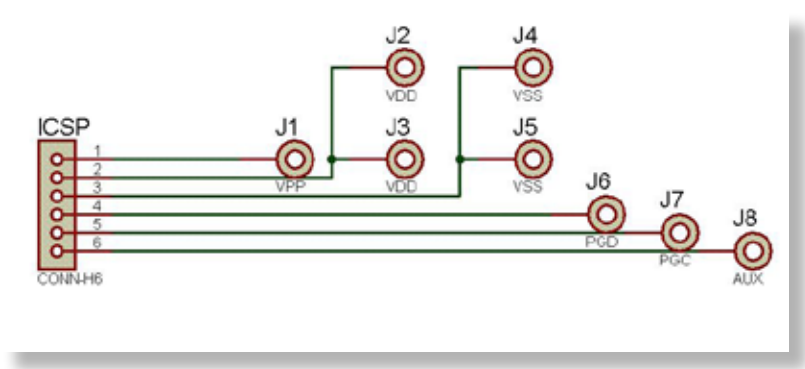
Nombre	Pin #	Descripción
VPP/MCLR	1	Voltaje de programación. Voltaje aplicado al PIC para que entre en modo de programación.
VDD	11 y 32	Alimentación positiva (Vcc). Puede ser de una fuente externa, no debe venir necesariamente del programador.
VSS	12 y 31	Alimentación negativo (Gnd). Puede ser de una fuente externa, no debe venir necesariamente del programador.
PGC	39	Línea de reloj de la interfaz serial.
PGD	40	Línea de datos de la interfaz serial.



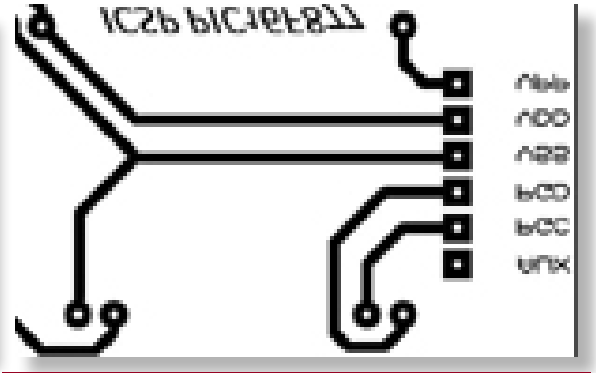
Vista superior 3D del modulo terminado



Vista inferior 3D del modulo terminado. Los pines que van al protoboard se colocan al contrario de los que van a la conexión ICSP



Esquema del Modulo ICSP para el PIC16F877



PCB del Modulo ICSP para el PIC16F877

//Mariano Madrid Ortiz//
marianommo@gmail.com

cálculo de disipadores para semiconductores de potencia

Siempre llega el momento en la vida del aficionado en que algún componente se calienta más de lo normal y acaba fundiéndose por exceso de temperatura o, en el peor de los casos, abrasándote la yema del dedo mientras compruebas porqué no funciona tu circuito. Además de la citada utilidad como protección, los disipadores también son necesarios por cuestiones de eficiencia, un integrado de potencia siempre funciona mejor a temperaturas razonables que en su límite máximo soportable. Algunos circuitos integrados pueden incluso, entregar una mayor potencia de salida al soportar una mayor temperatura, producida por una corriente más grande.

La solución a los problemas de temperatura y a la mejora de eficiencia de circuitos integrados de potencia, pasa por incorporar un disipador. Si nuestro circuito no es de grandes pretensiones en cuanto a potencia se refiere o no tenemos restricciones en cuanto a espacio, nos bastara con una refrigeración pasiva sin necesidad de ventilador.

Para calcular el disipador necesario vamos a comentar unas pautas y, para su comprensión final, la mejor manera es explicarlo mediante un ejemplo.

Como suele ocurrir en muchos casos, cuando nos sacas a un electrónico de nuestros niveles lógicos, nuestras tensiones y nuestra teoría de circuitos, podemos acabar un poco desorientados. Sin embargo, nos distinguimos por saber solucionar estas dificultades llevándolas a nuestro terreno. “Si no puedes a tu enemigo, únete a él”. No iba a ser menos el caso de las fuentes y las resistencias térmicas, que mejor manera de abordarlas que buscando una analogía con un circuito eléctrico. (Ver recuadro 1)

Sin embargo este esquema puede resultar demasiado detallado para la gran mayoría de aplicaciones. Con una simplificación podemos acometer con éxito la refrigeración de la mayoría de integrados de potencia. Solo en los casos más extremos de diseño, en los que una diferencia de unos pocos grados pueda dar lugar al desastre, utilizaremos el esquema completo. Para el resto de aplicaciones simplificando resistencias con órdenes de magnitud insignificantes, podemos aproximar cualquier integrado con disipador a este otro circuito (Ver recuadro 2).

Con este esquema, teniendo en cuenta la temperatura ambiente, potencia disipada por el integrado en los peores casos y además, con los datos de la hoja de características podemos calcular Rs, obteniendo el valor de la resistencia térmica que ha de tener el disipador.

.Ejemplo

Vamos a calcular un disipador para un transistor que soporte 30A, puede ser un BUZ11 por ejemplo.

Lo primero es disponer de su hoja de características.

En ella vemos en el apartado Thermal, que su Rthj-case es de 1.67 ºC/W (En el esquema es la Rjc)

También nos dice que la Tj máxima que soporta es de 175ºC, nosotros hemos de calcular el disipador para que nunca alcance esa temperatura.

En nuestro circuito real, en el que está trabajando el BUZ11, hemos de calcular cual es la potencia máxima que va a disipar nuestro transistor por las técnicas habituales de cálculo. Imaginemos algo sencillo, el transistor lo hacemos trabajar de forma constante con un punto de trabajo de Id=30A y Vds=1V. Por lo tanto disipa una potencia Pd de 30W.

Ahora vamos con la Rcs, es decir, con la resistencia térmica que hay entre el encapsulado del transistor y el disipador. Es recomendable utilizar pasta térmica como la de los procesadores del PC. Por ejemplo supongamos de una marca comercial bastante conocida, cuya resistencia térmica es, según el fabricante, de <0.0045°C-in2/Watt (0.001 inch layer). Como ves depende de la superficie de contacto, “in2” son pulgadas cuadradas. Para facilitar los cálculos, suponemos que el encapsulado está en contacto con el disipador

en una superficie de 1 pulgada cuadrada, con lo que la Rcs es de 0.0045°C/W

Ahora hemos de ver dónde va a trabajar ese dispositivo, no es lo mismo que este dentro de un horno, que dentro de una nevera. Pongamos, por ejemplo, que se encuentra en una habitación donde la temperatura ambiente nuca va a sobrepasar los 40ºC. Entonces Ta es de 40ºC.

Con todos esos datos ya podemos calcular Rs, la resistencia térmica de disipador necesaria.

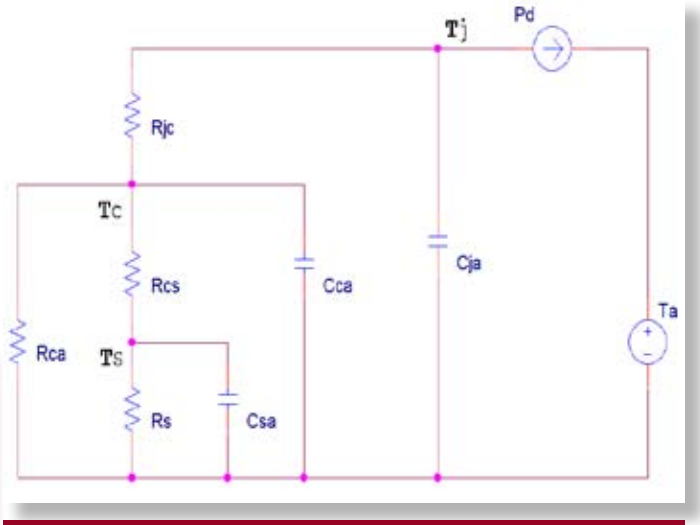
- *Tj<Tjmax - Condición para que no se funda*
- *Tj = Pd*(Rjc + Rcs + Rs) + Ta - Calculo de Tj según el esquema*
- *Tjmax > Pd*(Rjc + Rcs + Rs) + Ta - Sustituimos*
- *(Tjmax-Ta)/Pd - (Rjc + Rcs) > Rs - Y despejamos Rs*

En nuestro caso *Rs < 2.8455ºC/W*

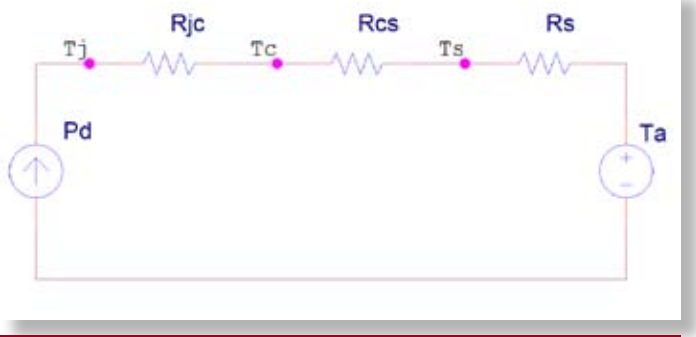
Con eso ya sabemos cuál es la resistencia térmica máxima que ha de tener nuestro disipador. Es cuestión de buscar en la tienda, uno que se adapte a lo que hemos calculado.

Este es un modelo simplificado, que es válido en la gran mayoría de casos, lo interesante, como siempre en electrónica, es sobredimensionar los datos y así funcionará perfectamente aunque consuma algo más o un día en la habitación haya 50ºC en lugar de 40.

Bibliografia:
Rashid, M.H., Electronica de Potencia, Prentice-Hall, 1995.



Circuito completo para el cálculo de disipadores



Circuito reducido para el cálculo de disipadores

RECUADRO 1:

Modelo de cálculo completo

Aquí podemos ver el circuito completo que nos sirve como analogía para el cálculo de disipadores. En él podemos ver:

Rjc:	Resistencia térmica entre unión y encapsulado
Rcs:	Resistencia térmica entre encapsulado y disipador
Rs:	Resistencia térmica del disipador
Cja:	Capacitancia térmica entre unión y ambiente
Cca:	Capacitancia térmica entre encapsulado y ambiente
Csa:	Capacitancia térmica entre disipador y ambiente
Rca:	Resistencia térmica entre encapsulado y ambiente
Tc:	Temperatura del encapsulado
Tj:	Temperatura de la unión
Ts:	Temperatura del disipador
Ta:	Temperatura ambiente
Pd:	Potencia disipada por el componente

RECUADRO 2:

Modelo de cálculo simplificado

Aquí podemos ver el circuito reducido que nos sirve como analogía para el cálculo de disipadores. Recomendamos este, por simplicidad, en la mayoría de los casos. En él podemos ver:

Rjc:	Resistencia térmica entre unión y encapsulado
Rcs:	Resistencia térmica entre encapsulado y disipador
Rs:	Resistencia térmica del disipador
Tc:	Temperatura del encapsulado
Tj:	Temperatura de la unión
Ts:	Temperatura del disipador
Ta:	Temperatura ambiente
Pd :	Potencia disipada por el componente

//Diego Garcia Cuervo//
picmania@garcia-cuervo.com
http://picmania.garcia-cuervo.net/

decodificando el protocolo ABA Track 2

Como complemento ideal a nuestro artículo anterior sobre los Fundamentos de la Transmisión Síncrona qué mejor que un ejemplo real como la vida misma. Me propongo ahora que veamos juntos un protocolo, usado por las antiguas lectoras de tarjetas magnéticas, que lleva decenas de años en uso y aún hoy viene implementado en la gran mayoría de lectores de tarjetas modernas, Chip, Proximidad, Mifare ... etc como emulación de aquellas para mantener cierta compatibilidad entre los distintos dispositivos a los que están conectados.

Este protocolo ABA Track 2 es un invento de la American Banking Association (de ahí lo de ABA) y todos y cada uno de sus detalles aparecen absolutamente descritos hasta su último detalle en la norma ISO-7813.

No es nuestro propósito describir esta **ISO-7813** como si en ello nos fuese la vida, sino mas bien acercarnos a ella desde un punto de vista mucho mas práctico y útil para Picmaníacos que lo único que desean es poder leer e interpretar uno de estos aparatos con protocolo **ABA Track 2.**

Empecemos pues manos a la obra y demos un primer vistazo a lo que se nos viene encima, y que no es más que un ***"cronograma"***, una representación gráfica del estado de un par de líneas en función del tiempo. Podemos verlo en la figura 1.

Supongamos que tomamos las dos líneas de salida de un dispositivo que transmite en **ABA Track II** y somos capaces de monitorizarlas en algún tipo de visualizador de líneas digitales (en mi caso he usado mi proyecto RR Logical Analyzer) No es necesario disponer de uno de estos aparatos, pero para que veáis qué ocurre nos viene pero que muy bien.

Tened en cuenta que lo que describimos en el anterior artículo es únicamente el cómo se transmite la información, y no dijimos ni una sola palabra del qué se está transmitiendo, del significado de lo que se transmite ni de la interpretación que podíamos hacer de lo que recibimos de esa transmisión.

Un protocolo nos obliga a más cosas que al simple método a usar para ser capaces de recibir o emitir una información. Nos dice también qué se va a transmitir y qué significado tiene eso que es transmitido y/o recibido. Es lo que coloquialmente conocemos como *"trama"*. La trama es lo que realmente aplicamos a lo recibido para obtener una interpretación válida de los datos. El protocolo pone de acuerdo al emisor y al receptor en todos y cada uno de los detalles para conseguir una completa, eficaz y cohe-

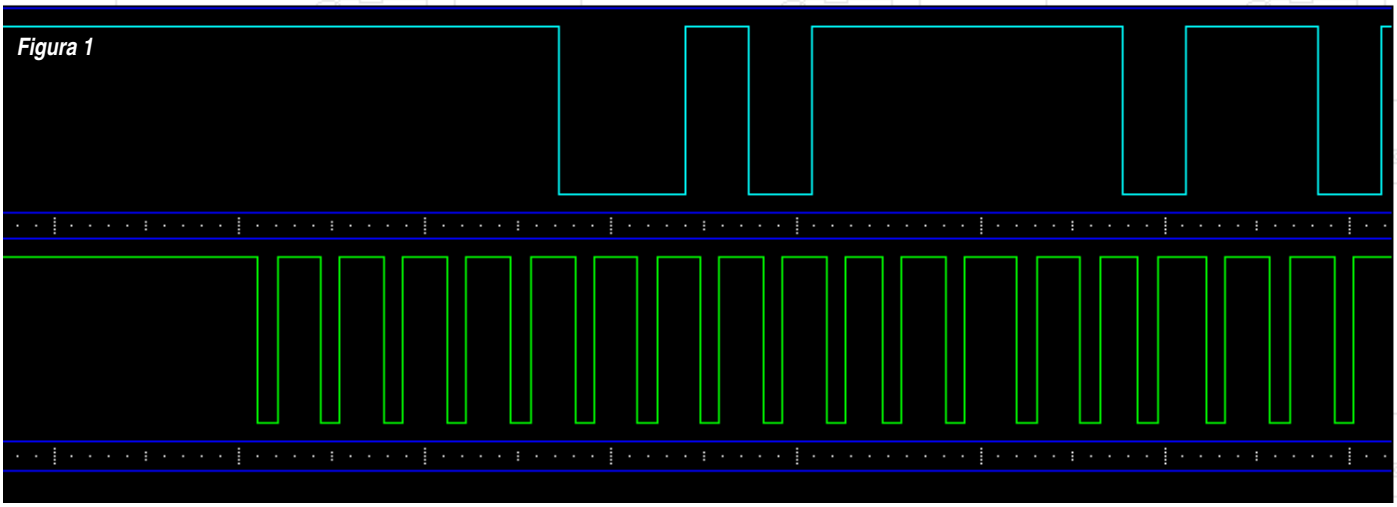
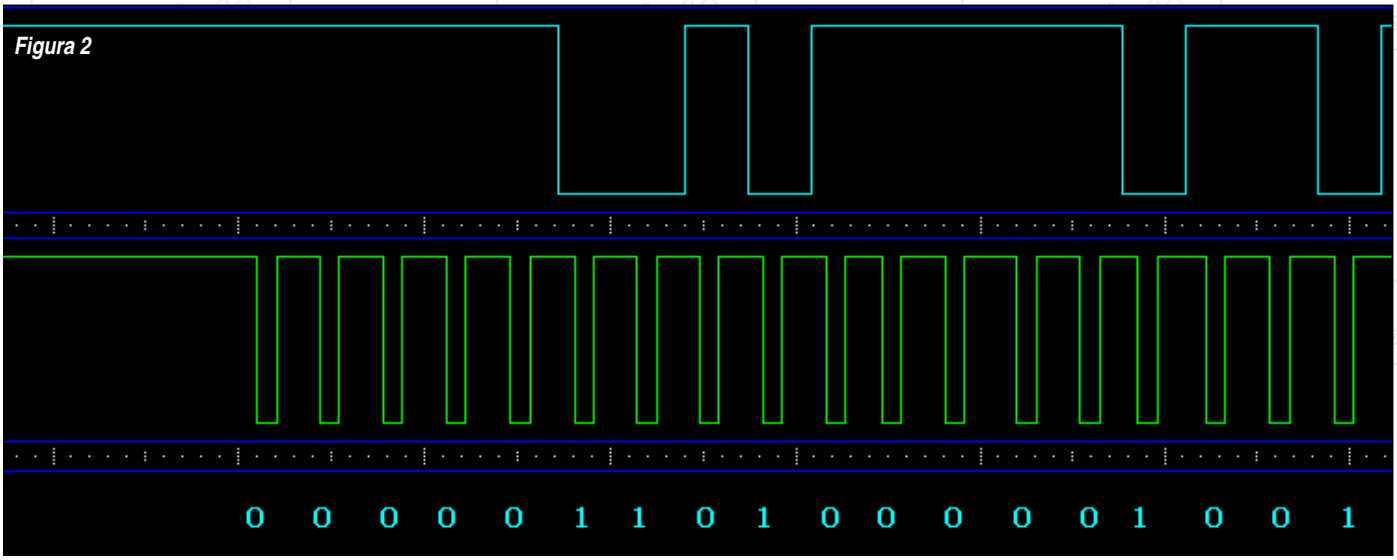


Figura 1



rente transmisión y recepción de los datos.

Con lo aprendido en el artículo sobre la *Transmisión Síncrona* podemos dar un paso más con solo interpretar nuestro cronograma anterior.

En aquél veíamos cómo una línea de reloj **(CLOCK)** nos indicaba cuando debíamos mirar, sensar, catar, leer la otra línea, la de datos **(DATA)**. De manera tal que al recibir un pulso del **CLOCK** si la línea **DATA** estaba en alto, a nivel de **VCC**, lo interpretábamos como que habíamos recibido un bit 0, y si por el contrario la línea **DATA** estaba en bajo, a nivel de **GND**, lo leíamos como un bit **1**.

Con esta simple norma íbamos recibiendo toda una ristra de bits, uno a uno sobre la línea de **DATA**, y cada uno de ellos al golpe de batuta de nuestra línea de **CLOCK,** con lo cuál estamos en disposición de añadir a nuestro cronograma la interpretación en Bits de lo que significa (Figura 2).

Quedémonos por ahora con los quince primeros bits recibidos: **00001101000001**.

¿Significan algo? Es difícil de decidir con la sola vista de esos bits. Podríamos intentar decidir si se ajustan a los patrones de la tabla **ASCII**, pero parece complicado ya que no son divisibles por 8, que son los bits que tiene un byte y que la tabla ASCII nos dice qué significan, o si por el contrario los dividimos en bloques de 4 bits ... o de cinco ... o ... Así que vamos a concluir que: **¡No! No significan nada salvo que digamos mas cosas.**

.ABA Track II

Y entonces viene en nuestro salvamento el Séptimo de Caballería, que a nuestros efectos es nuestro viejo amigo el Protocolo **ABA Track II**, y que dice en primer lugar, como Dios hablándole a Moisés con voz profunda y sentenciosa cual Primer Mandamiento: **Primero recibirás un montón de ceros**, de cinco a quince, pero no más ... ni menos. Ni cuatro, ni tres, ni dos, ni uno, sino cinco o más.

Vemos que en verdad así ocurre en nuestra ristra de bits, primero nos llegan un montón de ceros (originalmente venían quince pero los he dejado en cinco, editando la imagen del cronograma).

Y sigue nuestro **ABA Track 2** con su segundo mandamiento: A continuación cada **Byte de información llegará con cinco bits, los cuatro primeros definen el dato a enviar y el quinto es la paridad**. Que porque me da la gana va a ver **Impar. El Primer bit de los cinco es el Menos Significativo.**

En cada bloque, los 4 primeros bits son "el dato" y el quinto se calcula en función de los anteriores

Ya tenemos un poco mas de información y podemos empezar a intentar descifrar la "trama". A partir del primer 1 que encontremos vamos a dividir nuestro grupo de bits en bloques de 5, de los cuales los 4 primeros son "el dato" y el quinto se calcula en función de los anteriores.

Troceando, tomamos nuestro afilado cuchillo ritual y nos quedan dos pequeños bloques: **11010 y 00001**.... Y ya sabemos que ambos son "datos+paridad_impar" Luego el primero vamos a escribirlo en la forma **1101-0** y el segundo como **0000-1.**

Como hemos visto el **-0** del primero y el **-1** del segundo nos dicen que es la **Paridad Impar**. Esto significa que nos sirve para ajustar en cada bloque de 5 bits el número de unos que aparecen en él, y que por definición debe ser un número impar. Así 1101 son tres bits a 1, que es número impar donde los haya, por lo que el bit de paridad impar debe ser 0 para que no se nos convierta en par. Y el segundo dato recibido es 0000 que no tiene ningún 1 y por ello debemos poner el bit de paridad a 1 para que en ese bloque también haya un número impar de unos.

Y ahora otro palito mas a la burra. Tenemos la obligación de darle la vuelta a los Bits recibidos y ponerlos como Dios manda. **1101** es en verdad **1011**, o escrito en hexadecimal resulta que es el número **'Bh'**, y que el 0000 es **0000**, o sea el hexadecimal '0h', como no podía ser de otra forma.

Resumiendo: Recibimos un montón de ceros, recibimos el dato 'Bh', recibimos el dato '0h'...

Y continúa esta feria de vanidades. El ABA Track 2 nos dice muchas mas cosas. Sus mandamientos no acaban en los dos primeros sino que se añaden por lo menos cuatro más a la lista.

Tercer mandamiento: **Tras la recepción del dato 'Bh'**, recuerda que era 11010 en binario sin darle la vuelta como un calcetín, **Recibirás mas datos en la misma forma que los anteriores**, hasta un máximo de 79, pero que pueden ser 78 ó 77 ó 76 ... o ... ¡Yá!

Un protocolo nos obliga a más cosas que al simple método a usar para ser capaces de recibir o emitir una información

Tras el tercero ha de venir un Cuarto Mandamiento: **Los datos se acaban del todo cuando recibas un dato 'Fh'**, o sea una tira de bits de la forma 11111 (Nota que son 4 unos, número par de unos, por lo que la paridad debe ser también un 1 para que sean 5 unos, impar y no violemos la paridad impar)

Con esto ya podríamos leer todos los datos completos de una transmisión *ABA Track II*, pero aún falta un detalle.

Nuestro protocolo nos dice que tras enviarnos el 'Fh' aún nos ha de enviar una dato más para que seamos capaces de saber si todo lo anterior que nos envió es correcto o no. Para ello nos envía un dato especial (especial pero no tanto, son cuatro bits mas paridad también) que no es cualquiera sino el resultado de calcular los sucesivos OR EXCLUSIVOS de todos los anteriores que nos ha enviado, empezando por el 'Bh' y terminando por el 'Fh'. A esto se les ocurre llamarlo el **LRC** o sea *Longitudinal Redundancy Chek* **o** *Comprobación Longitudinal Redundante.*

Y como no, por último, como traca final fin de fiesta, como estertor de una celebración de altura, otro sano, largo y completo montón de ceros final, diciendo hasta aquí hemos llegado, amigos. En la figura 3 vemos toda una transmisión (real) de una serie de datos en Protocolo *Serie Asíncrono ABA Track 2* que consiste en:

- *Una serie de 10 ceros.*
- *Start Character 'Bh' también conocido como "Sentinel"*
- *Los datos 0321215679127*
- *End Character 'Fh'*
- *LRC o dato calculado para comprobar transmisión*
- *Otra serie de 9 ceros.*

Todo esto puede verse comentado en la figura 4.

.Software:

Hemos escrito una funcion en CCS que permite la transmisión de Data & Clock en ABA Track 2:

Figura 3

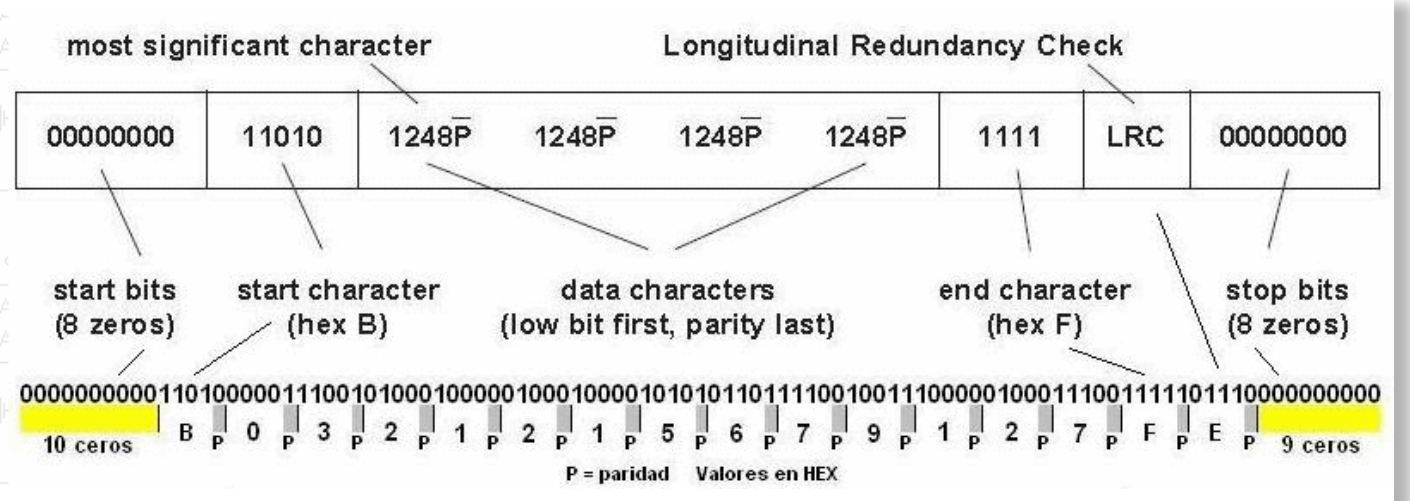


Figura 4

```
void  Transmite_Bit_Clock_and_Data(char c){

  // Data
  if((c&0x01)==0){
    output_high(OUT_DATA);
  }
  else{
    output_low(OUT_DATA);
  }
  // Clock
  delay_us(250);
  output_low(OUT_CLOCK);
  delay_us(250);
  output_high(OUT_CLOCK);
}

void Transmite_Byte_Clock_and_Data(char c){

  int i;
  char bt,paridad=0;

  for(i=0;i<4;i++){
    bt=(c>>i)&0x01;
    Transmite_Bit_Clock_and_Data(bt);
    paridad+=bt;
  }
  bt=~(paridad&0x01);
  Transmite_Bit_Clock_and_Data(bt);
}

void output_Code_clock_and_data(void){

  int i;
  char c,LCR=0x0B;

  // transmite cabecera de 15 ceros
  for(i=0;i<15;i++) Transmite_Bit_Clock_and_Data(0);
  // identificador de TX pista2
  Transmite_Byte_Clock_and_Data(0x0B);
  // bytes de code
  for(i=0;i<nextCodeChar;i++){
    c=(Code[i]-'0') & 0x0F;
    LCR=LCR^c;
    Transmite_Byte_Clock_and_Data(c);
  }
  // identificador fin
  Transmite_Byte_Clock_and_Data(0x0F);
  LCR=LCR^0x0F;
  // transmite LCR
  Transmite_Byte_Clock_and_Data(LCR);
  // transmite cola de 15 ceros
  for(i=0;i<15;i++) Transmite_Bit_Clock_and_Data(0);
}
```

LINKS:

Visualizador de líneas digitales RR Logical Analyzer :

http://picmania.garcia-cuervo.com/Proyectos_RRLogicalAnalyzer.php

//Marcos Lazcano//
marcos.lazcano@gmail.com

usando LCDs

segunda parte

En el artículo del número 1 de uControl, aprendimos a manejar los displays LCD. En esta nota veremos cómo utilizarlos asociados a un PIC. Utilizaremos tres de los lenguajes más populares en la programación de PICs: el Assembler, el PicBasic y también el lenguaje C de CCS.

Hasta ahora vimos que se necesita para el manejo de estos displays desde algún dispositivo.

Hoy veremos como manejarlos desde un microcontrolador PIC de Microchip.

Utilizaremos para manejar el display un PIC16F877A, uno de los PICs de gama media más populares desde el PIC16F84 (el más conocido del mundo).

Como hardware respetaremos el conexionado del **MicroPic Trainer**, que encontrarán en la página de **uControl.**

Usaremos el modo de control a 4 bits, utilizando el pin R/W en assembler y C, y sin utilizarlo en PBP.

Las señales para el bus de datos las tomaremos del Byte alto del PortD, conectándonos así:

D4 del display a PortD.4
D5 del display a PortD.5
D6 del display a PortD.6
D7 del display a PortD.7

Las señales para los pines de control las tomaremos del puerto B, conectándolas de la siguiente forma:

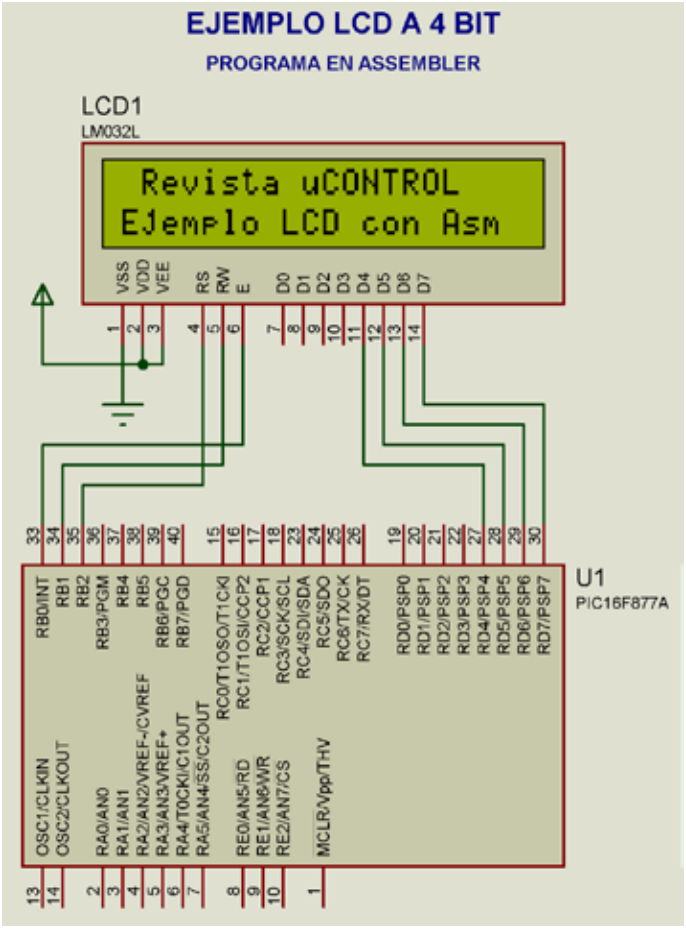
Enable (E) del display a PortB.0
Read/Write (R/W) del display a PortB.1
Register Select (RS) del display a PortB.2

En el caso del ejemplo CCS C los pines de control serán conectados al mismo puerto D, respetando las posiciones de la librería.

En otra nota veremos esa librería y veremos como modificarla.

.Código para usar el Display LCD en Assembler

El esquema de conexión para la versión en Assembler será el siguiente:



Circuito para la versión de software hecho en Assembler

El código en Assembler lo tomé y modifiqué para que fuera un solo archivo de una nota de aplicación de Microchip, originalmente para un PIC16C64, transportado a un PIC16F877A.

Deje los comentarios en el idioma original, ya que serian demasiados los cambios a realizar.

El código es el siguiente.

Ver Código en la siguiente página...

```
       LIST P=16F877A
        ERRORLEVEL  -302
;
; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
; module. The program assembles for either 4-bit or 8-bit data interface, depending
; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, RW ).
; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
;
       include <p16F877A.inc>

FALSE               EQU          0
TRUE                EQU          1

; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq      EQU           D'4000000'    ; Device Frequency is 4 MHz

DB_HI_BYTE    EQU           (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY      EQU           (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1

INNER_CNTR    EQU           40            ; RAM Location
OUTER_CNTR    EQU           41            ; RAM Location
;
T1OSO         EQU           0             ; The RC0 / T1OSO / T1CKI
;
RESET_V             EQU          0x0000        ; Address of RESET Vector
ISR_V         EQU           0x0004        ; Address of Interrupt Vector
PMEM_END      EQU           0x07FF        ; Last address in Program Memory
TABLE_ADDR    EQU           0x0400        ; Address where to start Tables
;
MSD           EQU           0x033         ; Temporary register, Holds Most Significant
                                          ; Digit of BIN to BCD conversion
LSD           EQU           0x034         ; Temporary register, Holds Least Significant
                                          ; Digit of BIN to BCD conversion
TEMP          EQU           0x035         ; Temporary register
CHAR          EQU           0x036         ; Temporary register, Holds value to send to LCD module.
;
WAIT_CNTR     EQU           0x040         ; Counter that holds wait time for key inputs
;
;
; LCD Module commands
;
DISP_ON             EQU    0x00C               ; Display on
DISP_ON_C     EQU    0x00E               ; Display on, Cursor on
DISP_ON_B     EQU    0x00F               ; Display on, Cursor on, Blink cursor
DISP_OFF      EQU    0x008               ; Display off
CLR_DISP      EQU    0x001               ; Clear the Display
ENTRY_INC     EQU    0x006               ;
DD_RAM_ADDR   EQU    0x080               ; Least Significant 7-bit are for address
DD_RAM_UL     EQU    0x080               ; Upper Left corner of the Display
;

LCD_DATA          EQU      PORTD
LCD_DATA_TRIS     EQU      TRISD
;
LCD_CNTL          EQU      PORTB
;
; LCD Display Commands and Control Signal names.
;
E             EQU           0             ; LCD Enable control line
RW            EQU           1             ; LCD Read/Write control line
RS            EQU           2             ; LCD Register Select control line
;
;
TEMP1            EQU      0x030
SavePort      EQU    0x031
;
      org     RESET_V                 ; RESET vector location
RESET
      GOTO    START               ;
;
; This is the Peripheral Interrupt routine. Should NOT get here
```

Continúa en la siguiente página...

```
;
    page
      org     ISR_V               ; Interrupt vector location
PER_INT_V
ERROR1
      BCF     STATUS, RP0     ; Bank 0
      BSF     PORTC, 0
      BCF     PORTC, 0
      GOTO    ERROR1
;
;
;
START                                 ; POWER_ON Reset (Beginning of program)
      CLRF    STATUS          ; Do initialization (Bank 0)
      CLRF    INTCON
      CLRF    PIR1
      BSF     STATUS, RP0     ; Bank 1
      MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
      MOVWF   OPTION_REG      ;
      CLRF    PIE1            ; Disable all peripheral interrupts
      MOVLW   0xFF            ;
      MOVWF   ADCON1          ; Port A is Digital.
;
;
      BCF     STATUS, RP0     ; Bank 0
      CLRF    PORTA           ; ALL PORT output should output Low.
      CLRF    PORTB
      CLRF    PORTC
      CLRF    PORTD
      CLRF    PORTE
      BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
;
      BSF     STATUS, RP0     ; Select Bank 1
      CLRF    TRISA           ; RA5 -  0 outputs
      MOVLW   0xF0        ;
      MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs

      CLRF    TRISC           ; RC Port are outputs
      BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
      CLRF    TRISD           ; RD Port are outputs
      CLRF    TRISE           ; RE Port are outputs
      BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
      BSF     OPTION_REG,NOT_RBPU  ; Disable PORTB pull-ups
      BCF     STATUS, RP0     ; Select Bank 0
;
    page
;
; Initialize the LCD Display Module
;
      CLRF    LCD_CNTL        ; ALL PORT output should output Low.

DISPLAY_INIT

      MOVLW   0x020           ; Command for 4-bit interface high nibble
      MOVWF   LCD_DATA        ;
      BSF     LCD_CNTL, E     ;
      BCF     LCD_CNTL, E     ;
;
; This routine takes the calculated times that the delay loop needs to
; be executed, based on the LCD_INIT_DELAY EQUate that includes the
; frequency of operation. These uses registers before they are needed to
; store the time.
;
LCD_DELAY
      MOVLW   LCD_INIT_DELAY  ;
      MOVWF   MSD             ; Use MSD and LSD Registers to Initialize LCD
      CLRF    LSD             ;
LOOP2
      DECFSZ  LSD, F          ; Delay time = MSD * ((3 * 256) + 3) * Tcy
      GOTO    LOOP2           ;
      DECFSZ  MSD, F          ;
END_LCD_DELAY
      GOTO    LOOP2           ;
;
```

Continúa en la siguiente página...

```
; Command sequence for 2 lines of 5x7 characters
;
CMD_SEQ
;
      MOVLW   0X020           ; 4-bit high nibble xfer
      MOVWF   LCD_DATA        ; This code for both 4-bit and 8-bit modes
      BSF     LCD_CNTL, E     ;
      BCF     LCD_CNTL, E     ;
;
      MOVLW   0x080           ; 4-bit high nibble xfer
      MOVWF   LCD_DATA        ;
      BSF     LCD_CNTL, E     ;
      BCF     LCD_CNTL, E     ;
;
; Busy Flag should be valid after this point
;
      MOVLW   DISP_ON         ;
      CALL    SEND_CMD        ;
      MOVLW   CLR_DISP        ;
      CALL    SEND_CMD        ;
      MOVLW   ENTRY_INC       ;
      CALL    SEND_CMD        ;
      MOVLW   DD_RAM_ADDR     ;
      CALL    SEND_CMD        ;
;
    page
;
      movlw   0               ;Table address of start of message
dispmsg2
      movwf   TEMP1           ;TEMP1 holds start of message address
      call    Table2
      andlw   0FFh            ;Check if at end of message (zero
      btfsc   STATUS,Z        ;returned at end)
      goto    out2
      call    SEND_CHAR       ;Display character
      movf    TEMP1,w         ;Point to next character
      addlw   1
      goto    dispmsg2
out2
      movlw   B'11000000'     ;Address DDRam first character, second line
          call   SEND_CMD

          movlw   0          ;Table address of start of message
         dispmsg
          movwf   TEMP1       ;TEMP1 holds start of message address
          call    Table
          andlw   0FFh       ;Check if at end of message (zero
          btfsc   STATUS,Z    ;returned at end)
          goto    out
          call    SEND_CHAR    ;Display character
          movf    TEMP1,w     ;Point to next character
      addlw   1
      goto    dispmsg

out
loop
      goto    loop            ;Stay here forever
;
;
INIT_DISPLAY
      MOVLW   DISP_ON             ; Display On, Cursor On
      CALL    SEND_CMD            ; Send This command to the Display Module
      MOVLW   CLR_DISP            ; Clear the Display
      CALL    SEND_CMD            ; Send This command to the Display Module
      MOVLW   ENTRY_INC           ; Set Entry Mode Inc., No shift
      CALL    SEND_CMD            ; Send This command to the Display Module
      RETURN
;
    page
;
;*****************************************************************
;* The LCD Module Subroutines                                    *
;*****************************************************************
;
```

Continúa en la siguiente página...

```
;
;*****************************************************************
;*SendChar - Sends character to LCD                              *
;*This routine splits the character into the upper and lower     *
;*nibbles and sends them to the LCD, upper nibble first.         *
;*****************************************************************
;
SEND_CHAR
      MOVWF   CHAR            ;Character to be sent is in W
      CALL    BUSY_CHECK      ;Wait for LCD to be ready
      MOVF    CHAR, w
      ANDLW   0xF0            ;Get upper nibble
      MOVWF   LCD_DATA        ;Send data to LCD
      BCF     LCD_CNTL, RW   ;Set LCD to read
      BSF     LCD_CNTL, RS    ;Set LCD to data mode
      BSF     LCD_CNTL, E     ;toggle E for LCD
      BCF     LCD_CNTL, E
      SWAPF   CHAR, w
      ANDLW   0xF0            ;Get lower nibble
      MOVWF   LCD_DATA        ;Send data to LCD
      BSF     LCD_CNTL, E     ;toggle E for LCD
      BCF     LCD_CNTL, E
      RETURN
;
    page
;
;*****************************************************************
;* SendCmd - Sends command to LCD                                *
;* This routine splits the command into the upper and lower      *
;* nibbles and sends them to the LCD, upper nibble first.        *
;* The data is transmitted on the PORT<3:0> pins                 *
;*****************************************************************
;
;
SEND_CMD
     MOVWF   CHAR            ; Character to be sent is in W
     CALL    BUSY_CHECK      ; Wait for LCD to be ready
     MOVF    CHAR,w
     ANDLW   0xF0            ; Get upper nibble
     MOVWF   LCD_DATA        ; Send data to LCD
     BCF     LCD_CNTL,RW    ; Set LCD to read
     BCF     LCD_CNTL,RS     ; Set LCD to command mode
     BSF     LCD_CNTL,E      ; toggle E for LCD
     BCF     LCD_CNTL,E
     SWAPF   CHAR,w
     ANDLW   0xF0            ; Get lower nibble
     MOVWF   LCD_DATA        ; Send data to LCD
     BSF     LCD_CNTL,E      ; toggle E for LCD
     BCF     LCD_CNTL,E
     RETURN
;
    page
;
;
;*****************************************************************
;* This routine checks the busy flag, returns when not busy      *
;*  Affects:                                                     *
;*      TEMP - Returned with busy/address                        *
;*****************************************************************
;
BUSY_CHECK
     BSF     STATUS, RP0     ; Select Register page 1
     MOVLW   0xF0 ;F           ; Set Port_D for input
     MOVWF   LCD_DATA_TRIS
     BCF     STATUS, RP0     ; Select Register page 0
     BCF     LCD_CNTL, RS    ; Set LCD for Command mode
     BSF     LCD_CNTL, RW   ; Setup to read busy flag
     BSF     LCD_CNTL, E     ; Set E high
     BCF     LCD_CNTL, E     ; Set E low
     MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
     ANDLW   0xF0            ; Mask out lower nibble
     MOVWF   TEMP
     BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
     BCF     LCD_CNTL, E
     SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
```

Continúa en la siguiente página...

```
      ANDLW   0x0F            ; Mask out upper nibble
      IORWF   TEMP            ; Combine nibbles
      BTFSC   TEMP, 7         ; Check busy flag, high = busy
      GOTO    BUSY_CHECK      ; If busy, check again
      BCF     LCD_CNTL, RW
      BSF     STATUS, RP0     ; Select Register page 1
      MOVLW   0x00 ;F
      MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
      BCF     STATUS, RP0     ; Select Register page 0
      RETURN
;
    page
;
Table
            addwf   PCL, F          ;Jump to char pointed to in W reg
            retlw   'E'
            retlw   'j'
            retlw   'e'
            retlw   'm'
            retlw   'p'
            retlw   'l'
            retlw   'o'
            retlw   ' '
            retlw   'L'
            retlw   'C'
            retlw   'D'
            retlw   ' '
            retlw   'c'
            retlw   'o'
            retlw   'n'
            retlw   ' '
            retlw   'A'
            retlw   's'
            retlw   'm'
            retlw   ' '
Table_End
            retlw   0
;
       if ( (Table & 0xFF) >= (Table_End & 0xFF) )
        MESSG   "Warning - User Defined: Table crosses page boundary in computed jump"
    endif
;
Table2
            addwf   PCL, F          ;Jump to char pointed to in W reg
            retlw   '*'
            retlw   'R'
            retlw   'e'
            retlw   'v'
            retlw   'i'
            retlw   's'
            retlw   't'
            retlw   'a'
            retlw   ' '
            retlw   'u'
            retlw   'C'
            retlw   'O'
            retlw   'N'
            retlw   'T'
            retlw   'R'
            retlw   'O'
            retlw   'L'
            retlw   ' '
            retlw   ' '
            retlw   ' '

Table2_End
            retlw   0
;
       if ( (Table2 & 0xFF) >= (Table2_End & 0xFF) )
        MESSG   "Warning - User Defined: Table crosses page boundary in computed jump"
    endif

    end
```

Este es el código de assembler.

.Código para usar el Display LCD en PicBasic Pro

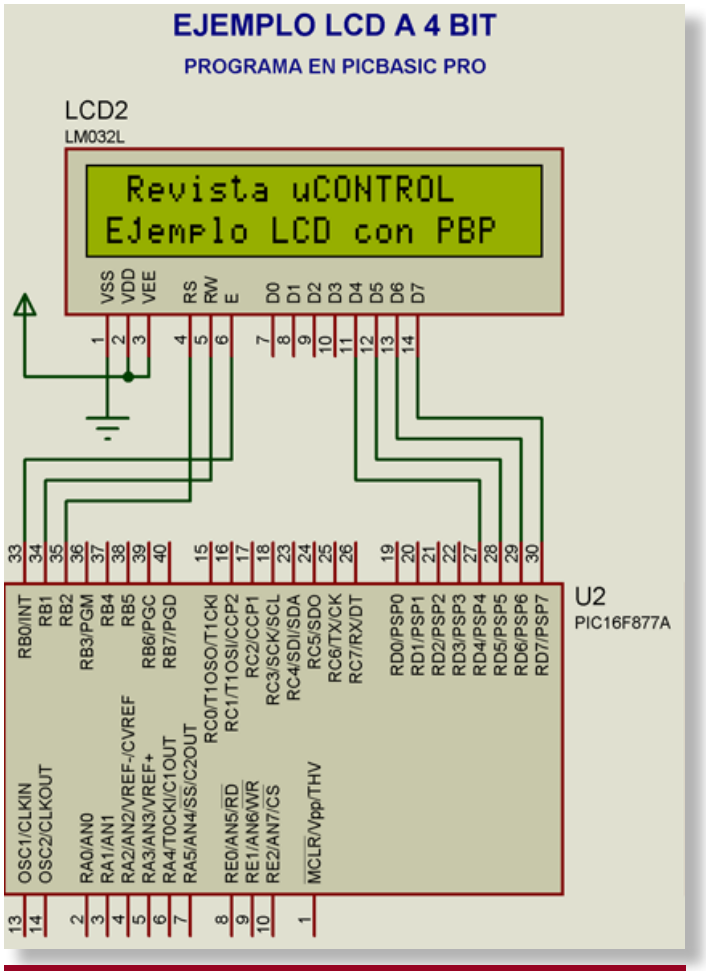
El esquema de conexión para la versión en PBP será el siguiente:

El código en PBP lo tomé y modifiqué de un ejemplo del mismo PicBasic Pro para la línea de 40 pines, realmente asombra lo corto que queda un programa para hacer el mismo despliegue que en Assembler.

El código es el siguiente.

```
Define ONINT_USED    1

‘ Define los registros y bits a usar en display
LCD
Define  LCD_DREG         PORTD
Define  LCD_DBIT         4
Define  LCD_RSREG        PORTB
Define  LCD_RSBIT        2
Define  LCD_EREG         PORTB
Define  LCD_EBIT         0


        ADCON1 = 7       ‘ Setea PORTA y POR
                         TE todo digital
        Low PORTB.1      ‘ Línea R/W del display
                         LCD en bajo (escribe)
        Pause 100        ‘ Espera para arranque
                         del display LCD


loop:   Lcdout $fe, 1    ‘ Borrar pantalla
        Pause 100        ‘ Espera antes de
                         mostrar mensaje

        Lcdout “ Revista uCONTROL  “   ‘ Mensaje
                                       en línea 1

        Lcdout $fe, $c0, “Ejemplo LCD con PBP  “
                        ‘ Mensaje en línea 2
        End
```

Código para el manejo del display en PicBasic Pro



Circuito para la versión de software hecho en PicBasic Pro

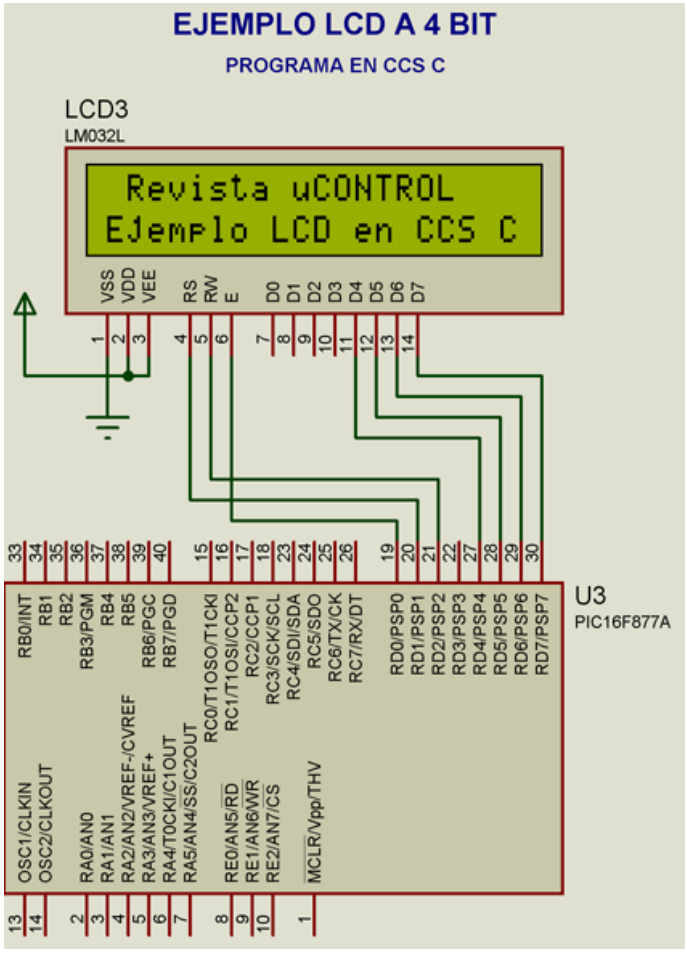
Código para usar el Display LCD en CCS C

El esquema de conexión para la versión en lenguaje C será el siguiente:

El código en lenguaje C lo tomé y modifiqué de un ejemplo del mismo CCS para la línea media de Microchip, también en este caso asombra lo corto que queda un programa para hacer el mismo despliegue que en Assembler.

El código es el siguiente.

Ver Código en la siguiente página...



Circuito para la versión de software hecho en lenguaje C

```
////////////////////////////////////////////////
////              LCD4bit.C                 ////
////////////////////////////////////////////////


#include <16F877A.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=4000000)

#include <lcd.c>

void main() {
   char k;

   lcd_init();

   lcd_putc("\f Revista uCONTROL  \n");
   lcd_putc("Ejemplo LCD en CCS C");
}
```

Código para el manejo del display en CCS C

.Conclusión

Los displays LCD pueden ser manejados en todos los lenguajes de programación de PICs.

Las diferencias consisten en que algunos lenguajes incorporan funciones preprogramadas (inaccesibles para el usuario) como tiene el PicBasic Pro, y otros disponen de librerías para el manejo de estos dispositivos.

En todos los casos, lo que hacen estas funciones preprogramadas o las librerías, es administrar los estados de los pines y puertos asignados al manejo del display , en su bus de datos o los pines de control, cumpliendo con los tiempos y sincronismo de las señales según los requerimientos del display que utilicemos.

En la próxima nota de esta serie explicaremos el uso de la librería de CCS para el display LCD, indicaremos como y porque reformarla para las necesidades de un proyecto en particular.

Referencias:

Microchip, fabricante de los microcontroladores PIC.

CCS INC, creador de la línea de compiladores CCS PCB, PCM, PCW y PCD .

Micro Engineering Labs, creador del compilador PicBasic Pro

Hoja de datos del controlador HD44780, de la firma Hitachi.

//Mariano Madrid Ortiz//
marianommo@gmail.com

detector de humo y gases microcontrolado

El proyecto consiste en un montaje sencillo, pero funcional, basado en el sensor de gases NAP-11AS, controlado con un PIC18F252 y con monitorización de estado a través del PC.
El sensor seleccionado utilizado frecuentemente en purificadores de aire, ventiladores y equipos de detección de polución, por ello no es recomendado como alarma de incendios sino como elemento de detección para aparatos de climatización.

En principio este proyecto tiene una finalidad eminentemente didáctica, y puede servirnos como aprendizaje del funcionamiento y control de cualquier sensor cuya salida varíe en tensión. Además nos brinda una introducción básica a la programación de PICs en lenguaje C (CCS), nos enseña a controlar su módulo de comunicación serie, su conversor A/D y la utilización de una librería para buzzers (tones.c) entre otras cosas.

.Descripción General de los componentes:

Para realizar el proyecto se ha diseñado una placa de circuito impreso con el software Orcad. De la lectura del sensor, las comunicaciones y las actuaciones se encarga "el cerebro" de esta placa, el PIC 18F252. La elección del modelo de PIC no es crítica, se podría utilizar cualquiera que dispusiera de modulo UART y conversor A/D. En mi caso opté por el 18F252 porque tenía una muestra que Microchip me envió amablemente.

El sensor es un "sensor olfativo" de tipo semiconductor. En el encapsulado de este componente, además del propio sensor, se encuentra una resistencia que sirve para calentar el aire de manera que el NAP11AS pueda realizar las mediciones de manera óptima. Este semiconductor es capaz de detectar desde humo de tabaco hasta café, pasando por insecticidas, cosméticos, olores de cocina, etc. (Ver recuadro 1)

Una vez el sensor ha detectado una concentración superior a la indicada de alguno de estos olores, se activan dos alarmas, una de tipo luminoso y otra sonora. Para la alarma luminosa, he utilizado un simple LED bicolor de cátodo común, que cambia de verde a rojo y para la sonora un buzzer piezoeléctrico (cualquier buzzer de 5V servirá).

Para la interface entre el microcontrolador y el puerto serie de nuestro ordenador, he utilizado el archiconocido circuito integrado **MAX232**. No es objeto de este proyecto el comentar sus funcionalidades, simplemente utilicé la configuración del esquema (Ver figura 1) proporcionada en su hoja de características.

Fuentes de Olor	Nivel de sensibilidad olfativa	Sensibilidad	Respuesta	Condiciones
Humo de tabaco	Fuerte	Alta	Rápida	5 cigarros
Cosméticos	Fuerte	Muy alta	Rápida	1 pulverización de colonia
Insecticidas	Fuerte	Muy alta	Rápida	Espray de 10 segundos
Carne frita	Media/Fuerte	Alta	Media	100g de cerdo
Cebollas fritas	Fuerte	Alta	Rápida	3 cebollas
Pimientos verdes fritos	Media	Media	Lenta	5 pimientos
Café	Débil	Baja	----------	5 tazas

Para alimentar todo el circuito he recurrido a otro clásico, el **LM7805**. Este regulador de tensión nos proporciona 5V constantes a la salida, si a la entrada introducimos una tensión superior a VCC+3V.

El conjunto se ha soldado en una placa de circuito impreso, mediante la técnica de insolación de placa fotosensible. Para ello se obtuvo previamente un fotolito mediante el software Orcad Layout (Ver imagen 2).

.Principio de funcionamiento:

Ya hemos comentado que para el perfecto funcionamiento del sensor, es necesario un calentamiento del aire circundante, esta es la misión de una resistencia interna que incluye el sensor. Una vez el aire es calentado, para lo cual es necesario un tiempo de establecimiento que dependerá de la tempera externa, podemos comenzar a tomar las medidas. Experimentalmente se determinó que en una habitación a temperatura de 22ºC, el nivel que arrojaba el sensor una vez alcanzado el punto de funcionamiento era del orden de 95-98 en nuestra escala de medida. Pera evitar falsas medidas, nuestro código (Ver recuadro 2), hace que esperemos forzosamente hasta que el sensor este operativo.

Para la medición se configuró un divisor de tensión entre la salida del sensor y la resistencia R2, del mismo orden de magnitud que el sensor en reposo. De esta manera, la salida del divisor es introducida al conversor A/D del PIC, el cual realiza las mediciones y muestra los resultados en el PC que le indicamos en el código (Ver recuadro2).

Una vez el sensor está listo para comenzar a medir y tras avisarnos de ello en el Hyperterminal, comienza a medir, si la medida es inferior a la variable “Umbral”, el LED verde permanece encendido y no escuchamos la señal acústica. Sin embargo, cuando el “Umbral” es sobrepasado, el LED se torna rojo y comenzamos a escuchar los repetitivos pitidos del buzzer.

Como funcionalidad añadida, podemos pulsar las teclas “+”, “-” y “m” del teclado. Pulsamos “+” para disminuir la sensibilidad al aumentar el “Umbral” o “-” para hacer lo contrario. La tecla “m” nos arroja la medición instantánea del ”Umbral”.

Como indiqué al principio, este es un proyecto con finalidad didáctica. Por ello, en aplicaciones finales, será necesaria una calibración más fina del sensor, en función del olor que deseemos controlar. Para ello es recomendable la lectura de la hoja de características que podéis encontrar en la sección de descargas de la página del autor, *http://www.GigAmperios.com*

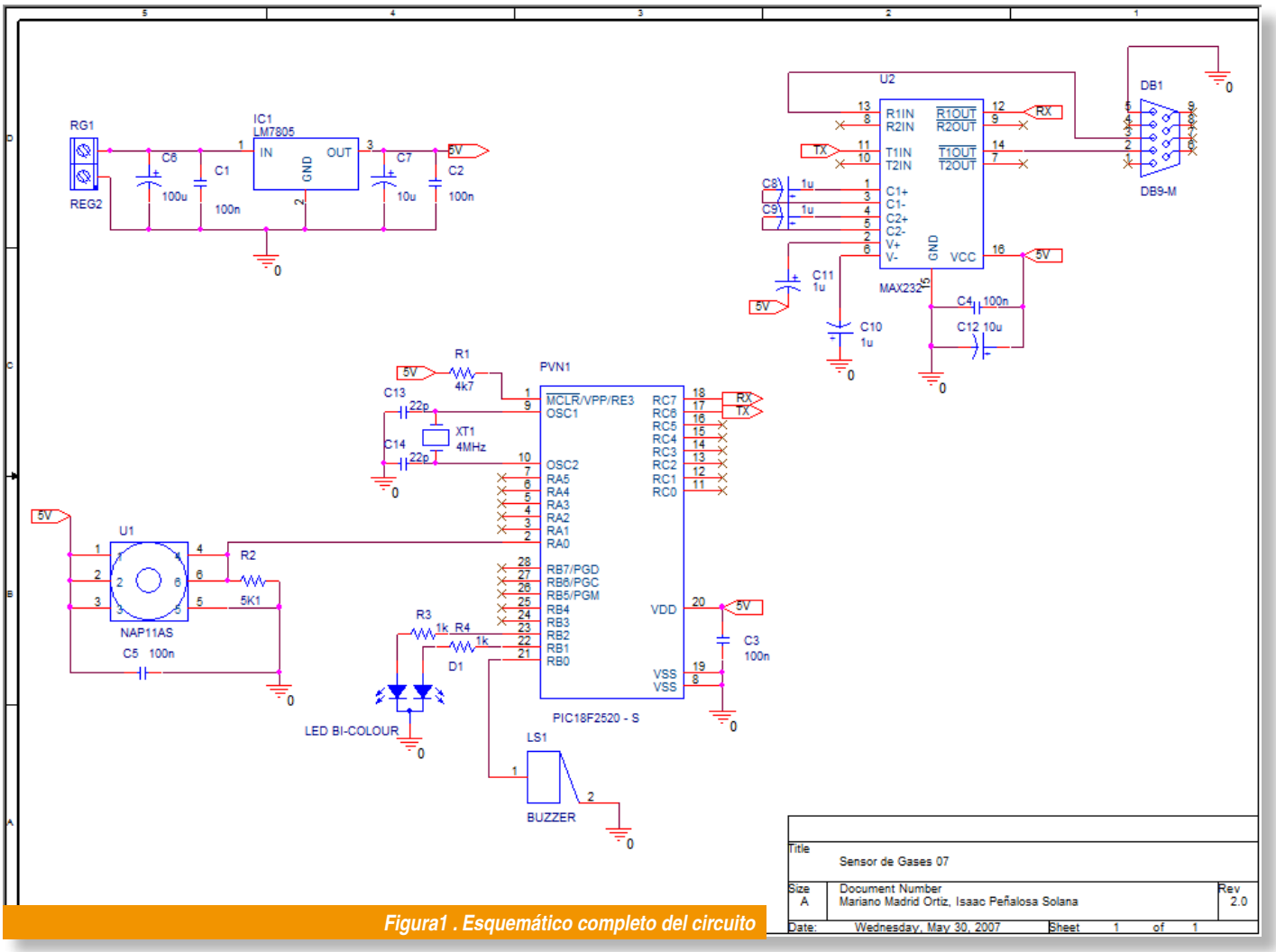
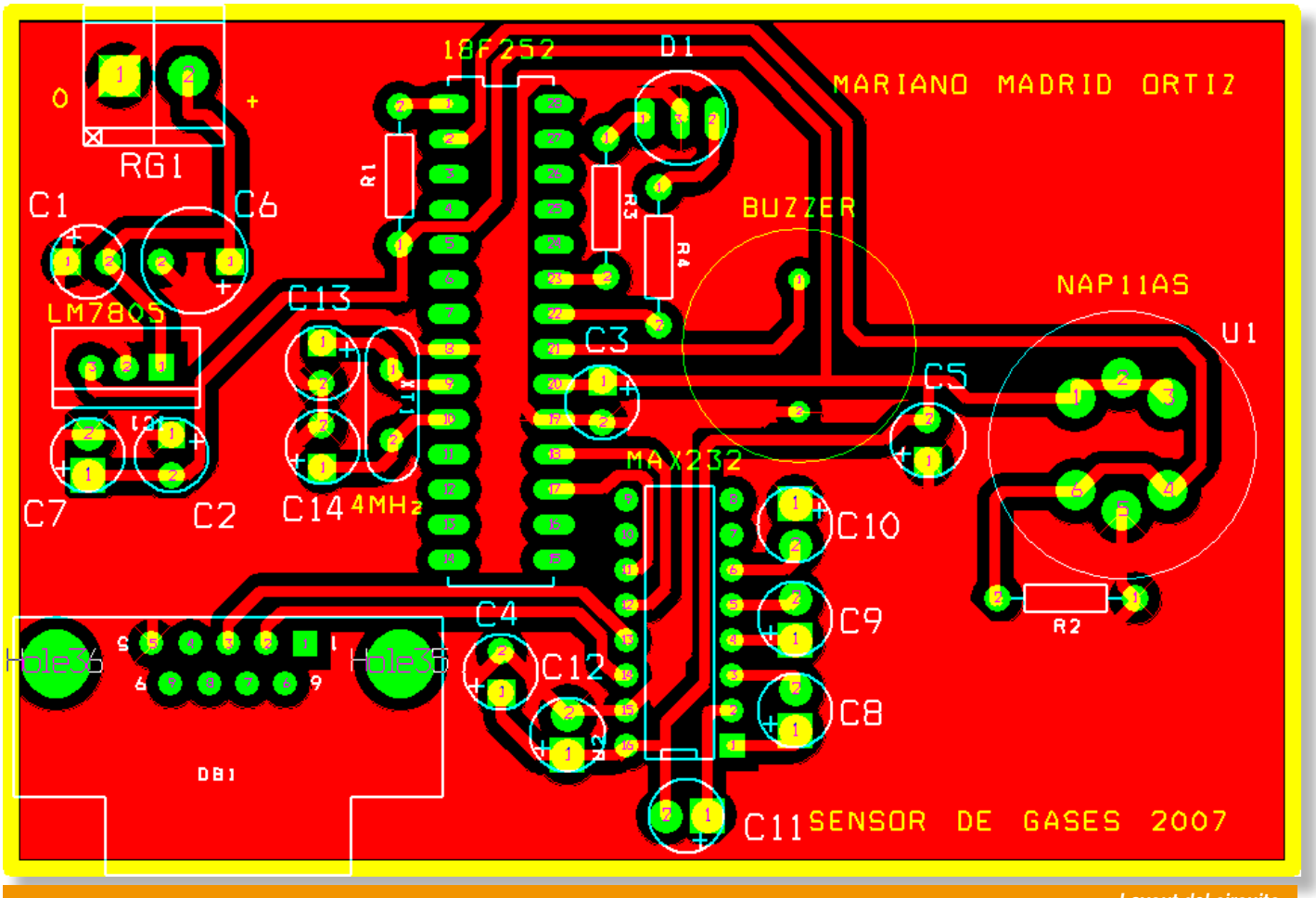


Figura1 . Esquemático completo del circuito

Código fuente del PIC:

```
#include <18f252.h>
#fuses XT,NOWDT,NOPROTECT,NOLVP,PUT,BROWNOUT
#use delay(clock=4000000)
#use standard_io(b)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#include <tones.c>

long adc_gas=0x00;
long Umbral=100;
long Nivel=0;
int var=0;
char Keypress=' ';

#int_rda
void rda_isr() {
   Keypress=0x00;
   if(kbhit()){
      Keypress=getc();
   }
}

void main() {

   setup_adc(ADC_CLOCK_INTERNAL);
   setup_adc_ports(RA0_ANALOG);

   enable_interrupts(int_rda);
   enable_interrupts(global);

   printf("\r\n" );
   printf("SENSOR DE GASES 07\r\n" );
   printf("\r\n" );
   printf("AUTORES:\r\n");
   printf("MARIANO MADRID ORTIZ\r\n" );
   printf("GigAmerios.com\r\n");
   printf("\r\n" );
   printf("\r\n" );
   printf("Para subir el Umbral de alarma, pulse +\r\n");
   printf("Para bajar el Umbral de alarma, pulse -\r\n");
   printf("Para monitorizar el Nivel del sensor, pulse m\r\n");
   printf("\r\n" );
   printf("\r\n" );
   printf("\r\n" );
   printf("Espere al calentamiento del dispositivo" );

   do {
       // Lectura del canal 0 -> AN0 GAS
       set_adc_channel(0);
       delay_ms(1);
       adc_gas=read_adc();
       delay_ms(100);
       Nivel=adc_gas;
       printf("." );

   }while(Nivel>98);

   printf("\r\n" );
   printf("El dispositivo esta listo para su uso\r\n" );


   do {
       // Lectura del canal 0 -> AN0 GAS
       set_adc_channel(0);
       delay_ms(1);
       adc_gas=read_adc();
       delay_ms(100);
```

```
        Nivel=adc_gas;

           if(Nivel<Umbral){
                output_high(PIN_B1);
                output_low(PIN_B2);
           }
           if(Nivel>Umbral){
                output_low(PIN_B1);
                output_high(PIN_B2);
                generate_tone(1000, 1000);
           }

        if(Keypress!=0x00){

              switch(Keypress){

                 case '+': Umbral=Umbral+10;
                           printf("Disminuyendo sensibilidad\r\n" );
                           printf("El nuevo Umbral de alarma es: %Lu\r\n",Umbral);
                           break;
                 case '-': Umbral=Umbral-10;
                           printf("Aumentando sensibilidad\r\n" );
                           printf("El nuevo Umbral de alarma es: %Lu\r\n",Umbral);
                           break;
                 case 'm': printf("El nivel de gas es de: %Lu\r\n",adc_gas);
                           break;

              }
              Keypress=0x00;
           }
      } while (TRUE);

}
```



Layout del circuito

//Ariel Palazzesi//
arielpalazzesi@gmail.com

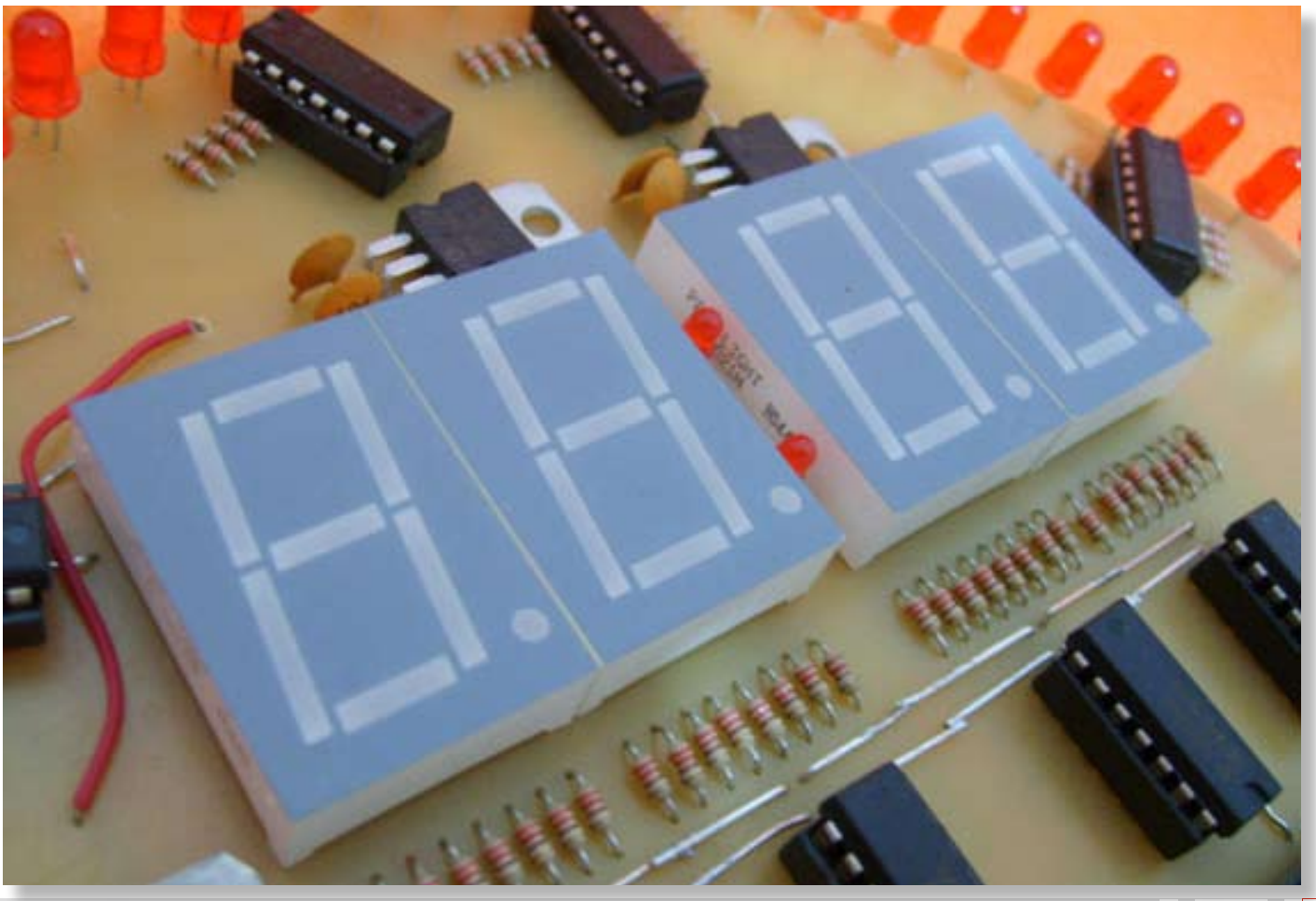
"el relojito" segunda parte

Seguimos avanzando con la programación de nuestro relojito. Esta vez le toca el turno al display que debe mostrar las horas y los minutos. Montado a partir de 4 displays de 7 segmentos y un registro de desplazamiento, la construcción de rutinas o funciones que lo controlen representa un desafío muy interesante, que ya mismo abordaremos.

Como vimos en la descripción del hardware de este proyecto, la visualización de las horas y los minutos se realiza mediante cuatro displays de 7 segmentos de cátodo común. Esto significa que para encender alguno de sus segmentos debemos proporcionar tensión al pin correspondiente de cada modulo. Para evitar utilizar varios pines del microcontrolador PIC16F628A elegido como "cerebro" del proyecto al control de este display, se utilizó un registro de desplazamiento compuesto por 4 circuitos integrados 74HC164N. Cada uno de ellos se encarga del control de los 7 segmentos y del punto decimal de un display.

El nombre de los segmentos de cada display, según la mayoría de las hojas de datos, es el de la figura 1. Allí podemos ver que se nombran con letras de la "a" (el segmento superior del "8") hasta la "g" (el segmento central), avanzando en sentido horario. El punto decimal suele llamarse "dp" (supongo que por "dot point"), pero nosotros lo llamaremos "h".

Cada uno de los segmentos (y el punto decimal) de cada display se encuentran conectados, mediante un resistor que limita la corriente que los atraviesa, a una de las salidas de los 74HC164N. Estos están conectados en "cascada", por lo que cuando un dato "sale" de unos de los integrados se aplica a la entrada del siguiente. Esto sig-



nifica que con solo dos pines (CLOCK y DATA) podemos escribir los 4 displays.

Los primeros 8 bits enviados al registro de desplazamiento serán los encargados de determinar el encendido de los segmentos del display de la izquierda (las decenas de las horas). Los siguientes 8 bits controlarán las unidades de las horas, los 8 que vienen a continuación manejarán el display que muestra las decenas de los minutos, y los últimos 8 bits determinaran el contenido del display que muestra las unidades de los minutos.

Esto quiere decir que si quisiésemos mostrar "23:15" en el display, primero deberíamos enviar los datos del "2", luego los del "3", los del "1" y finalmente los correspondientes al "5".

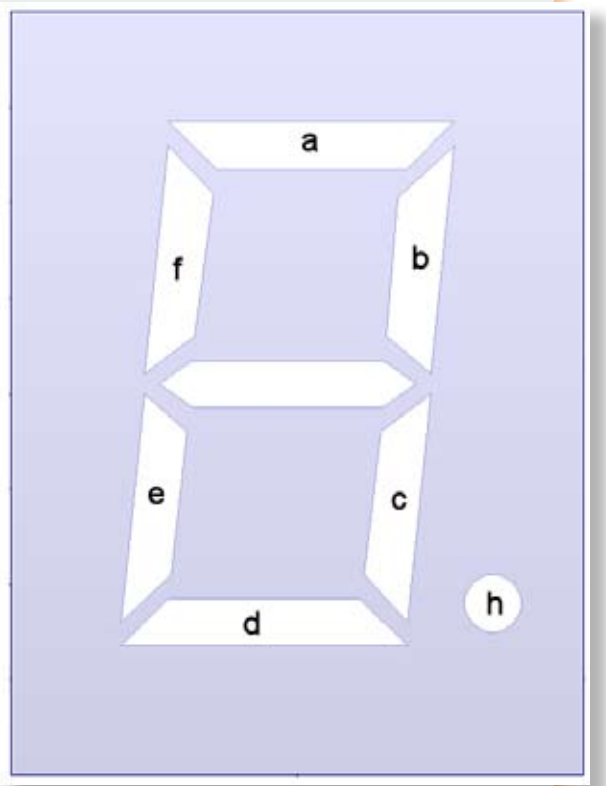
Para saber cual es el contenido que debemos enviar para representar cada digito es necesario que tengamos bien presente la forma en que los circuitos integrados 74HC164N están conectados a los displays.

Si miramos el esquema eléctrico de la figura 2, veremos que el primer bit ingresado se encarga del encendido (o apagado) del segmento "d" del display, luego de ser "empujado" por los 7 bits correspondientes a los demás segmentos del display. Concretamente, el orden en que deben ingresarse los datos es "d", "h", "c", "g", "b", "a", "f" y "e". La tabla de la figura 3 muestra el valor de cada uno de estos bits para formara cada uno de los dígitos del 0 al 9. Hemos incluido el valor del byte en decimal y binario, para facilitar al lector la programación del display.

.¡A programar!

Una vez que tenemos claro como debemos proceder, veamos como escribir un programa que muestre información en el display. Comencemos por un ejemplo que muestra como enviar un "2" al registro desplazamiento. El código en PIC BASIC es el siguiente:

```
'-----CONFIGURO PUERTOS----------
AllDigital

'Configuro el portA:
TRISA.2 = 0  'DATA HH:MM
TRISA.3 = 0  'CLOCK HH:MM
'Configuro el portB:
TRISB.1 = 0  'Salida, LEDs : en HH:MM

'------VARIABLES------------------
Dim i As Byte
Dim col As Byte
Dim aux As Byte

'----- Symbol -----------------
Symbol clock7 = PORTA.3
Symbol data7 = PORTA.2

'Limpio el contenido del registro de desplazamiento
'escribiendo 32 "0" seguidos:
For i = 1 To 32
               data7 = 0
               clock7 = 0
               clock7 = 1
Next i

'Escribo un "2" en el primer display
aux = 157  'Valor decimal de "2" (ver tabla)

'Este bucle recorre el byte enviando sus bits
'al registro de desplazamiento:
For col = 1 To 8
               'Si el bit es "0", escribo un "0".
               If aux.0 = 0 Then
                      data7 = 0
                      clock7 = 0
                      clock7 = 1
               Else
               'Si el bit es "1", escribo un "1".
                      data7 = 1
                      clock7 = 0
                      clock7 = 1
               Endif
               'Paso al bit siguiente
               aux = ShiftRight(aux, 1)
Next col
```

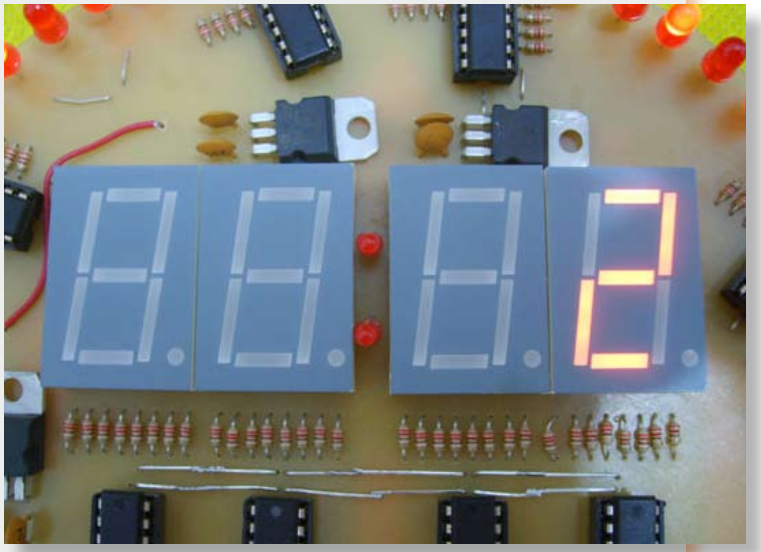


Típico display de 7 segmentos

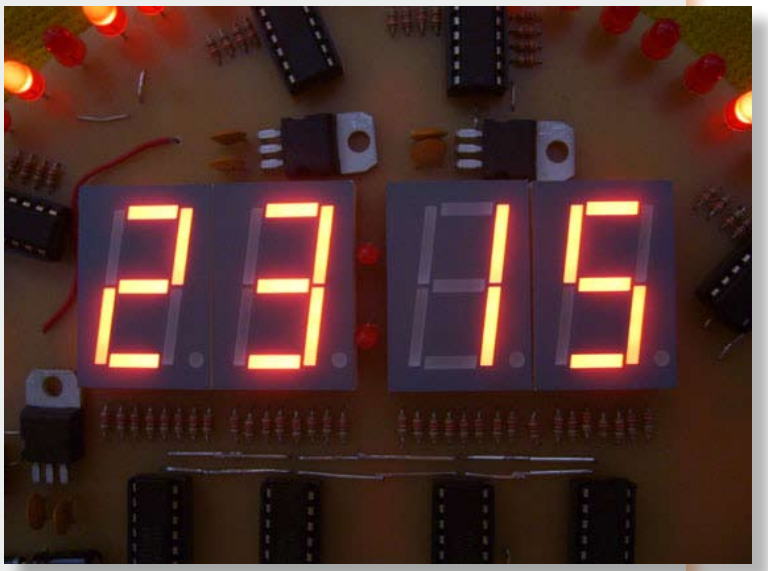
En nuestro sitio web encontrarás un video que muestra como se van “corriendo” los datos por el registro de desplazamiento hasta formar el “2”. Por supuesto, hemos agregado un retardo de un segundo después de enviar cada bit, para que pueda verse como funciona.

Ahora, modifiquemos el programa anterior para que podamos mostrar la hora “23:15” en el display. Como puede verse, hemos transformado las instrucciones que se encargan de enviar los 8 bits en una rutina, a la que llamamos 4 veces, pasándole como dato (en “aux”) el byte a escribir:

```
‘-----CONFIGURO PUERTOS----------
AllDigital

‘Configuro el portA:
TRISA.2 = 0  ‘DATA HH:MM
TRISA.3 = 0  ‘CLOCK HH:MM
‘Configuro el portB:
TRISB.1 = 0  ‘Salida, LEDs : en HH:MM

‘------VARIABLES------------------
Dim i As Byte  ‘Variable auxiliar
Dim col As Byte
Dim aux As Byte  ‘Variable auxiliar uso gral (WORD)

‘----- Symbol -----------------
Symbol clock7 = PORTA.3
Symbol data7 = PORTA.2

‘Limpio el contenido del registro de desplazamiento
‘escribiendo 32 “0” seguidos:

For i = 1 To 32
                data7 = 0
                clock7 = 0
                clock7 = 1
Next i
‘Escribo un “2” en el primer display
aux = 157  ‘Valor decimal de “2” (ver tabla)
Gosub escribo
‘Escribo un “3” en el segundo display
aux = 188  ‘Valor decimal de “3” (ver tabla)
Gosub escribo
‘Escribo un “1” en el tercer display
aux = 40  ‘Valor decimal de “1” (ver tabla).
Gosub escribo
‘Escribo un “5” en el cuarto display
aux = 182  ‘Valor decimal de “5” (ver tabla)
Gosub escribo

End
‘Este bucle recorre el byte enviando sus bits
‘al registro de desplazamiento:
escribo:
For col = 1 To 8
                ‘Si el bit es “0”, escribo un “0”.
                If aux.0 = 0 Then
                        data7 = 0
                        clock7 = 0
                        clock7 = 1
                Else
                ‘Si el bit es “1”, escribo un “1”.
                        data7 = 1
                        clock7 = 0
                        clock7 = 1
                Endif
                ‘Paso al bit siguiente
                aux = ShiftRight(aux, 1)
Next col
Return
```



Un “2” en el display



El display mostrando horas y minutos

Veamos este último ejemplo, pero escrito en CCS:

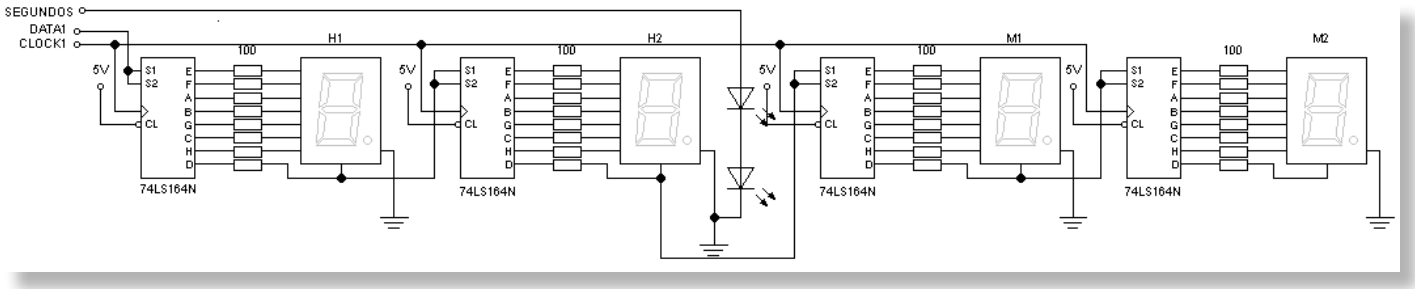
```
//Device/Fuses/Etc.--------------------------------
#INCLUDE <16F628A.H>            //Usamos un 16F628A
#FUSES NOWDT                    //No Watch Dog Timer
#FUSES XT                       //Con oscilador a cristal...
#use delay(clock=4000000)       //..de 4MHz.
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOPROTECT                //No protegemos el codigo.
#FUSES NOBROWNOUT               //No Brownout Reset
#FUSES NOLVP                    //No low voltage prgming
#FUSES NOCPD                    //No EE protection

//Declaramos la posición de los puertos-----------------------
#BYTE PORTA = 0x05
#BYTE PORTB = 0x06
#BYTE PORTA_TRIS = 0x85
#BYTE PORTB_TRIS = 0x86

//Definimos el valor de CLOCK y DATA
#DEFINE DATA7  PIN_A2 //Nos referimos a PORTA.2 como “data7”
#DEFINE CLOCK7 PIN_A3 //Nos referimos a PORTA.3 como “clock7”

//-------------------------------------------------
//---Envia un digito al registro de desplazamiento:
//-------------------------------------------------
void escribo(int8 aux){
   int i;
   for (i=0;i<8;i++) {   // “i” irá de 0 a 7, de 1 en 1.
      //Si el bit es “0”, escribo un “0”.
      if (bit_test(aux,i) == 0) {
         output_low(DATA7);    //Pongo “0” en DATA7...
         output_low(CLOCK7);   //Pongo el CLOCK en bajo...
         output_high(CLOCK7);  //...y de nuevo en alto. Listo!
      }
      //Si el bit es “1”, escribo un “1”.
      if (bit_test(aux,i) == 1) {
         output_high(DATA7);    //Pongo “1” en DATA7...
         output_low(CLOCK7);   //Pongo el CLOCK en bajo...
         output_high(CLOCK7);  //...y de nuevo en alto. Listo!
      }
   }
}
//-------------------------------------------------
//---Limpia el display:
//-------------------------------------------------
void borro_display(void){
   int i;
   for (i=1;i<33;i++) {   // “i” irá de 1 a 32, de 1 en 1.
         output_low(DATA7);    //Pongo “0” en DATA7...
         output_low(CLOCK7);   //Pongo el CLOCK en bajo...
         output_high(CLOCK7);  //...y de nuevo en alto. Listo!
   }
}
main(){
   //Asignamos cada pin como E/S según corresponda:
   PORTA_TRIS = 0b00000000; //1=ENTRADA, 0=SALIDA
   PORTB_TRIS = 0b11110001; //1=ENTRADA, 0=SALIDA
   //Limpiamos el display
   borro_display();
   //Escribo un “2” en el primer display
   escribo(157); //  ‘Valor decimal de “2” (ver tabla)
   //Escribo un “3” en el segundo display
   escribo(188); //  ‘Valor decimal de “3” (ver tabla)
   //Escribo un “1” en el tercer display
   escribo(40); //  ‘Valor decimal de “1” (ver tabla)
   //Escribo un “5” en el cuarto display
   escribo(182); //  ‘Valor decimal de “5” (ver tabla)
}
```

Como puede suponerse, el redibujado del display ocurre a tal velocidad que es imperceptible para el ojo. Y al no tener los dígitos correspondientes a los segundos, solo debe escribirse el display una vez por minuto. El tiempo que insume enviar los 32 datos al registro de desplazamiento ronda los 160 microsegundos.

Suponiendo que tenemos resuelto el mecanismo que cada un segundo pone un flan en alto (algo que veremos en el próximo numero, usando interrupciones), deberíamos escribir una rutina (o una función, si usamos CCS) que actualice el display y el "segundero" cuanto corresponda. Cada un tiempo determinado una "bandera" se pondría en "1", y el cuerpo principal del programa debería actualizar la hora y mostrarla en el display. Eso, justamente, es lo que hace el siguiente ejemplo en PIC BASIC:

```
'Esta rutina muestra HH:MM
muestro_hhmm:
        'Muestro las decenas de las horas
        aux1 = hora / 10
        aux = LookUp(175, 40, 157, 188, 58, 182, 183, 46, 191, 190), aux1
        Gosub escribo

        'Muestro las unidades de las horas
        aux1 = hora - (hora / 10) * 10
        aux = LookUp(175, 40, 157, 188, 58, 182, 183, 46, 191, 190), aux1
        Gosub escribo

        'Muestro las decenas de los minutos
        aux1 = minu / 10
        aux = LookUp(175, 40, 157, 188, 58, 182, 183, 46, 191, 190), aux1
        Gosub escribo

        'Muestro las unidades de los minutos
        aux1 = minu - (minu / 10) * 10
        aux = LookUp(175, 40, 157, 188, 58, 182, 183, 46, 191, 190), aux1
        Gosub escribo
Return
```

Se asume que aux1 y aux están declaradas como "BYTE", y que la rutina "escribo" vista antes esta presente en el programa.



Los 74HC167N están conectados "en cascada

Digito a mostrar	d	h	c	g	b	a	f	e	Binario	Decimal
1	0	0	1	0	1	0	0	0	00101000	40
2	1	0	0	1	1	1	0	1	10011101	157
3	1	0	1	1	1	1	0	0	10111100	188
4	0	0	1	1	1	0	1	0	00111010	58
5	1	0	1	1	0	1	1	0	10110110	182
6	1	0	1	1	0	1	1	1	10110111	183
7	0	0	1	0	1	1	1	0	00101110	46
8	1	0	1	1	1	1	1	1	10111111	191
9	1	0	1	1	1	1	1	0	10111110	190
0	1	0	1	0	1	1	1	1	10101111	175

Valores a enviar para formara cada uno de los dígitos

//Ariel Palazzesi//
arielpalazzesi@gmail.com

Paleotrónica

SID 6581, el mejor chip generador de sonidos de la historia

Han pasado ya 25 años de su nacimiento, pero aún es buscado por músicos y DJ. Desarrollado especialmente para dotar de sonido a los ordenadores Commodore, el chip SID 6581 de MOS Technology es considerado por muchos profesionales como "indispensable" para generar efectos de sonido "retro". Responsable en gran medida del éxito arrollador del Commodore, se ha transformado ya en un objeto de culto entre los expertos.

Por regla general, en el mundo de la electrónica los componentes más nuevos superan en prestaciones a los antiguos. Casi todos los circuitos integrados son considerados obsoletos en solo un puñado de años. Sin embargo, en el ámbito de la generación de música electrónica hay un chip que a pesar de tener ya 25 años de edad, sigue siendo buscado por los expertos por poseer características que lo hacen único. Nos estamos refiriendo al SID (Sound Interface Device) 6581, y también a su "primo" el SID 8580, ambos de MOS Technology.

Este circuito integrado fue el chip de sonido incorporado en los ordenadores *CBM-II*, *Commodore 64*, *Commodore 128* y *Commodore MAX Machine*, todos de la empresa Commodore. En los 1980s, las denominadas "Home Computers" no tenían una tarjeta de sonido como los ordenadores actuales, sino que la mayoría de las veces se limitaban a emitir algún sonido simple mediante un pequeño parlante incorporado. Sin embargo, Commodore apostó fuerte al campo de la generación de sonidos complejos, algo que le permitiría desarrollar juegos más atractivos, y puso a sus ingenieros a trabajar en el diseño de un chip especializado. La empresa suponía, con razón, que esto la pondría una un paso por delante de la competencia.

> En los 1980s, las denominadas "Home Computers" no tenían una tarjeta de sonido como los ordenadores actuales.

MOS Technology, una empresa especializada en la fabricación de circuitos integrados, había sido adquirida por Commodore no mucho tiempo atrás. Esto le permitía al fabricante de *Home Computers* obtener microprocesadores a un muy bajo precio, y, como finalmente hizo con estos chips de sonido, diseñar sus propios circuitos integrados a medida. Como no podía ser de otra forma, MOS fue la elegida para la fabricación del nuevo chip, allá por el año 1982. El resultado fue un integrado sintetizador y generador de efectos de sonido de solo 28 pines y compatible con la familia de microprocesadores **65XX** que empleaba Commodore en sus ordenadores. El chip SID fue creado por un equipo dirigido por el ingeniero **Robert Yannes**, quien más tarde fundaría la compañía de sintetizadores *Ensoniq*.

El **SID6581** dispone en su interior de toda la circuitería de control necesaria para sacarle provecho, lo que permite una sencilla programación y el uso de muy pocos componentes externos. Una de las figuras que acompaña este artículo muestra que tan simple puede ser conectar un **SID6581** a un microprocesador o microcontrolador. Solo se necesitan 5 líneas de direcciones, 8 de datos y tres de control.

El SID provee un control amplio y preciso de la frecuencia, contenido armónico y volumen del sonido que genera. El chip original (SID 6581) no se destacaba precisamente por el rendimiento de sus filtros, a pesar de que estos son un componente vital en la síntesis de sonido analógica. Como es fácil de deducir, puede imaginarse, No podía rivalizar con los sintetizadores comerciales de la época. Por eso, a finales de la década de 1980, Commodore sustituyó los **SID6581** un modelo más complejo, que fue bautizado con el nombre **SID8580**, que solucionaba, entre otros detalles, el mencionado problema de los filtros. El Commodore 128 fue el primer ordenador en incluir de fábrica el nuevo chip.

Ya ha pasado un cuarto de siglo, y ambos chips han sido descatalogados por el fabricante. Esto ha obligado a los

fanáticos de la música digital a buscar desesperadamente ordenadores Commodore en desuso para "canibalizarlos" y obtener sus chips de sonido. De hecho, es una suerte que se hayan vendido millones de ellos, ya que eso facilita en gran medida la búsqueda. No obstante, el valor de estos integrados suele llegar hasta los 30 o 35 euros en los sitios de subastas.

Entre las características más sobresalientes de los chips SID se incluyen sus 3 osciladores (0 a 4KHz.), con 4 formas de onda por oscilador (Triangulo, Diente de Sierra, Pulso variable y Ruido). Cada oscilador tiene su modulador de amplitud (de 48dB.) y su generador de envolvente. Es posible fijar el rango de Ataque (2 ms a 8 s), de Decaimiento (6 ms a 24 s), de Sostenimiento (0 a volumen de pico) y el rango de Relajación (6 ms a 24 s).

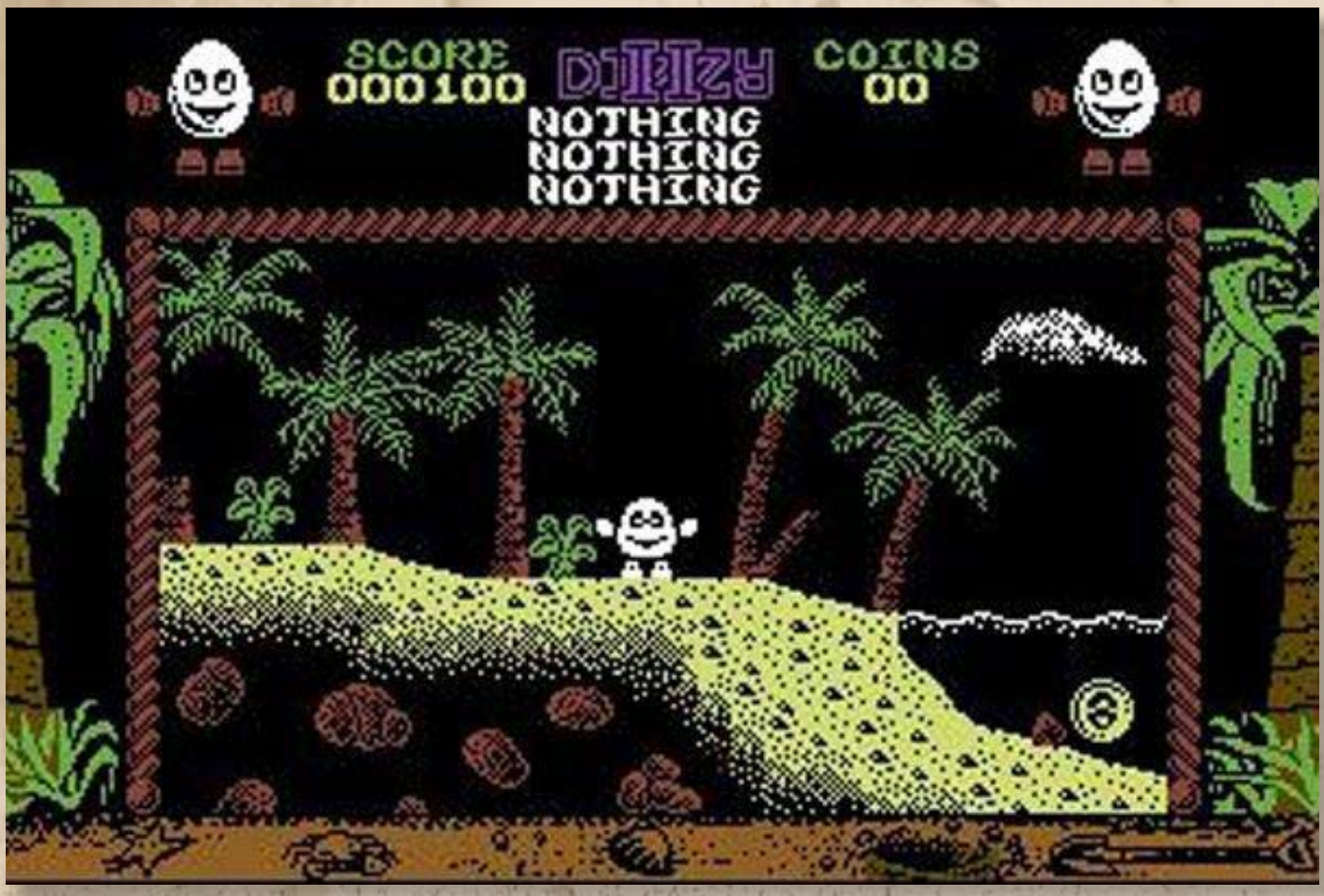
Los osciladores internos pueden ser sincronizados entre sí, y el filtro programable tiene un rango de corte de 30Hz. a 12KHz., con salidas de paso bajo, alto, banda y eliminación de banda. Dispone de un *Control Maestro de Volumen*, 2 interfaces para potenciómetros (con los correspondientes conversores A/D), un generador de números aleatorios y una entrada de audio externo que permite conectar varios de estos chips en cascada. Realmente, una colección de características impresionante, máxime si tenemos en cuenta la edad de estos pequeños chips.

Tan interesante es el sonido de este chip, que varias bandas (el caso de la Argentina Miranda) lo utilizan como parte de su bagaje tecnológico.

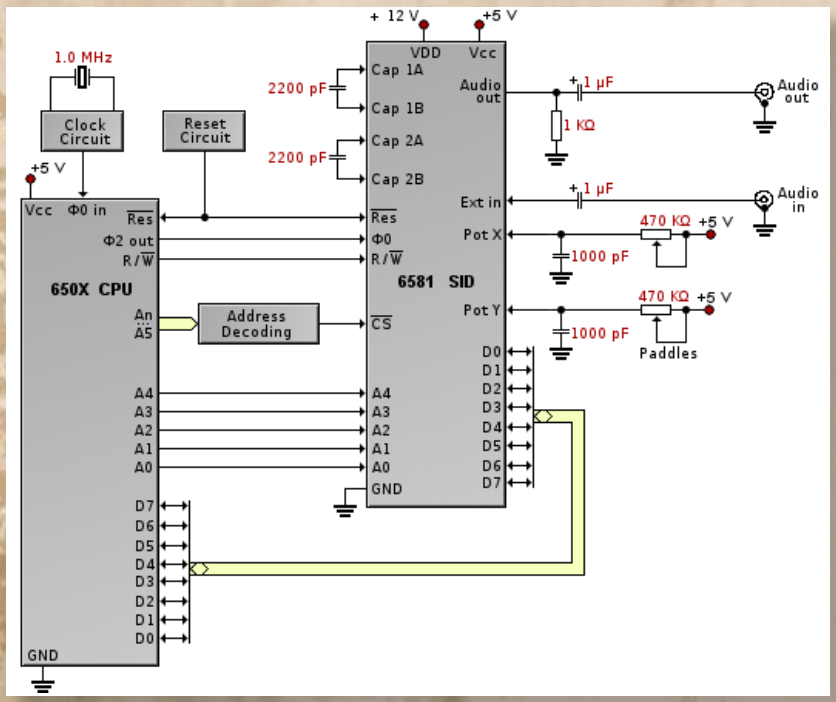
En la red es posible encontrar varios productos especiales para DJ que emplean estos chips como "cerebro". Varios de ellos utilizan más de un SID (y, por supuesto, algún microcontrolador) para generar espectaculares efectos de sonido o melodías. De hecho, el formato de audio ".SID", también llamado "fichero PSID", consiste en un archivo de datos de sonido que contiene las notas y el código en ensamblador del CMOS 6502 necesario para reproducir la música en el SID. Si quieres saber cómo sonaban estos chips, puedes descargar algunas de las 30.000 canciones en este formato disponibles gratuitamente en Internet. Solo necesitas instalar un Plug-In en Winamp (Chipamp) para poder reproducirlos en tu ordenador.

En la red es posible encontrar varios productos especiales para DJ que emplean estos chips como "cerebro".

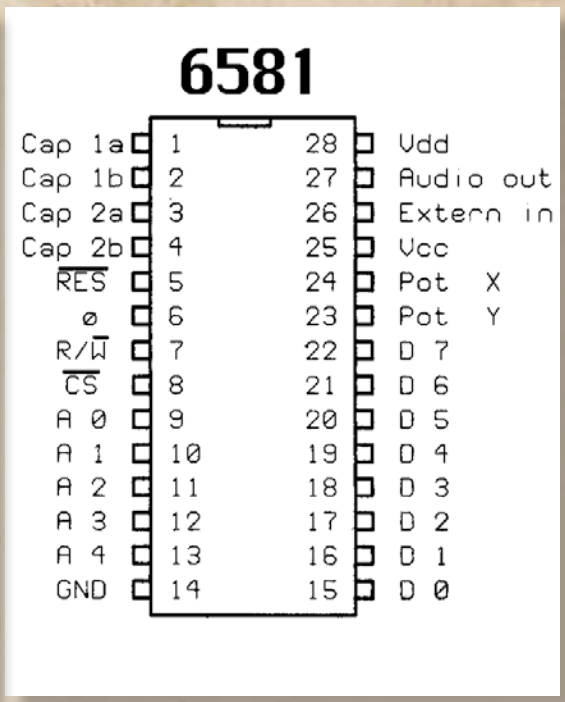
El MOS SID 6581 tiene la Patente nº 4,677,890, solicitada el 27 de febrero de 1983 y concedida el 7 de julio de 1987. Extrañamente, dicha patente expiró el 7 de julio de 2004, pero a pesar de ello y de lo buscados que son estos viejos circuitos integrados, ningún otro fabricante lo está comercializando.



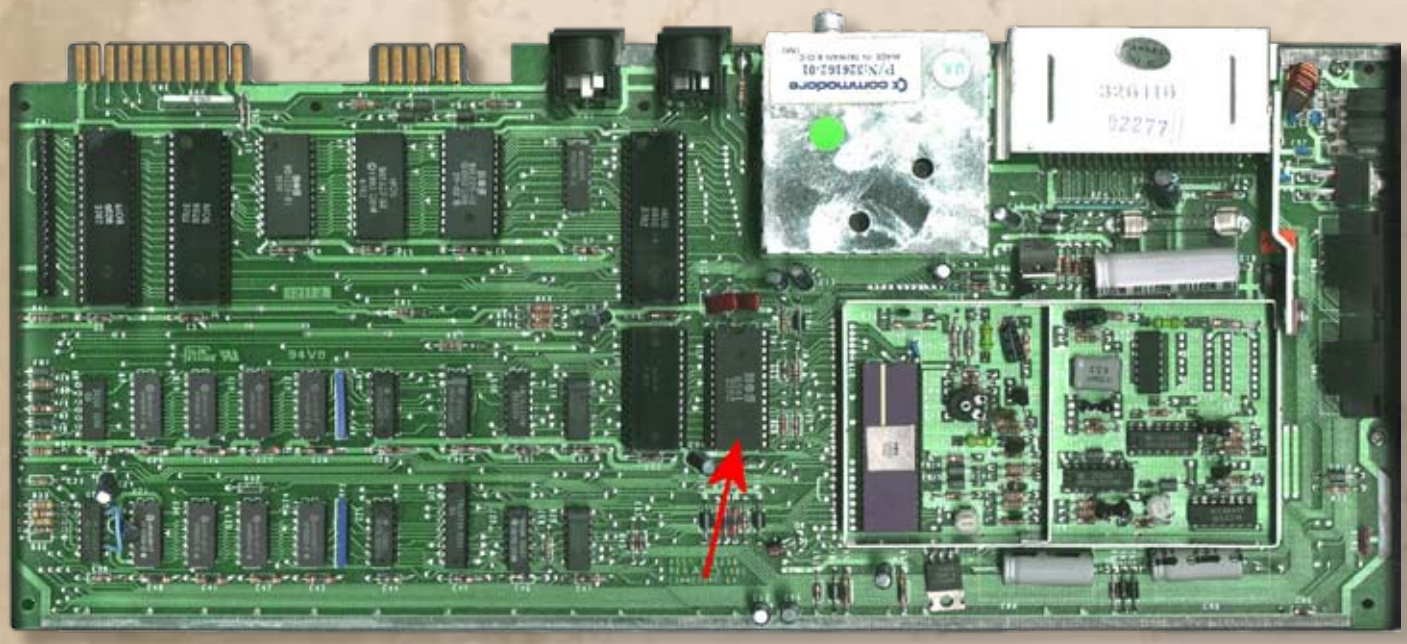
Los juegos del Commodore64 aprovecharon el sonido de estos chips



Este chip necesita de muy pocos componentes externos.



Disposición de pines del SID6581."



Placa base de un C64, con el SID SID6581 indicado con una flecha.

PROXIMAMENTE...
GRAN CONCURSO

µCONTROL
Electrónica en General Pics en Particular

www.ucontrol.com.ar

no te lo puedes perder!

Diseño y Diagramación
azimut.estudio@gmail.com / la plata / bs as / argentina