// DESCARGA GRATUITA /

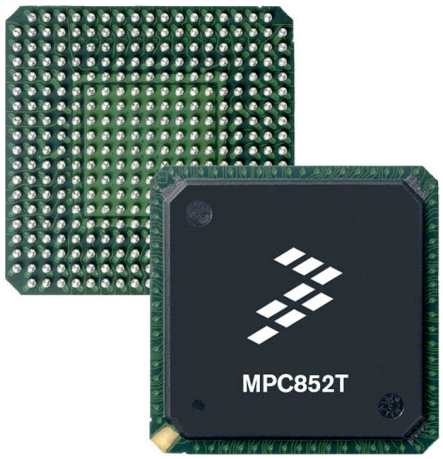
"int año = 2, número 5" /* JULIO 2009 /*

µCONTROL

Electrónica en General Pics en Particular

PIC16F628A: programación en assembler

Nos iniciamos en la utilización de este lenguaje para aprovechar al máximo los recursos del micro



Seguridad en micros Freescale

¿Cómo proteger nuestro código? ¿Qué posibilidades existen de evitar el espionaje industrial?

Addon para PSI

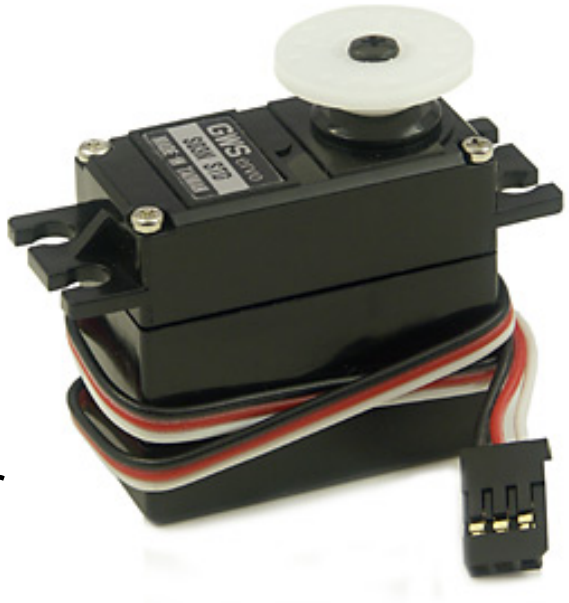
Programación de un módulo externo para PIC Simulator IDE

PIC BASIC [iv]

Seguimos programando PICs en lenguaje BASIC

Control de servos

Tutorial de manejo de un servomotor desde una computadora



El apagón analógico
Llegó el "día D" a los EE.UU.

Commodore Amiga
Un ordenador legendario

PIC Trainer 18
Módulo para el entrenador de PICs

Primer Congreso Virtual

Los Microcontroladores y sus Aplicaciones

14 al 30 de setiembre de 2009

Publique en este Congreso sus Aplicaciones realizadas con Microcontroladores
Participe del Congreso que unira a Universidades, Empresas y Desarrolladores Independientes

Áreas Temáticas

UTN

FICH - UNL

Facultad de Ciencia y Tecnología
Universidad Autónoma de Entre Ríos

Informes e Inscripciones

www.areacapacitacion.com.ar
e-mail: congreso.microcontroladores@gmail.com

Organiza:
Cátedras de Técnicas Digitales
FRP UTN

MICROCHIP freescale semiconductor ATMEL RABBIT ARM

http://www.areacapacitacion.com.ar

.staff

número = 5; año = 2;

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

Diseño y Diagramación:
Lucas Martín Treser
Argentina
lmtreser@gmail.com

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

Maximiliano Martín Simonazzi
Argentina
maxisimonazzi@gmail.com

Alejandro Casanova
Argentina
inf.pic.suky@live.com.ar

Diego Márquez García-Cuervo
España
diego@ucontrol.com.ar

Sergio Luis Scarnatto
Argentina
sergiols@keko.com.ar

Germán Reula
Argentina
gerreula@yahoo.com.ar

Descarga Gratuita.
Este contenido se rige por la licencia de Creative Commons "Licencia Creative Commons Atribución-No Comercial-Sin Obras Derivadas 3.0"

.indice

.editorial

Luego de casi un año sin publicarse, nuevamente tienes en tus manos un número de la **Revista uControl**. Varios motivos impidieron que pudiésemos cumplir en tiempo y forma con el lanzamiento de éste, el quinto número de nuestra publicación. En esos meses se han sumando una buena cantidad de colaboradores, y hemos recibido centenares de correos preguntando "¿cuándo vuelve a aparecer un numero de la revista?" Bien, la respuesta ya la conoces: a mediados de junio de 2009.

Fueron muchos meses de espera, pero creemos que no habrá sido en vano. A los antigüos colaboradores se han sumando nuevos, hemos cambiado un poco el diseño de la revista, estamos utilizando otro software para la edición de este documento, pero en el fondo, todo sigue igual: intentamos mostrar, de forma clara y ordenada, como puede hacerse algo interesante y útil con un puñado de componentes electrónicos.

Igual que en la "primera época" de la Revista uControl, hemos hecho lo posible para satisfacer a todos los lectores potenciales. Tenemos montajes completos, tutoriales sobre lenguajes de programación de microcontroladores, alguna que otra explicación sobre el funcionamiento de los componentes que utilizamos en nuestros proyectos y -al final de la revista- un poco de historia. De hecho, a partir de este número comenzarán a aparecer una serie de artículos sobre programación de PICs en ensamblador (del inglés assembler), algo que muchos habían reclamado insistentemente.

A largo de este tiempo hemos creado un foro, el **Foro uControl**, al cual se han sumado mas de 1200 amigos de la electrónica. En él hay material como para llenar varios números de la revista, y cada día se suma más gente con proyectos muy interesantes para compartirlos con la comunidad. De alguna manera, ellos son los dueños de uControl, y a ellos es a quien tienes que agradecerle el contar con este pequeño "PDF" cada dos meses.

La Revista uControl tiene un lugar propio dentro del Foro. La idea es que -luego de leer este ejemplar- puedas opinar allí y contarnos que te ha parecido, que cosas te gustaron y cuales no, que temas crees que habría que agregar, cuales quitar, etc. En definitiva, participar activamente en el contenido y el formato de la revista, incluso aportando tus propios trabajos, proyectos o tutoriales para que sean incluidos en números venideros.

De nuestra parte intentaremos hacer lo posible para satisfacer sus pedidos. Por lo pronto, la Revista uControl está nuevamente viva, y ahora puedes participar activamente de ella. Nos vemos dentro de un par de meses.

Foro uControl: http://www.ucontrol.com.ar/forosmf/index.php

PIC basic

cuarta parte

Continuamos con nuestro cursillo de programación de microcontroladores en lenguaje PIC BASIC del PIC SIMULATOR IDE. En esta entrega veremos como emplear este lenguaje para escribir en displays LCD alfanuméricos.

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

El manejo de los LCD en PIC BASIC se hace mediante el uso de varias sentencias del tipo “DEFINE”.

A grandes rasgos, y a pesar de la simplicidad que brinda el disponer de un mismo integrado especializado en casi todos los modelos de displays alfanuméricos (concretamente, el benemérito HITACHI HD44780), el envío de caracteres a una de estas pantallas de un microcontrolador es una tarea relativamente compleja. Esto se debe a que es necesario respetar protocolos de inicialización y tiempos a rajatabla, so pena de no obtener el resultado deseado. Todo esto hace bastante tediosa su programación en assembler.

Pero afortunadamente el dialecto de BASIC que estamos aprendiendo a utilizar dispone de un juego de instrucciones especiales para manejar displays. De hecho, y como veremos a continuación, podemos escribir en estos displays en dos modos diferentes: en modo “8 bits” y en modo "4 bits". Como es lógico, las instrucciones de alto nivel de BASIC nos evitan toda la complejidad y cantidad de instrucciones que tan bien dominan los buenos programadores en assembler.

.La importancia de DEFINE

El manejo de los LCD en PIC BASIC se hace mediante el uso de varias sentencias del tipo “DEFINE”. Básicamente, estas instrucciones se encargan de especificar al compilador a que pines del microcontrolador hemos conectado cada uno de los pines del LCD y en que formato de datos deseamos manejarlos.

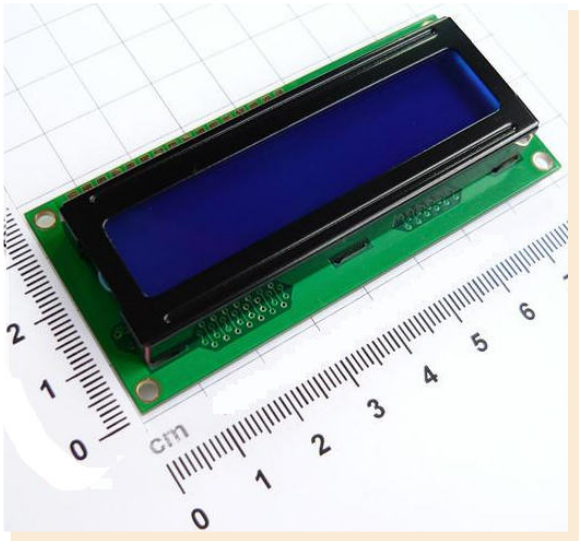
La forma genérica de la instrucción DEFINE es la siguiente:

```
DEFINE parámetro = valor
```

Donde “parámetro” es el nombre del parámetro al que le queremos asignar el “valor”. Existen

una buena cantidad de estos parámetros, cada uno con una función determinada. Los disponibles para el manejo de LCD alfanuméricos son los siguientes:

LCD_BITS: Define el número de bits de la interfaz de datos. Se pueden asignar valores de 4 u 8, siendo 4 el valor por defecto.
LCD_DREG: Define a que puerto del PIC tenemos conectado el port de datos del LCD. Los valores permitidos son PORTA, PORTB, PORTC, etc. Por defecto se asume PORTB.
LCD_DBIT: Define cual es el primer pin del puerto que usamos para enviar los datos al LCD cuando seleccionamos un bus de 4 bits. Solo puede ser el 0 (para los pines el 0, 1, 2 y 3) o 4 (para usar los pines 4, 5, 6 y 7). Por defecto se asume “4”, y esta instrucción se ignora para LCD_BITS = 8.
LCD_RSREG: Define a que puerto del PIC tenemos conectado el pin RS del LCD. Los valores permitidos son PORTA, PORTB, PORTC, etc. Por defecto se asume PORTB.
LCD_RSBIT: Define a que pin del puerto tenemos conectado el pin RS del LCD. Por defecto se asume “3”.
LCD_EREG: Define a que puerto del PIC tenemos conectado el pin E del LCD. Los valores permitidos son PORTA, PORTB, PORTC, etc. Por defecto se asume PORTB.
LCD_EBIT: Define a que pin del puerto tenemos conectado el pin E del LCD. Por defecto se asume “2”.
LCD_RWREG: Define a que puerto del PIC tenemos conectado el pin RW del LCD. Los valores permitidos son 0, PORTA, PORTB, PORTC, etc. Por defecto se asume “0”, que significa “no usamos el pin RW”.
LCD_RWBIT: Define a que pin del puerto tenemos conectado el pin RW del LCD. Por defecto se asume “0”, que significa “no usamos el pin RW”.
LCD_COMMANDUS: Define cuantos microsegundos demora la escritura de un comando en el display. Por defecto, este valor es de 5000. La mayoría de los LCD funcionan bien con un valor de 200 o incluso menor, así que conviene consultar su hoja de datos para colocar el valor correcto y hacer más rápidos nuestros programas.
LCD_DATAUS: Define cuantos microsegundos demora la escritura de un dato en el LCD. Por defecto, este valor es de 100.
LCD_INITMS: Define cuantos microsegundos demora la inicialización de la electrónica del LCD. Por defecto, este valor es de 100.



Display LCD "inteligente", incluye un circuito integrado que se encarga de mediar con el circuito de control

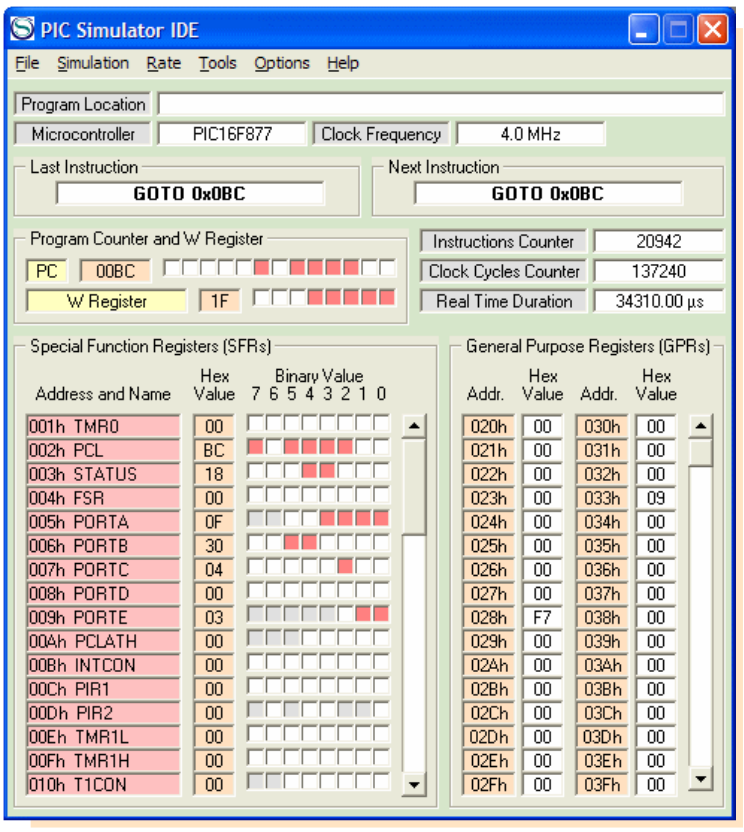
.Comandos

PIC BASIC del PIC SIMULATOR IDE dispone también de una serie de instrucciones que permiten el envío de comandos e instrucciones al display.

La primera de ellas es LCDINIT, destinada a inicializar la electrónica a bordo del display. Como es lógico, esta es la instrucción que debe utilizarse antes de enviar cualquier comando o dato al LCD. La forma de esta instrucción es la siguiente:

```
LCDINIT n
```

Donde “n” es el tipo de cursor que queremos que muestre el display. Enviar un “0” hará que el cursor permanezca oculto, un “1” significa que el cursor parpadeará, un “2” nos mostrará un cursor subrayado, y un “3” nos presentará un cursor subrayado y parpadeando.



Pantalla principal de PIC SIMULATOR IDE.

"Existen una buena cantidad de parámetros, cada uno con una función determinada"

La siguiente instrucción de este grupo es LCDCMDOUT. Esta se encarga de enviar comandos de control al LCD. Se emplea de la siguiente manera:

LCDCMDOUT comando

Donde "comando" puede ser alguno de los siguientes:

LcdClear: Borra completamente el contenido de la pantalla del LCD.
LcdHome: Lleva el cursor a la primera posición del primer renglón del LCD.
LcdLine2Home: Lleva el cursor a la primera posición del segundo renglón del LCD.
LcdLeft: Mueve el cursor una posición a la izquierda de la actual.
LcdRight: Mueve el cursor una posición a la derecha de la actual.
LcdShiftLeft: Desplaza el contenido del LCD una posición a la izquierda.
LcdShiftRight: Desplaza el contenido del LCD una posición a la derecha.
LcdLine1Clear: Borra la primera línea del LCD.
LcdLine2Clear: Borra la segunda línea del LCD.
LcdLine1Pos(x): Coloca el cursor en la posición "x" del primer renglón del LCD. "x" puede tener cualquier valor entre 1 y 40
LcdLine2Pos(x): Coloca el cursor en la posición "x" del segundo renglón del LCD. "X" puede tener cualquier valor entre 1 y 40.

.Envío de datos

LCDOUT envía datos al display. Por "datos" entendemos caracteres que el módulo LCD puede mostrar en su pantalla. Si son caracteres sueltos o cadenas de texto (incluidos símbolos especiales y dígitos), simplemente los ponemos entre comillas a continuación del comando. Si los que se trata de mostrar es el contenido de una variable, se debe escribir el nombre de la variable (precedida por "#") a continuación del comando. Si se necesitan imprimir varias variables, simplemente se separa el nombre de una y otra mediante una "coma".

A continuación veremos un par de ejemplos de cómo se utilizan todas estas instrucciones. El primero de ellos se encarga de mostrar un texto parpadeando en la

primera línea del display. Es un buen ejercicio recorrer el código expuesto mientras se intenta deducir como está conectado el LCD al PIC mirando las instrucciones "DEFINE" del principio del programa.

En el segundo ejemplo se muestra como imprimir el contenido de una variable ("A") en el LCD. Concretamente, se muestra un texto en el primer renglón, mientras que en el segundo se cuentan los números del 65535 al 0.

.Conclusión

Como hemos visto, utilizar un display de este tipo desde PIC BASIC es una tarea muy sencilla, y al alcance de todos los lectores de uControl.

En la siguiente entrega de este tutorial veremos como utilizar los "hermanos mayores" de estos LCD: los displays gráficos o GLCD, de 128x64 puntos.

¡Hasta la próxima!

"Es un buen ejercicio recorrer el código expuesto mientras se intenta deducir como está conectado el LCD al PIC"

```
DEFINE LCD_BITS = 8
DEFINE LCD_DREG = PORTB
DEFINE LCD_DBIT = 0
DEFINE LCD_RSREG = PORTD
DEFINE LCD_RSBIT = 1
DEFINE LCD_EREG = PORTD
DEFINE LCD_EBIT = 3
DEFINE LCD_RWREG = PORTD
DEFINE LCD_RWBIT = 2
'
LCDINIT 0 'inicializo el LCD sin cursor.
'
loop:
  LCDOUT "www.uControl.com" 'Muestra el texto…
  WAITMS 1000 'Espero un segundo
  LCDCMDOUT LcdClear 'Borro el display
  WAITMS 1000 'Espero un segundo
GOTO loop 'Vuelvo a loop: para repetir indefinidamente.
```

Ejemplo número 1.

```
DEFINE LCD_BITS = 8
DEFINE LCD_DREG = PORTB
DEFINE LCD_DBIT = 0
DEFINE LCD_RSREG = PORTD
DEFINE LCD_RSBIT = 1
DEFINE LCD_EREG = PORTD
DEFINE LCD_EBIT = 3
DEFINE LCD_RWREG = PORTD
DEFINE LCD_RWBIT = 2
'
DIM A AS WORD
A = 65535
'
LCDINIT 3 'Cursor parpadeando
WAITMS 1000
'
loop:
  LCDOUT "¡Estoy contando!" 'Texto del primer renglón
  LCDCMDOUT LcdLine2Home 'Paso al Segundo renglón
  LCDOUT #A 'Muestro el valor de A
  A = A - 1
  WAITMS 250
  LCDCMDOUT LcdClear 'Limpio del display
GOTO loop
```

Ejemplo número 2.

seguridad en micros freescale

Todos los que trabajamos con micros desde mucho tiempo siempre tuvimos un gran problema y era el hecho de que cualquiera podia clonar nuestro firmware. En este artículo veremos como la empresa Freescale nos brinda una solución muy práctica.

// por: Maximiliano Martín Simonazzi //
maxisimonazzi@gmail.com

Tener seguridad en un microcontrolador siempre fue algo bastante complejo. El único punto a favor que tenia el diseñador es que el que copiaba el firmware solo se llevaba el código máquina (también conocido como archivo hex) y eso hacia que el código sea casi imposible de modificar.

Sin embargo Freescale penso un poco en nosotros y nos entregó un buen sistema de seguridad que permite proteger los datos grabados en la memoria flash con una contraseña de 8 bytes (no es demasiado pero nos da una protección de 64 bits casi irrompible por métodos convencionales).

Esta clave se debe introducir cada vez que se quiera ingresar en el modo monitor luego de un POR (Power On Reset). El modo monitor es el que nos permite ver y modificar el contenido de la memoria flash. Una vez que se chequea la clave, si esta es erronea, se permite el ingreso al modo monitor pero solo se puede realizar un borrado total de la Flash y si se intenta leer el contenido, siempre vamos a obtener como resultado $AD. Por el contrario, si la clave es correcta, podremos ver, grabar y borrar todo el bloque de memoria Flash. Para verificar si el código ingresado es el correcto, solo basta con dirigirnos a verificar el bit 6 de la dirección de memoria RAM $40, si esta seteado (o sea tiene un valor 1) entonces el código fue ingresado correctamente y podemos acceder a la Flash.

A continuación explicare de una manera sencilla como utilizar este método y evaluaremos que tan segura es esta protección.

Para almacenar la clave se utilizan 8 posiciones de memoria, y esas son desde FFF6 a FFFD. Estas direcciones coinciden con los 4 vectores anteriores al reset. Depende de la fami-

lia estos pueden estar implementados o no. Para tomar un ejemplo:

68HC908GP32:

FFF6/FFF7 = Vector CH0 timer 1
FFF8/FFF9 = Vector PLL
FFFA/FFFB = Vector IRQ
FFFC/FFFD = Vector SW1

68HC908JL3:

FFF6/FFF7 = Vector CH0 timer
FFF8/FFF9 = Libre
FFFA/FFFB = Vector IRQ
FFFC/FFFD = Vector SW1

Pero, ¿por qué usar los vectores como clave? Esta idea surge de considerar que, difícilmente, dos programas coincidan en todos los vectores, por ende, no hay que destinar 8 bytes específicos para la protección. Pero esto tiene un efecto secundario y es que a medida que el programa se modifica, los vectores pueden cambiar de posición, y por lo tanto también la clave.

Para cada uno de los 4 vectores existen dos opciones:

Vector en uso: Si utiliza el vector, el valor debe estar dentro del rango de la memoria FLASH y apuntar al código de la interrupción correspondiente. Esto restringe el rango de valores posibles y permite a un atacante barrer un menor número de posibilidades para descubrir la clave correcta. Lo mejor en este caso es no dejar todos los vectores en valores muy próximos, sino distribuirlos a lo largo de toda la FLASH. Esto puede realizarse mezclando las rutinas a lo largo del programa (si tiene la FLASH muy comprometida de espacio) o forzando posiciones "raras" mediante el comando "ORG" en lugares vacíos de la FLASH.

Vector libre: Si tiene la precaución de deshabilitar la causa de interrupción o el procesador no la implementa, puede poner el valor que desee en los dos bytes correspondientes al vector con "DW $xxxx".

Tenga presente que la IRQ por defecto está activa en el reset y debe deshabilitarse con el comando mov #2, INTSCR. NUNCA deje los vectores de la clave que no usa en $FF.

"Para almacenar la clave en el microcontrolador se utilizan 8 posiciones de la memoria"

```
01. ORG $FFDC
02. dw  VECTOR_VACIO    ;VECTOR_TIMEBASE
03. dw  VECTOR_VACIO    ;VECTOR_ADC_COMPLETE
04. dw  VECTOR_VACIO    ;VECTOR_KEYBOARD
05. .......
06. dw  VECTOR_TIM1_CH1 ;
07. ;--------------------------- CLAVE
    ----------------------------------
08. dw  VECTOR_TIM1_CH0 ; <- valor fijado por el vector
09. dw  $9D5B           ;VECTOR_PLL << no se usan:
10. dw  $4722           ;VECTOR_IRQ <<  uso cualquier valor de
    clave
11. dw  $1254           ;VECTOR_SWI <<
12. ;-----------------------------------------------------------
13. dw  VECTOR_RESET
```

Ejemplo #1. 68HC908GP32 utilizando los vectores: RESET, TIMER1 CH0 y TIMER1 CH1. VECTOR_VACIO apunta a una instrucción RTI. Si VECTOR_TIM1_CH0=8523, la clave será: 85 23 9D 5B 47 22 12 54.

```
01. ORG $FFDC
02. dw  VECTOR_VACIO    ;VECTOR_TIMEBASE
03. dw  VECTOR_VACIO    ;VECTOR_ADC_COMPLETE
04. dw  VECTOR_VACIO    ;VECTOR_KEYBOARD
05. .......
06. dw  VECTOR_TIM1_CH1
07. ;--------------------------- CLAVE
    ----------------------------------
08. dw  VECTOR_TIM1_CH0 ; <- valor fijado por el vector
09. dw  VECTOR_PLL_SEG  ; <- se "desvia" el vector
10. dw  VECTOR_IRQ_SEG  ; <- se "desvia" el vector
11. dw  VECTOR_SWI      ; <- valor fijado por el vector
12. ;-----------------------------------------------------------
13. dw  VECTOR_RESET
14.
15. ;Opción 1:
16. ;--- se desvían los vectores a lugares no utilizados de
    memoria ---
17. ORG $EDC3           ;usar lugares libres de memoria!!
18. VECTOR_IRQ_SEGUR:
19. jmp    VECTOR_IRQ   ;salto al verdadero vector
20. ORG $F17A           ;usar lugares libres de memoria!!
21. VECTOR_PLL_SEGUR:    jmp    VECTOR_PLL   ;salto al verdadero
    vector
22.
23. ;Opción 2:
24. ;--- se mueve el código directamente ---
25. ORG $F729           ;usar lugares libres de memoria!!
26. VECTOR_SWI:
27. ....
28. RTI
```

Ejemplo #2. 68HC908GP32 utilizando los vectores: RESET, TIMER1 CH0, TIMER1 CH1, PLL, IRQ y SWI. Si VECTOR_TIM1_CH0=8015, la clave será: 80 15 ED C3 F1 7A F7 29.

```
ORG $FFDE
dw  VECTOR_VACIO    ;VECTOR_ADC_COMPLETE
dw  VECTOR_VACIO    ;VECTOR_KEYBOARD
.......
dw  VECTOR_TIM_CH1  ;USO CH1 y dejo CH0 libre para la clave
;---------------------------- CLAVE
 ---------------------------------
dw  $D17F           ;no se usa (elijo valor)
dw  $2673           ;NO TIENE PLL (elijo valor)
dw  VECTOR_IRQ      ; <- valor fijado por el vector
dw  $1F9B           ;no se usa (elijo valor)
;-----------------------------------------------------------------
dw  VECTOR_RESET
```

Ejemplo #3. 68HC908JL3 utilizando los vectores: RESET, TIMER CH1, IRQ. En este caso elijo usar el canal 1 del timer para dejar libre el vector del canal 0. Si VECTOR_IRQ=ED5F, la clave será: D1 7F 26 73 ED 5F 1F 9B.

. ¿Es seguro este método de protección?

Al tener un código de seguridad de 64-bits, la máxima cantidad de combinaciones es de 1.8×10^18. Este número se obtiene al multiplicar la cantidad de combinaciones de cada vector por 4, las cuales dependen de si se utilizan o puede usarse cualquier valor, dado que en el primer caso la cantidad de combinaciones es igual al tamaño en bytes de la FLASH y en el segundo es 65536 (todos los valores posibles con 2 bytes). Esta diferencia es más importante en los procesadores con muy poca FLASH. Entonces:

NCombTotal= NCombV1 x NCombV2 x CombV3 x CombV4

Este es un número muy grande, por lo cual puede suponerse que es muy difícil romper este código. El punto clave será entonces que tan rápido se puede probar una por una hasta cubrir todas las posibles combinaciones. La única forma de probar una clave es luego de un POR o sea, hay que quitarle alimentación al microcontrolador y esperar un tiempo antes de conectarlo nuevamente para probar otra clave. El envío de la clave también demora un tiempo, ya que debe enviarse en forma serial (excepto en el modo paralelo del GP32) que a 9600 baudios equivale a 8.33 mS.

.Ejemplos mas concretos

En el mejor caso todos los vectores están libres:

NCombTotal= 655364 = 1.8 × 10^19

Peor caso del 68HC908GP-32: NCombTotal= 322564 = 1.1 × 10^18

Peor caso del 68HC908JK1: NCombTotal= 65536*15363 = 2.4 × 10^14 (ya que al no tener PLL uno de los vectores siempre está libre).

Lo cual traducido a tiempo da (considerando que cada clave se puede probar en 1 milisegundo y en la mitad de las pruebas encuentran el valor correcto):

Mejor caso: 292 millones de años.

Peor caso del 68HC908GP-32: 17 millones de años.

Peor caso del 68HC908JK1: 3765 años.

```
;================================================
;== inicializa / lee modo de lectura ==
FF1C    MOV     #FF,$40         ;[$40].6=1  ==> por ahora
 clave correcta
;[$40].7=1  ==> leer bytes PARALELO (PORTA)
FF20    LDHX    #FFF6           ;primer byte de la clave
FF23    BRSET   7,PORTA,FF28    ;(PA.7 == 0)? ==> leer SERIE
FF26    BCLR    7,$40           ;[$40].7=0  ==> leer bytes
 SERIE (PA.0)
;== loop para clave ok hasta el momento ==
FF28    CBEQX   #FE,FF45        ;leyó los 8 bytes?
FF2B    LDA     PORTA           ;lee PORTA (por si es modo
 paralelo)
FF2D    BRSET   7,$40,FF33      ;modo paralelo? => ya leyó el
 byte
FF30    JSR     FE97            ;modo serie => lee un byte
 desde PA.0
FF33    NOP                     ;ajusta delay "NOP+CBEQ X+"=
 1+4= 5 ciclos
FF34    CBEQ    X+,FF28         ;compara con el vector, igual
 => ver próximo
;distinto => error, seguir leyendo
;== loop para clave incorrecta ==
FF36    CBEQX   #FE,FF4B        ;repite lectura paralelo o
 serie del
FF39    LDA     PORTA           ;bloque anterior pero no
 compara porque ya
FF3B    BRSET   7,$40,FF41      ;sabe que hubo error en la
 clave.
FF3E    JSR     FE97            ;
FF41    AIX     #1              ;delay "AIX+BRA"= 2+3= 5
 ciclos
FF43    BRA     FF36            ;
FF45    LDX     #FF             ;=== CLAVE CORRECTA ===
 [$40].6=1
FF47    RSP                     ;
FF48    JMP     FE20            ;entrada que permite leer la
 FLASH
FF4B    BCLR    6,$40           ;=== CLAVE INCORRECTA ===
 [$40].6=0
FF4D    LDX     #FF             ;
FF4F    RSP                     ;
FF50    JMP     FE21            ;entrada que NO PERMITE leer
 la FLASH
;================================================================
```

Ejemplo #4. Implementación del algoritmo de verificación de la clave en la ROM Monitor del 68HC908GP32.

Este análisis es válido dado que no se conoce el resultado hasta no haber ingresado los 8 bytes. Si se pudiera saber si cada byte es correcto o no sin esperar al final, bastaría con probar 128 x 8 veces = 1024 pruebas y obtener la clave correcta en 1 segundo!.

Siguiendo con el ejemplo #4, en la dirección FF33 se puede ver la instrucción NOP insertada para que se produzca el mismo delay ante clave correcta e incorrecta en ambos lazos, no permitiendo inferir desde fuera el resultado del testeo. Sin esta instrucción podría medirse el tiempo desde los bytes enviados y el BREAK que genera el procesador al finalizar la verificación y obtener la clave en pocos segundos

.Ingreso de la clave en "paralelo" en el micro 68HC908GP32

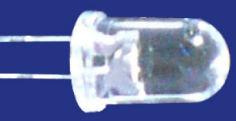
En el listado de la ROM puede verse que el modo paralelo permite acelerar el ingreso de la clave vacía (todos $FF) poniendo 8 resistores de "pull-up" en el PORTA. Utilizar este modo para ingresar otra clave es muy complicado (aunque posible) dado que en la dirección FF23 se lee PA7 para ver si es serie o paralelo y en FF2B se lee el primer valor desde el PORT, no existiendo una referencia de tiempo desde el exterior que permita saber cuando cambiar el valor del PORTA. Una forma posible es mediante un circuito de reset muy preciso, sin usar el PLL y determinando el delay de todas las instrucciones comprendidas entre cada lectura.

Una vez finalizada la clave, el procesador envía un BREAK y desde allí en más todos los comandos deben enviarse en forma serie.

Como conclusión, podemos decir que este método de protección de los datos es muy seguro y podemos estar tranquilos que nuestra información siempre va a estar segura dentro de un microcontrolador Freescale.

.Fuentes

Datasheet de micros 68HC908, notas de aplicación Freescale, nota de aplicación Ing. Dubatti e Ing. DiLella



¿Buscás LEDs?

D+
LED

Todos los LEDs, un solo lugar.

www.dled.com.ar

Soler 4949 Timbre "1" | Capital Federal | Argentina | Teléfono +54 11 4770 9366 y líneas rotativas.

módulo PIC Trainer 18

Se trata del segundo módulo de nuestro entrenador destinado a albergar un microcontrolador. En este caso, pueden utilizarse los PICs de 18 pines más populares, como el 16F84A, el 16F88, 16F627A, 16F628A, 16F818A y muchos más. Su construcción no presenta dificultades, y seguramente podrás aprender mucho sobre programación de PICs utilizándolo. ¡Manos a lo obra!

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

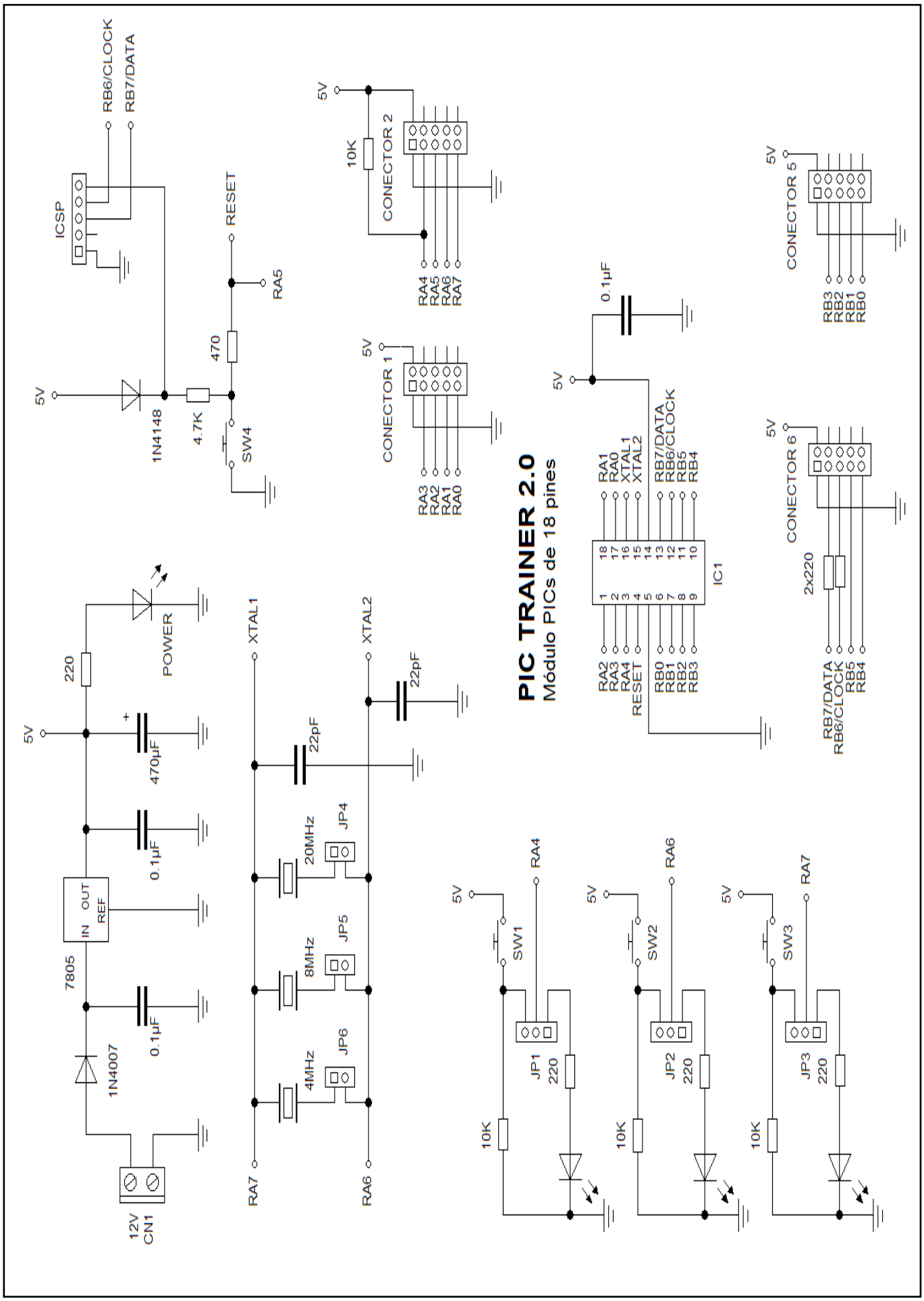
“Con este módulo podremos experimentar programando los PICs de 18 pines.”

Tal como ocurre con los microcontroladores de 40 pines, la empresa Microchip coloca los puertos de los micros de 18 pines casi siempre en la misma posición. Eso nos permite diseñar una placa “universal” que pueda utilizarse con una gran variedad de modelos. De hecho, no solo podremos hacer experimentos con los muy populares “16F” y 16C”, sino que también vamos a poder usar un buen surtido de chips de la familia “18F”. Solo debemos asegurarnos, consultando la hoja de datos del chip en cuestión, que los pines de alimentación, oscilador, etc., estén en una posición que los haga compatibles con la distribución de señales empleada en el entrenador.

.El circuito

Como puede verse en el diagrama que acompaña este artículo, el circuito de este módulo también tiene una gran sencillez. Hemos colocado una serie de conectores IDC10 (en los bordes del PCB) que permiten a los módulos periféricos acceder a cada pin de entrada / salida del microcontrolador que ocupe el zócalo central del entrenador.

La alimentación del módulo se ha resuelto mediante un regulador de voltaje integrado de la serie LM78xx, concretamente el modelo LM7805, y una bornera de dos tornillos permite alimentar a la placa desde una fuente externa de corriente continua. Esta debe ser capaz de entregar una tensión de entre 7.5 y 15V, lo mas estable posible. Un diodo 1N4007 se encarga de proteger el resto de la etapa de alimentación de una conexión con la polaridad invertida, y condensador electrolítico de 470uF/16V filtra el poco riple que pueda haber escapado al filtro de la fuente que estemos empleando.


PIC TRAINER 2.0
Módulo PICs de 18 pines
ICSP
RB6/CLOCK
RB7/DATA
RESET
RA5
470
4.7K
1N4148
SW4
5V
10K
CONECTOR 2
RA4
RA5
RA6
RA7
CONECTOR 1
RA3
RA2
RA1
RA0
0.1µF
RA1
RA0
XTAL1
XTAL2
RB7/DATA
RB6/CLOCK
RB5
RB4
IC1
RA2
RA3
RA4
RESET
RB0
RB1
RB2
RB3
CONECTOR 5
RB3
RB2
RB1
RB0
CONECTOR 6
2x220
RB7/DATA
RB6/CLOCK
RB5
RB4
220
POWER
470µF
0.1µF
7805
IN
OUT
REF
1N4007
12V
CN1
XTAL1
XTAL2
22pF
20MHz
JP4
8MHz
JP5
4MHz
JP6
RA7
RA6
SW1
SW2
SW3
RA4
RA6
RA7
JP1
JP2
JP3

El regulador de voltaje está dotado de los dos condensadores de 0.1uF de rigor, y un diodo LED, en serie con un resistor de 220V se enciende cuando el circuito está alimentado, para que sepamos que se encuentra en esta situación.

Tal como ocurría con la placa para microcontroladores de 40 pines, en lugar de utilizar un cristal como oscilador del PIC que está siendo empleado en el módulo, hemos colocado 3 de ellos. Una serie de jumpers (identificados como JP4, JP5 y JP6) se encargan de seleccionar el que se corresponda con la frecuencia de trabajo que requiera nuestro proyecto. Los dos condensadores de 22pF completan esta parte del circuito, y parecen funcionar bien con las frecuencias típicas utilizadas. En caso de utilizar PICs que funcionen a más de 20MHz, es posible que haya que cambiarlos por condensadores de 15pF o incluso algo menos.

Aunque parezca obvio, tenemos que recordar al lector que no debe colocar más de un jumper a la vez, ya que en ese caso el microcontrolador no funcionará.

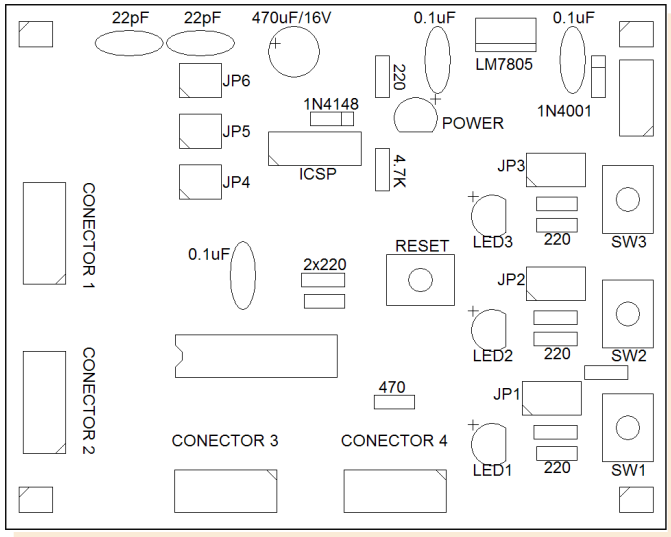
En cuanto a los cristales, hemos elegido (como puede verse en el esquema eléctrico) valores de 4MHz, 8MHz y 20MHz, pero nada impide utilizar otros. El lector puede cambiarlos a gusto.

En caso de utilizar algún modelo de PIC que disponga de oscilador interno y se quieran utilizar los pines 15 y 16 del mismo (correspondientes, en general, a A6 y A7) como pines de entrada / salida, bastará con no colocar ninguno de los jumpers mencionados.

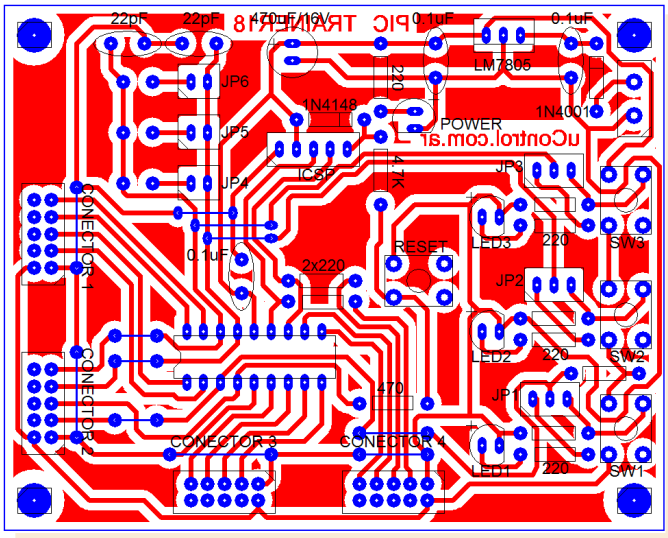
En el caso de configurar los pines 15 y 16 como entrada / salida, estos se comportarán de la misma manera que el pin 3, correspondiente al bit 4 del PORT-A. Esto habilita los LEDs y pulsadores incorporados en el módulo, permitiendo su uso como forma de ingresar (o representar) datos a (o de) nuestro programa. Los jumpers JP1, JP2 y JP3 permiten seleccionar si conectamos al PIC el LED o el pulsador.

En caso de seleccionar los pulsadores, debemos recordar que estos ponen el pin correspondiente a 5V cuando son presionados. Mientras que están en reposo, las entradas se mantienen a GND a través de sendos resistores de 10K.

Los conectores siguen las mismas normas que explicamos en el artículo principal de nuestra revista número 4, así que no deberías tener problemas a la hora de determinar la función de cada pin. Como regla general, recuerda que de los pines exteriores de cada conector solo se emplea uno (+V) y los otros cuatro están sin conectar. De lo cinco interior, uno corresponde a GND y los otros 4 a datos. No es mala idea tener a mano el gráfico con la función de cada pin cuando decidas hacer



Guia para el montaje de los componentes.



Hemos decidido utilizar un PCB de una sola cara.

algún programa para cargar en el entrenador.

La única excepción, o desvío de lo normal, que puedes ver en el diagrama de los conectores es en los pines correspondientes a RB6 (CLOCK) y RB7 (DATA), ya que poseen un resistor de 220 ohms en serie. Cumplen con la función de permitir programar el PIC mediante el conector ICSP sin necesidad de retirar el cable que conecta el entrenador con el módulo de turno. Por supuesto, si lo deseas puedes reemplazar esos dos resistores por sendos puentes, y a otra cosa. Solo deberás quitar el cable plano a la hora de reprogramar el PIC.

El pin 4, que corresponde al RESET (y al bit 5 del PORTA) en los microcontroladores PIC de 18 pines (al menos, en los que son compatibles con este entrenador), esta unido a un pulsador a través de un resistor de 470 ohms y a +V mediante otro de 4.7K y un diodo 1N4148. Al presionar el pulsador, el microcontrolador se resetea. Durante el funcionamiento normal del programa, el pin esta a +V. El diodo impide los problemas que podrían surgir entre las alimentaciones del módulo y del programador al utilizar el conector ICSP.

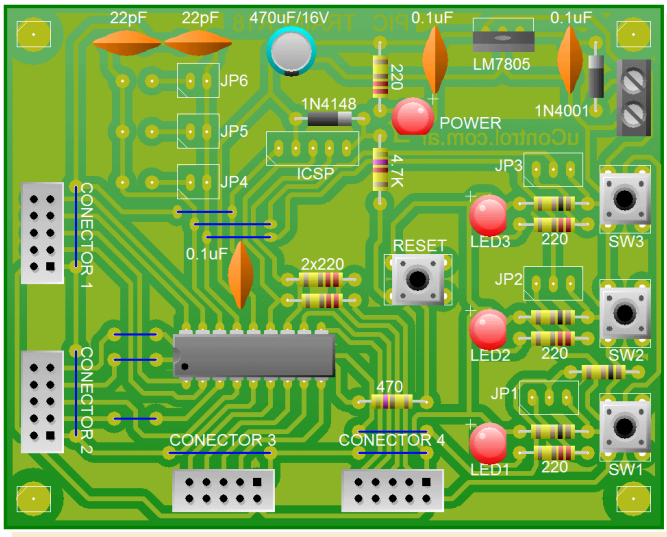
.Construcción

Si ya has montado alguno de nuestros proyectos, no tendrás ninguna dificultad a la hora de construír tu propio entrenador. Descarga el archivo PDF correspondiente al PC desde nuestra Web, y mediante la forma que más te guste (puedes usar el "método de la plancha" explicado en la revista número 1) transfiérelo a un trozo de PCB virgen. Luego, al baño de cloruro férrico; y por último, una buena limpieza y haces los agujeros.

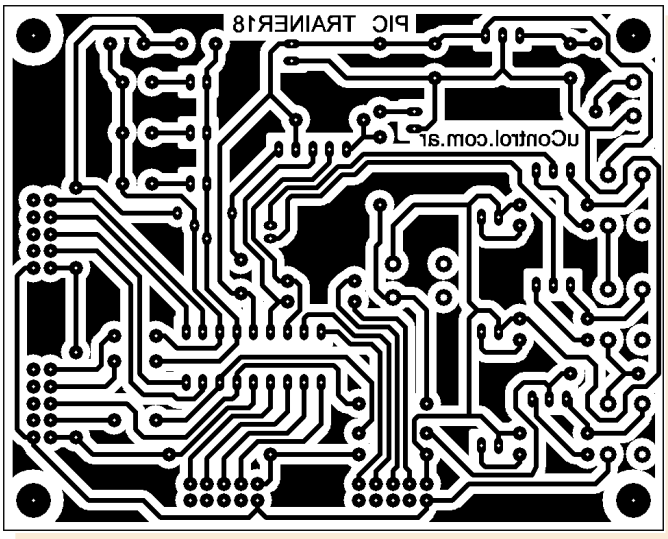
A la hora de soldar los componentes, como siempre, resulta más sencillo si primero vas colocando los que son más bajos, como los puentes, diodos, zócalos y resistores. Deja para el final los conectores, regulador de voltaje y condensadores. Asegúrate de que, involuntariamente, no haces un puente entre dos puntos del circuito.

Presta especial atención a la hora de soldar los componentes que tienen "polaridad", como los diodos, LEDs y condensadores electrolíticos. También es importante que coloques el zócalo destinado al PIC en la dirección correcta, ya que de hacerlo mal puedes confundirte cuando insertes el microcontrolador, dañándolo.

Una vez montado todo, sin colocar el PIC en su lugar, alimenta el circuito con una tensión de entre 7.5 y 12V. El LED "Power" debería encenderse. Si es así, verifica con un multímetro que la tensión entre los pines 5 (GND) y 14 (Vcc) del zócalo del microcontrolador sea de 5V. También puedes verificar que en los conectores de expansión este presente esa tensión. Si todo esta bien, ya



El aspecto, según el programa de diseño, que tendrá el módulo.



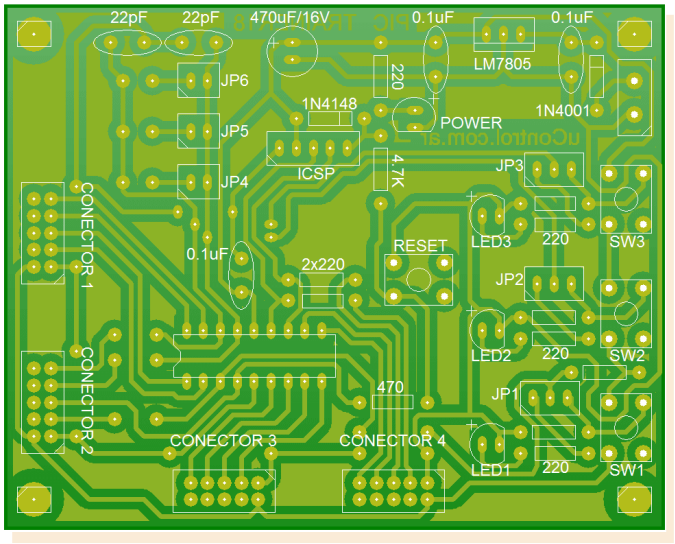
El PCB a utilizar, tan solo mide 98x80 milímetros.

tienes listo tu entrenador. Caso contrario, repasa las soldaduras y posición de los componentes.

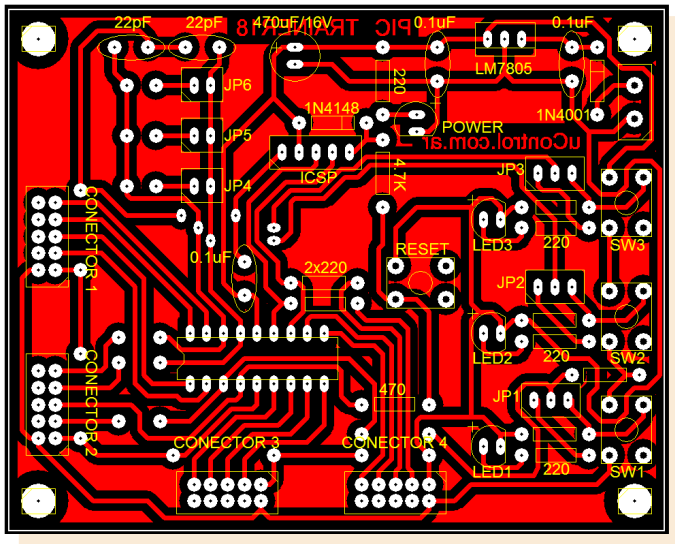
.Conclusión

Hemos montado otra placa muy importante de nuestro entrenador. Existen una gran cantidad de proyectos que pueden resolverse con PICs considerados “pequeños” o “medianos” como los que soporta este módulo, así que seguramente podrás pasar muchas horas aprendiendo con él. Recuerda que para mejorar sus posibilidades de entrada / salida, puedes construir el módulo de 8 entradas / salidas publicado en el número anterior.

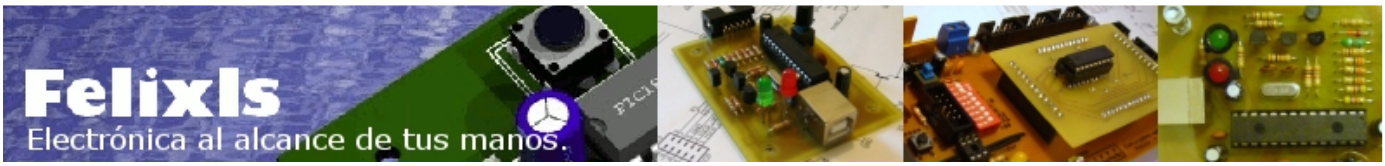
"Los jumpers JP1, JP2 y JP3 permiten seleccionar si conectamos el LED o el pulsador al PIC.”



Aspecto de la serigrafía del PCB a construir.



Otra vista del diseño del PCB.



resistores

Conocidos en algunos países como “resistencias”, los modestos resistores forman parte de prácticamente la totalidad de nuestros proyectos. El código de colores que se emplea para denotar su valor es una de las primeras cosas que aprende todo aficionado a la electrónica. En este pequeño artículo te contamos todo lo que necesitas saber para emplear correctamente este componente en tus circuitos.

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

“Los modestos resistores forman parte de prácticamente la totalidad de nuestros proyectos.”

Prácticamente no existen esquemas electrónicos en los que no se vean una o más resistencias. Estos componentes tienen como función distribuir adecuadamente las tensiones y corrientes que circulan por el circuito. Su funcionamiento se basa en la dificultad que ofrecen al paso de la corriente eléctrica algunos materiales, generalmente con valores de resistividad altos.

Para definir el valor de un resistor se utiliza como unidad el Ohm, que se representa por la letra griega omega (Ω), en honor a Georg Simon Ohm (1789-1854). Ohm fue un físico y matemático alemán que aportó a la teoría de la electricidad la Ley que relaciona la intensidad de una corriente eléctrica, su fuerza electromotriz y la resistencia. En 1827 formuló la ley, que lleva su nombre, y que establece que: $U = I \times R$.

.Los resistores como componentes

Si bien teóricamente es posible construir resistores de prácticamente cualquier valor, por una cuestión practica solo se las construye de una serie de valores perfectamente normalizados, y que combinados como veremos mas adelante, permiten lograr cualquier valor de resistencia que necesitemos para nuestro proyecto. Dichas series de valores se encuentran agrupadas en las llamadas “Familias E”.

Existen varias familias de valores posibles, con nombres como E6, E12, E24, etc., donde el número que acompaña a la E representa la cantidad de valores diferentes que componen la familia mencionada. A los valores base se los multiplica por 10, 100, 1.000, 10.000, 100.000 o 1.000.000 para obtener los valores de los resistores con resistencias más altas. En el

cuadro número 1 figuran los valores base de cada familia de resistencias. Las demás series, como la **E48** (2% de tolerancia), y las menos utilizadas **E96** y **E192** agregan valores intermedios a los mencionados, y tolerancias más pequeñas.

Para no tener la necesidad de escribir grandes cantidades de ceros al expresar valores de resistencias elevadas, se utilizan la letra **K** y **M**, que designan factores multiplicativos de 1.000 y 1.000.000. Si a un valor cualquiera de la tabla anterior, por ejemplo a 4,7 le agregamos la **K** obtenemos 4.7K que significan 4700 Ω. Si le añadimos la **M**, nos queda 4.7M que indica 4.700.000Ω. Muchas veces se utiliza la letra en lugar de la coma, por lo que 4.7K y 4K7 representan el mismo valor.

Cuando nos referimos a la "tolerancia" que tiene un resistor, estamos hablando de la máxima desviación del valor teórico que podemos esperar encontrar al medir su valor. Por ejemplo, una resistencia con un valor declarado de 1KΩ y una tolerancia del 5% tiene un valor real comprendido entre 950 Ω y 1050 Ω.

E6	1.0, 1.5, 2.2, 3.3, 4.7, 6.8	Tolerancia: 20%
E12	1.0, 1.2, 1.5, 1.8, 2.2, 2.7, 3.3, 3.9, 4.7, 5.6, 6.8, 8.2	Tolerancia: 10%
E24	1.0, 1.1, 1.2, 1.3, 1.5, 1.6, 1.8, 2.0, 2.2, 2.4, 2.7, 3.0, 3.3, 3.6, 3.9, 4.3, 4.7, 5.1, 5.6, 6.2, 6.8, 7.5, 8.2, 9.1	Tolerancia: 5%

.Código de colores

Físicamente, las resistencias más comunes consisten en un pequeño cilindro con dos terminales, uno en cada extremo, con anillos de colores sobre su cuerpo que representa el valor en ohms. Existen básicamente dos tipos de códigos, uno utiliza tres bandas y el otro cinco. En el código de cuatro bandas, los dos primeros anillos representan los dígitos que forman el valor base de la resistencia, el tercero el número de ceros que es necesario añadir, y el cuarto el valor de la tolerancia.

Por ejemplo, si tomamos una resistencia que tiene una banda marrón, una roja, una naranja y otra dorada, su valor será 12000 ohms, con el 5% de tolerancia, dado que según la tabla de colores el marrón representa el 1, el rojo un 2 y el naranja significa que se agregan tres ceros. Las resistencias con cinco bandas de colores se leen de la misma manera, pero teniendo en cuenta que las tres primeras son los dígitos que forman el valor base, la cuarta banda la cantidad de ceros a agregar y la quinta la tolerancia.

.Comportamiento en un circuito

Como decíamos antes, a partir de los valores disponibles en cada serie de resistencias es posible obtener prácticamente cualquier valor que deseemos, simplemente combinándolas de a

Color de la banda	Valor de la 1°cifra significativa	Valor de la 2°cifra significativa	Multiplicador	Tolerancia	Coeficiente de temperatura
Negro	-	0	1	-	-
Marrón	1	1	10	±1%	100ppm/°C
Rojo	2	2	100	±2%	50ppm/°C
Naranja	3	3	1 000	-	15ppm/°C
Amarillo	4	4	10 000	-	25ppm/°C
Verde	5	5	100 000	±0,5%	-
Azul	6	6	1 000 000	-	10ppm/°C
Violeta	7	7	-	-	5ppm/°C
Gris	8	8	-	-	-
Blanco	9	9	-	-	1ppm/°C
Dorado	-	-	0.1	±5%	-
Plateado	-	-	0.01	±10%	-
Ninguno	-	-	-	±20%	-

Código de colores.

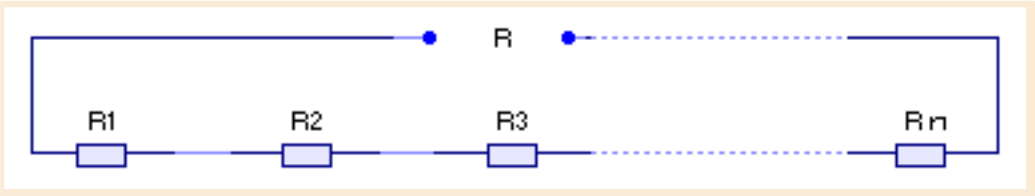


Figura 1. Resistores en serie.

"A partir de los valores disponibles en cada serie de resistencias es posible obtener prácticamente cualquier valor que deseemos."

dos o mas. Básicamente hay dos maneras de hacer esto, y se denominan agrupación en serie y agrupación en paralelo.

.Agrupación en serie

La agrupación en serie consiste en unir las resistencias una a continuación de la otra, como se ve en el esquema de la figura correspondiente. De esta manera, la corriente I que circula por ambas es la misma, mientras que, cada resistencia presenta una diferencia de potencial distinta entre sus extremos, que dependerá, según la ley de Ohm, de los valores de cada resistencia.

No es difícil jugar matemáticamente sumando los productos parciales de tensiones y corrientes para demostrar que la resistencia total de la agrupación de resistencias en serie es igual a la suma de las resistencias individuales.

En serie: R = R1+ R2+ R3+ ... + Rn

.Agrupación en paralelo

En el caso de la agrupación en paralelo, la conexión se efectúa como muestra la figura siguiente, donde se ve que los terminales se unen en dos puntos comunes llamados nodos. En este caso, por cada rama, compuesta por una resistencia, circula una corriente diferente, pero la tensión aplicada a todas es la misma. Nuevamente, trabajando matemáticamente con las corrientes y tensiones se puede demostrar que la resistencia equivalente de una asociación en paralelo es igual a la inversa de la suma de las inversas de cada una de las resistencias.

En paralelo: 1/R = 1R1+1/R2+1/R3+...+1/Rn

Hay dos casos particulares que debemos tener en cuenta. La resistencia equivalente a dos resistores en paralelo es **R=(R1xR2)/(R1+R2)**; y si todas las resistencias son iguales, **R=R/n**.

Por supuesto, nada impide asociar resistores de maneras que sean una combinación de las dos agrupaciones vistas. En esos casos, se dice que las resistencias presentan una asociación mixta, y para calcular el valor del resistor equivalente habrá que ir resolviendo el circuito por partes, en cada una de las cuales utilizaremos alguna de las fórmulas que vimos, según sea el caso.

En el caso del circuito de la figura 3, la resistencia total se calcularía sumando en primer lugar las agrupaciones en serie R1 y R2 por un lado, y R3 y R4 por otro, con lo que el circuito quedaría como una agrupación en paralelo de cuatro resistencias: R1+R2, R3+R4, R5 y R6. Utilizando la fórmula vista más arriba, podemos calcular el valor de la resistencia equivalente del circuito.

.Resistencias especiales

Además de las resistencias fijas que ya estudiamos, existen otras cuyo valor puede variar. Quizás las más comunes dentro de este grupo sean las llamadas potenciómetros o presets

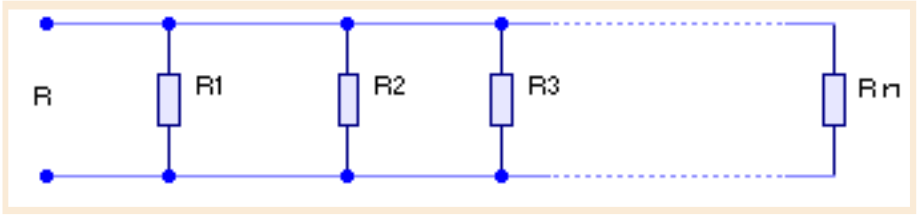
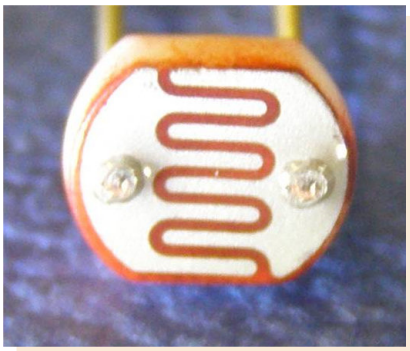


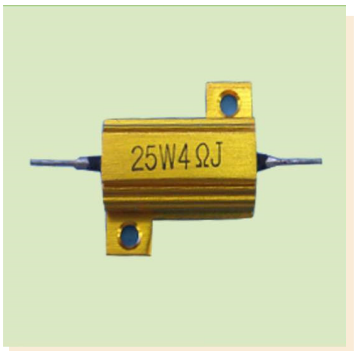
Figura 2. Resistores en paralelo.

que consisten en una pista de material resistivo por la que se desliza un cursor capaz de recorrerla de un extremo al otro al ser accionado por un mando externo. La resistencia del dispositivo se toma entre uno de los extremos y el cursor, por lo que su valor varia de acuerdo a la posición de este. En el caso de los potenciómetros, están construídas para que su valor se varíe con frecuencia, y se utilizan por ejemplo para controlar el volumen de un amplificador o la luminosidad de una lámpara. En el caso de los presets, la función es de ajuste, y se supone que solo se modificara su valor muy de vez en cuando, por lo que generalmente no disponen de un mando sino de un tornillo o ranura para ser accionadas con un destornillador. La forma en que varia la resistencia a medida que deslizamos el cursor puede ser lineal o logarítmica. En algunas aplicaciones, como el audio, se utilizan potenciómetros logarítmicos dado que se ajustan mejor a las características del oído humano.

También existen resistencias para usos especiales que varían su valor con la temperatura. Se fabrican de dos tipos, dependiendo si su resistencia aumenta o disminuye con la temperatura. Reciben el nombre de **NT**C y **PTC**, según tengan un coeficiente negativo (su valor disminuye al aumentar la temperatura) o positivo de temperatura.



Resistor dependiente de la luz (LDR).



Resistor de 25W.

Las **LDR** (Light Dependent Resistor, o Resistor Dependiente de la Luz) son, como su nombre lo indica, resistores cuyo valor varia de acuerdo al nivel de luz al que están expuestas. Los valores extremos que adopta una LDR cuando esta en total oscuridad o expuesta a plena luz varían de un modelo a otro, y se sitúan en el rango de los 50Ω a 1000Ω (1K) cuando están iluminadas con luz solar y valores comprendidos entre 50.000Ω (50K) y y varios megohmios (millones de ohms) cuando está a oscuras.

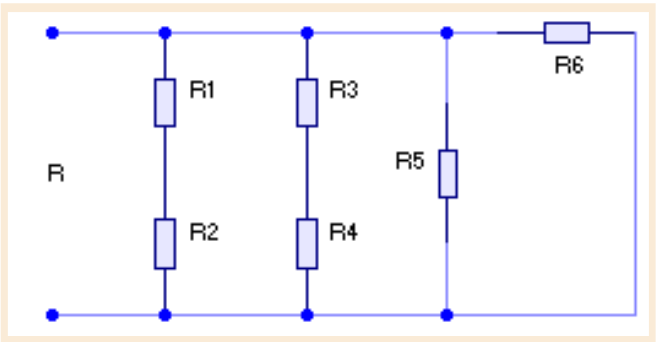


Figura 3. Es posible la asociación de resistencias en formas más complejas.

.Potencia

Por último, al momento de seleccionar una u otra resistencia en nuestros proyectos debemos considerar la potencia máxima para la que fue construída. En efecto, la caída de tensión que se produce cuando la corriente atraviesa la resistencia se transforma en calor, y el componente elegido debe ser capaz de soportarlo sin destruirse. Para potencias pequeñas, de 1/8 de Watt a 1 Watt suelen ser fabricadas a partir de una barra de carbón, pero las que son capaces de disipar potencias mayores se construyen arrollando un hilo resistivo sobre un cilindro metálico, todo cubierto por un esmalte vitrificado. Este tipo de resistencia pueden llegar a disipar hasta 100W, y a menudo es necesario algún tipo de mecanismo para proveer la ventilación adecuada.

el PIC16F628A en assembler

primera parte

Aquí se presenta un tutorial del PIC16F628A en el cual se da una pequeña introducción al microcontrolador y luego iremos a lo interesante, la programación. Se comenzará desde cero hasta abarcar cada módulo que tiene, siempre con un ejemplo sencillo de aplicación. La idea es ir adquiriendo conceptos a medida que los utilizaremos en el desarrollo de los ejemplos, de esta manera lo que se presenta teóricamente lo asociamos inmediatamente con la practica.

// por: Alejandro Casanova //
inf.pic.suky@live.com.ar

El PIC16F628A es un microcontrolador de 8 bits de Microchip. Pertenece a la familia de rango medio, es muy económico y dispone de varios periféricos integrados que nos permitirán realizar una gran variedad de aplicaciones. Cuenta con un set de instrucciones reducido, tan solo 35, lo que nos facilitará su programación.

"El micro PIC16F628A pertenece a la familia de rango medio de Microchip"

.Puertos

El PIC16F628 cuenta con dos puertos A y B, algunos pines de estos puertos de entrada/salida son multiplexados con una función alternativa de los periféricos del dispositivo. Cuando un periférico es activado el pin no puede ser usado para propósitos generales de entrada/salida.

El PUERTO A es un pu-

Características principales

- Conjunto reducido de instrucciones (RISC).
- Oscilador interno de 4MHz.
- Opera con una frecuencia de reloj externa de hasta 20 MHz (ciclo de máquina de 200ns).
- Memoria de programa: 2048 locaciones de 14 bits.
- Memoria de datos: RAM de 224 bytes (8 bits por registro).
- Memoria EEPROM: 128 bytes (8 bits por registro).
- Stack de 8 niveles.
- 16 Terminales de I/O que soportan corrientes de hasta 25 mA.
- 3 Temporizadores.
- Módulo de comunicación serie (USART).
- Módulo CCP (Captura/Comparación/PWM).
- 2 comparadores analógicos, 1 referencia de voltaje programable.

erto de entrada de 8 bits. Todos los pines, excepto **RA5**, pueden ser configurados como entrada o salida con la respectiva configuración del registro TRISA. El pin **RA4** esta multiplexado con la entrada de reloj **T0CKI** y como salida se comporta como colector abierto, por lo tanto debemos poner una resistencia pull-up a Vdd. El pin **RA5** es un disparador Schmitt solo de entrada y no cuenta con controladores de salida, según la configuración puede ser usado como **MCLR** (reset externo), y además sirve también para entrar en el modo de programación cuando se aplica una tensión igual a Vpp (13,4V mínimo). Los demás pines del puerto trabajan de entrada como disparador de Schmitt Trigger y como salida lógica CMOS. Los pines **RA0-RA3** sirven de entrada para los comparadores analógicos y por defecto vienen asociados a ellos, así que para usarlos como I/O digital deben ser previamente configurados. Los pines **RA6** y **RA7** cuando no se utiliza oscilador externo se usan para entrada externa de reloj y salida de oscilador, dependiendo la configuración que se use.

El **PUERTO B** es un puerto bidireccional de 8 bits, del cual por software se pueden habilitar resistencias de pull-up internas. El PUERTO B es multiplexado con interrupciones externas, tales como detección de flanco por **RB0**, cambio de nivel por **RB4** a **RB7**, módulo **USART**,

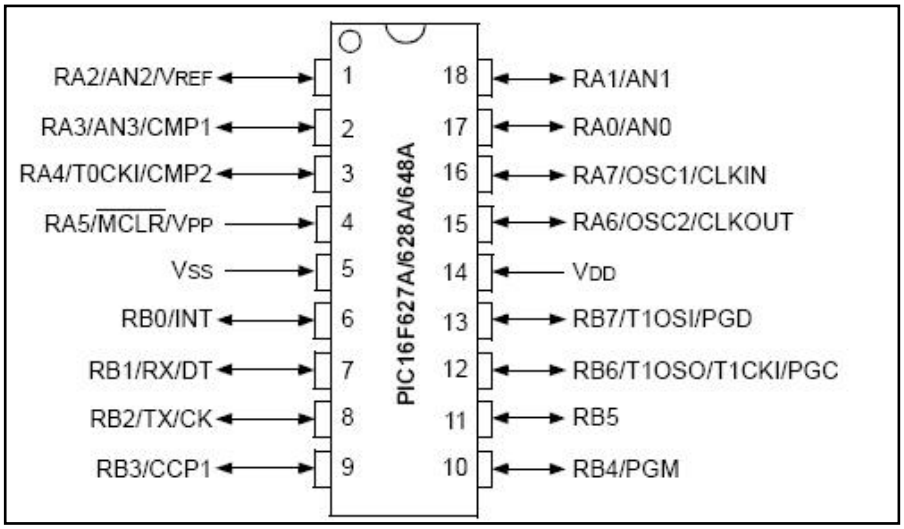


Figura 1. Diagrama de pines.

el módulo **CCP** y el reloj de entrada/salida **TMR1**. Los demás pines son: **VDD**, pin de alimentación positiva (de 2 a 5,5 Vcc) y **VSS**, pin de alimentación negativa.

.Estructura interna del microcontrolador

La arquitectura que utiliza el PIC es la Harvard, esta dispone de dos memorias independientes, una que contiene solo instrucciones (memoria de programa) y la otra solo contiene datos (memoria RAM). Ambas disponen de sus respectivos buses de acceso y es posible realizar operaciones de acceso simultáneamente en ambas.

. Memoria de programa

El PIC16F628A posee un contador de programa de 13 bits, capaz de direccionar un espacio de memoria de 8Kx14. Sin embargo, únicamente los primeros 2Kx14, desde 0000h hasta 07FFh, están implementados. Los vectores de reset e interrupción están en las direcciones 0000h y 0004h, respectivamente. La pila (stack) es de 8 niveles, lo cual significa que puede soportar hasta 8 direcciones de retorno de subrutina.

.Memoria RAM

El PIC16F628A posee un espacio de memoria RAM

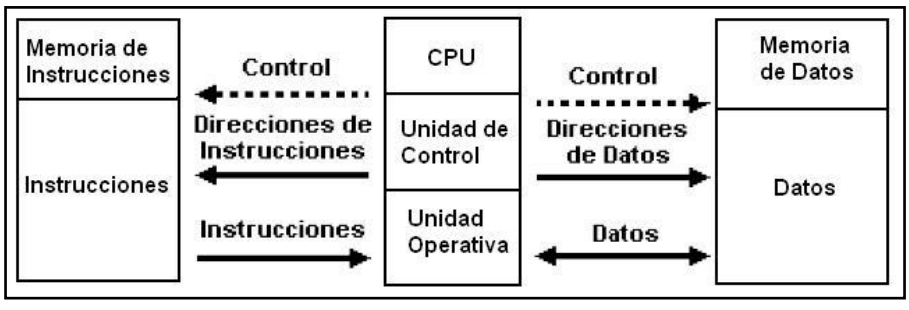


Figura 2. Arquitectura del PIC.

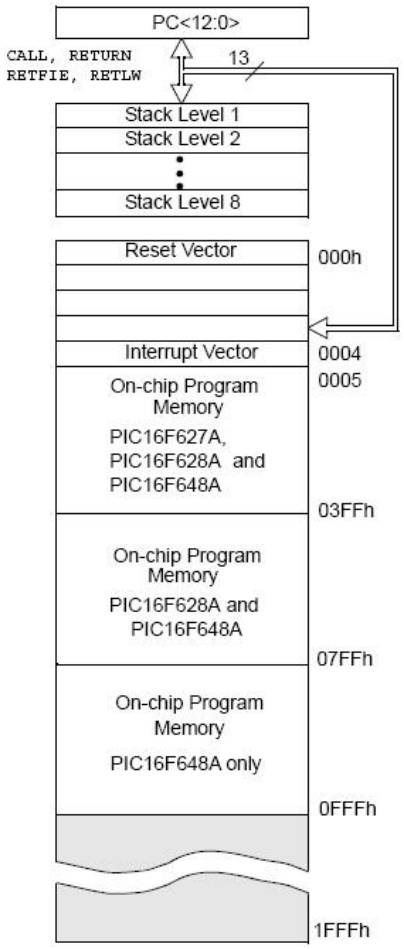


Figura 3. Memoria de programa.

de datos de 512x8, dividido en 4 bancos de 128 bytes cada uno. Sin embargo, sólo están implementados 330 bytes, correspondiendo 224 al área de los registros de propósito general (GPR) y 36 al área de los registros de función especial (SFR). Los restantes 70 bytes implementados son espejos de algunos SFR de uso frecuente, así como de los últimos 16 GPR del banco 0. Por ejemplo, las posiciones 0Bh, 8Bh, 10Bh y 18Bh corresponden al registro INTCON, de modo que una operación hecha en cualquiera de ellos, se refleja automáticamente en los otros. Se dice, entonces, que las posiciones 8Bh, 10Bh y 18Bh están mapeadas en la posición 0Bh. Esta característica agiliza el acceso a estos registros, puesto que no siempre es necesario especificar el banco donde se encuentran. La selección del banco de ubicación de un SFR o un GPR particular se hace mediante los bits 6 (RP1) y 5 (RP0) del registro STATUS.

.Configuración de fusibles

El PIC16F628 ha sido construído con características tales que se puede configurar para funcionar en modos de operación que no necesitan componentes externos tales como el circuito de reloj o de reset. Esto implica que es necesario configurar su modo de operación a través de una palabra de configuración.

La palabra de configuración se encuentra mapeada en la dirección 2007h de la memoria de programa y solo puede ser accesada durante la programación de dispositivo.

.Circuito reset externo

Los microcontroladores disponen de un pin destinado para ejecutar un RESET en el caso de una falla o cuando sea necesario aplicarle un RESET. En el caso del **PIC16F628A** debe estar habilitado por fuse **MCLRE**. Esta entrada está negada, por lo cual tendremos que conectarlo a la alimentación positiva **VCC** si queremos que nuestro PIC funcione. Una forma de tener control sobre el RESET es utilizar el circuito de la figura #7.

Palabra de configuración:

CP1:CP0: Bits de protección de código. Los bits 13-10 encargados de proteger la memoria de programa.
CPD: Bit de protección para código de datos.
1 = Protección deshabilitada de la memoria de datos.
0 = Protección habilitada en la memoria de datos.
LVP: Habilitación de la programación por voltaje bajo.
1 = LVP habilitado, la terminal RB4/PGM tiene tal función.
0 = LVP: deshabilitado, RB4/PGM es una terminal I/O.
BODEN: Bit de reset por voltaje de alimentación bajo.
1 = Reset por BOD habilitado
0 = Reset por BOD deshabilitado
MCLRE: Habilitación del terminal de reset.
1 = Terminal de reset en RA5.
0 = MCLR conectado internamente a Vdd, RA5 es un pin I/O.
PWRTEN: Bit de habilitación de temporizador al energizar.
1 = PWRT habilitado.
0 = PWRT deshabilitado.
WDTEN: Bits de habilitación de Watch-Dog.
1 = WDT habilitado.
0 = WDT deshabilitado.
FOSC2:FOSC1:FOSC0: Selección del tipo de oscilador.

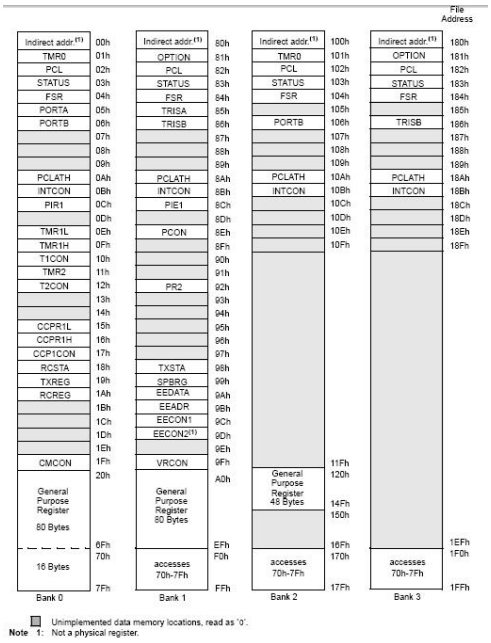


Figura 4. Memoria RAM estática.

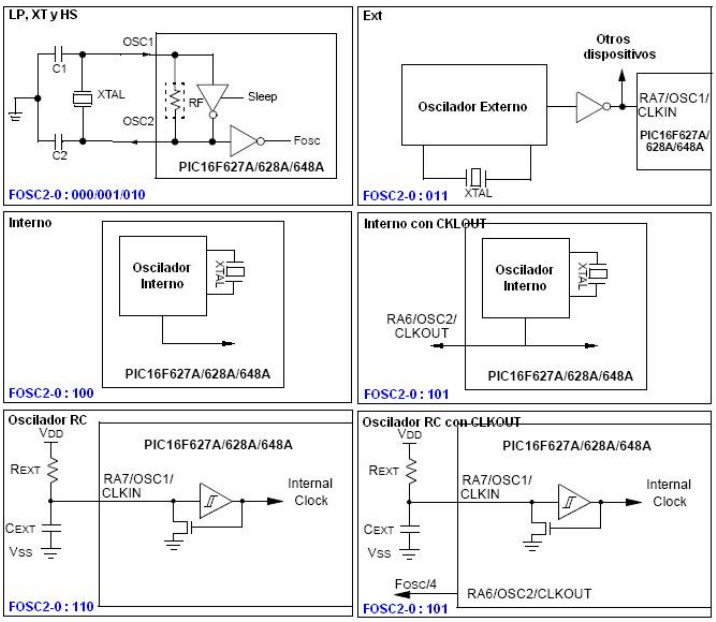


Figura 6. Configuración de osciladores.

.Comenzando con la creación del código

A continuación vamos a desarrollar nuestro primer programa. Este activará un LED conectado a RB0 siempre que el interruptor conectado a RA0 este cerrado. Para ello vamos a necesitar el circuito de la figura #8.

En **RA0** tenemos conectado un pulsador de forma que cuando lo pulsemos se introduzca un cero lógico en el pin y cuando no lo pulsemos se introduzca un uno lógico. Tenemos un LED con su correspondiente resistencia limitadora de corriente en el pin **RB0**.

Primero que nada debemos especificar con que microcontrolador estamos trabajando, esto lo realizamos en las dos primeras líneas:

```
; **** Encabezado ****
list p=16F628A
#include P16F628A.inc
```

En el archivo **P16F628A.inc** se encuentran las definiciones de las direcciones de los registros específicos, los bits utilizados en cada registro y los fusibles del microcontrolador.

.Configuración de fusibles

Hay ciertos aspectos del PIC que han de ser activados o desactivados mediante hardware a la hora de programarlo. Esto quiere decir que no se pueden volver a cambiar hasta que el chip no se reprograme de nuevo. En este ejemplo usamos, CP deshabilitada, Watchdog apagado, Boden habilitado, Power-on habilitado, Oscilador interno, CP de data deshabilitada, LVP deshabilitado y MCLR habilitado.

```
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON
```

.Definición de variables que utilizaremos en nuestro proyecto

En este caso solo definiremos bits, por ejemplo LED y Pulsador.

Para organizar nuestro programa lo estructuraremos de la siguiente manera:

1- Nivel
2- Directiva
3- Operandos
4- Comentarios

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP1	CP0	CP1	CP0	-	CPD	LVP	BODEN	MCLRE	FOSC2	PWRTE	WDTE	FOSC1	FOSC0

Figura 5. Palabra de configuración.

```
;**** Definición de variables ****
Led             equ   0       ; Definimos Led como el bit 0 de un registro, en este caso PORTB.
Pulsador        equ   0       ; Definimos Pulsador como el bit 0, en este caso será para PORTA.
```

.Configuración de puertos

Para la configuración necesitamos los siguientes registros: **STATUS > 0x03; PORTA > 0x05; PORTB > 0x06; TRISA > 0x86; TRISB > 0x86 y CMCON > 0x1F**. Por defecto los puertos quedan configurados como entradas de datos y si se quiere cambiar hay que configurarlos. Esto se realiza con los registros **TRISA** y **TRISB**, teniendo en cuenta que si se asigna un cero (0) a un pin, quedará como salida y si se asigna un uno (1), quedará como entrada. Además en este microcontrolador debemos configurar los pines **RA-0** a **RA3**, que por defecto vienen asociados a los comparadores, esto se realiza con el registro **CMCON**.

En nuestro caso se necesita colocar **TRISA** igual a 11111 (o se puede dejar por defecto), 111 en **CMCON** (para todos los pines I/O digitales) y **TRISB** 11111110. Ahora bien, cuando el PIC arranca se encuentra en el banco 0, **TRISA** y **TRISB** se encuentran en el banco 1, entonces debemos cambiar de banco. Esto se realiza con el bit **RP0** del registro **STATUS**. Si este se pone un cero a **RP0**, estaremos en el banco 0. Si se coloca un uno, estaremos en el banco 1.

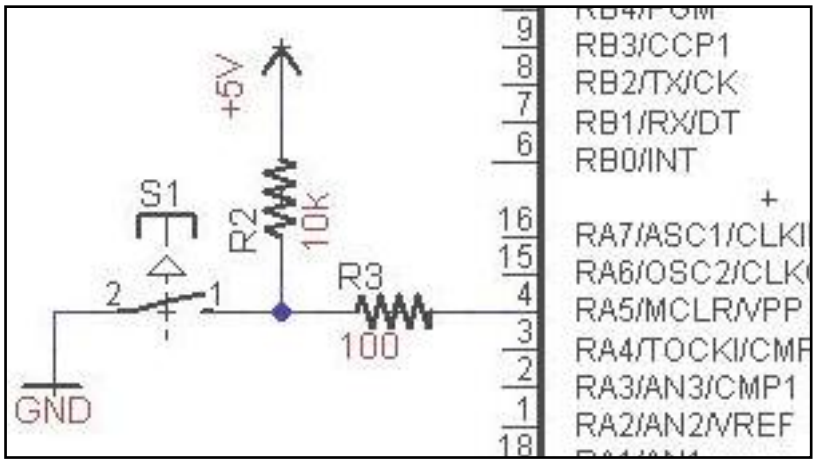


Figura 7. Circuito de reset.

.Registro de trabajo W

Es el registro más importante que tiene el microcontrolador y es denominado *acumulador*. Este registro almacena temporalmente uno de los datos que intervienen en la operación de la **Unidad lógica y Aritmética (ALU)**. ALU como indica su nombre, realiza las operaciones aritméticas y lógicas previstas en la colección de instrucciones del microcontrolador.

Ya configurado nues-

```
;**** Vector Reset ****
Reset
org     0x00                    ; Aquí comienza el micro.-
        goto    inicio          ; Salto a inicio de mi programa.-
        org     0x05            ; Origen del código de programa.-
;**** Programa Principal ****
Inicio
;**** Configuración de puertos ****
        movlw b'00000111'       ; Configuramos PORTA como I/O Digital.
        movwf CMCON             ; Movemos 111 a W, y W a CMCON.-
bsf     STATUS,RP0              ; Pasamos de Banco 0 a Banco 1.-
                                ; TRISA por default esta en 11111111.-
        movlw b'11111110'       ; Movemos 11111110 a W.-
        movwf TRISB             ; Movemos W a TRISB.-
        bcf     STATUS,RP0      ; Paso del Banco 1 al Banco 0
        bcf     PORTB,Led       ; Apago Led.-
```

“Definiendo los primeros parámetros antes de empezar con la sección principal de nuestro programa.”

tro PIC, vamos a realizar la rutina que ejecutará. Aquí solamente en un bucle infinito testeamos continuamente el estado del pulsador, y según su estado se encenderá o apagará el LED.

.Creación de demoras

Ciclo de máquina. Es la unidad básica de tiempo que utiliza el microcontrolador y equivale a 4 ciclos de reloj. Ósea, si tenemos un oscilador de 4 MHz, el ciclo de reloj (Tosc) seria de 250-ns y el ciclo de máquina (Tcy) de 1 us.

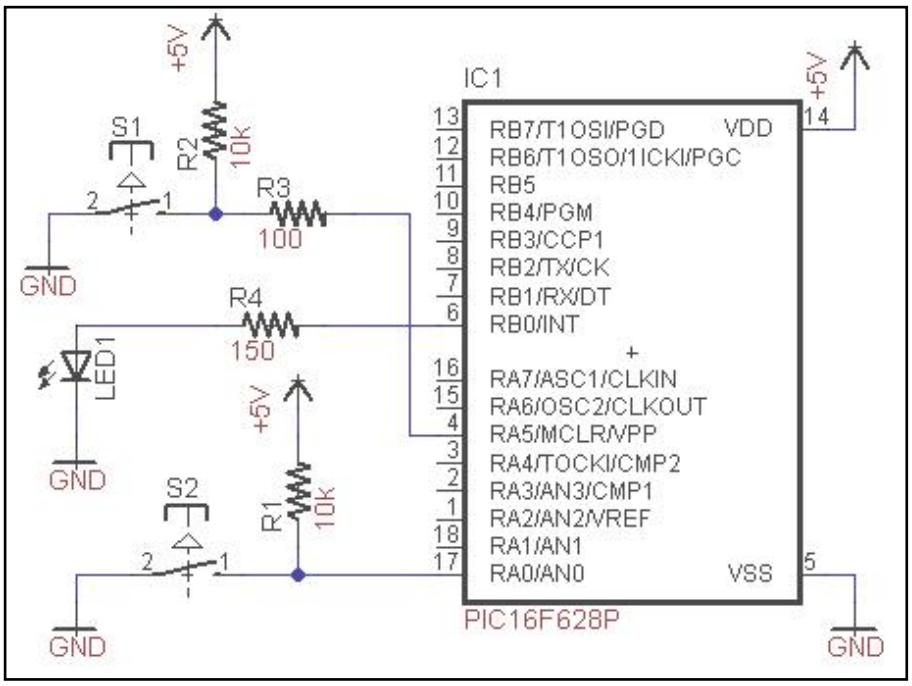


Figura 8. Hardware necesario para el primer ejemplo.

```
;**** Control de Led ****
Bucle
btfsc   PORTA,Pulsador          ; Preguntamos si esta en 0 lógico.-
        goto    Apagar          ; Esta a 1 lógico, Apagamos Led.-
        bsf     PORTB,Led       ; Esta a 0 lógico, Encendemos Led.-
        goto    Bucle           ; Testeamos nuevamente la condición del Pulsador.-

Apagar
bcf     PORTB,Led               ;Apagamos Led.-
        goto    Bucle           ; Testeamos nuevamente la condición del Pulsador.-

        end                     ; Terminamos código.-
```

```
; **** Encabezado ****
 list p=16F628A
 #include P16F628A.inc
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON

;**** Definición de variables ****
Led             equ     0       ; Definimos Led como el bit cero de un registro, en este caso
                                ; PORTB.-
Pulsador        equ     0       ; Definimos Pulsador como el bit 0, en este caso será para PORTA

;**** Vector Reset ****
Reset
org     0x00                    ; Aquí comienza el micro.-
        goto    inicio          ; Salto a inicio de mi programa.-
        org     0x05            ; Origen del código de programa.-
```

Programa completo.

```
;**** Programa Principal ****
Inicio
;**** Configuración de puertos ****

        movlw b'00000111'       ; Configuramos PORTA como I/O Digital.
        movwf CMCON             ; Movemos 111 a W, y W a CMCON.-
bsf     STATUS,RP0              ; Pasamos de Banco 0 a Banco 1.-
                                ; TRISA por default esta en 11111111.-
        movlw b'11111110'       ; Movemos 11111110 a W.-
        movwf TRISB             ; Movemos W a TRISB.-
        bcf     STATUS,RP0      ; Paso del Banco 1 al Banco 0
        bcf     PORTB,Led       ; Apago Led.-

;**** Control de Led ****
Bucle
btfsc   PORTA,Pulsador          ; Preguntamos si esta en 0 lógico.-
        goto    Apagar          ; Esta a 1 lógico, Apagamos Led.-
        bsf     PORTB,Led       ; Esta a 0 lógico, Encendemos Led.-
        goto    Bucle           ; Testeamos nuevamente la condición del Pulsador.-

Apagar
bcf     PORTB,Led               ; Apagamos Led.-
        goto    Bucle           ; Testeamos nuevamente la condición del Pulsador.-

        end                     ; Terminamos Código.-
```

Las instrucciones del microcontrolador necesitan 1 ciclo de máquina excepto algunas excepciones, como son los comandos que incluyen saltos (goto, call, btfss, btfsc, return, etc.) que necesitan dos ciclos de máquina.

.Demoras mediante lazo simple

Para explicar como se calcula empezaremos con una de 1 solo ciclo, ósea:

Entre paréntesis se muestra el número de ciclos que demora cada instrucción. De manera que el número de ciclos de instrucción Tsub consumidos por la rutina, incluyendo los 2 ciclos de la llamada (CALL) serán:

[b]Tsub = [2 + 1 + 1 + (0xXX - 1)*(1 + 2) + 2 + 2] ciclos = (3*0xXX + 5) *Tcy [/b]

Donde Tcy es la duración en segundos de un ciclo de instrucción. Utilizando un oscilador de 4 MHz la mayor duración posible es de 770us, con 0xXX = 0xFF.

.Demoras mediante lazos anidados

Para lograr demoras de mayor duración deben utilizarse lazos anidados, poniendo un lazo de demora dentro de otro. Veamos el ejemplo de la página siguiente para poder comprenderlo:

```
Demora_xxus
        movlw 0xXX      ; Cargamos valor XX que controla duración (1)
        movwf Contador  ; Iniciamos Contador (1)
Repeticion
        Decfsz Contador ; Decrementa contador y si es cero sale  (1 si no sale, 2 si sale)
        goto   Repeticion ; No es 0, repetimos (2)
        return          ; Regresamos de la subrutina (2)
```

```
Demora_xx
        movlw 0xXX              ; (1)
        movwf Contador1         ; (1)
Repeticion1
        movlw 0xYY              ; (1)
        movwf Contador2         ; (1)
Repeticion2
        decfsz Contador2,1      ; (1 si no sale, 2 si sale)

        goto   Repeticion2      ; (2)
        decfsz Contador1,1      ; (1 si no sale, 2 si sale)
        goto   Repeticion1      ; (2)
        return                  ; (2)
```


"Para lograr demoras de tiempo creamos lazos simples o una combinación de estos"

La duración de esta rutina en ciclos de reloj está dada por la siguiente fórmula, y deberá ser:

Tsub = 2 + 1 + 1 + (0xXX)*[1 + 1 + (0xYY - 1)*(1 + 2) + 2 + 1 + 2] + [1 + 1 + (0xYY - 1)*(1 + 2) + 2 + 2 + 2] ciclos

Lo cual se puede simplificar como sigue:

[b]Tsub = [0xXX*((0xYY - 1)*3 + 7) + 5] Tcy[/b]

En este caso, para oscilador de 4MHz el máximo que se puede conseguir es de aproximadamente 196mS.

.Ejemplo #2: LED titilando

En este ejemplo se aplicará la rutina de demora. Se hará titilar un LED conectado a RB0 siempre que el interruptor conectado a RA0 este cerrado. El hardware necesario es idéntico al del primer ejemplo.

```
; **** Encabezado ****
list p=16F628A
 #include P16F628A.inc
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON

;**** Definición de variables ****
Contador1    equ    0x20   ; Seleccionamos posición en la memoria RAM (GPR) para guardar
                           ; registro utilizado para demora.-
Contador2    equ    0x21   ; Registro utilizado en demora.-
Led          equ    0      ; Definimos Led como el bit cero de un registro, en este caso
PORTB.-
Pulsador     equ    0      ; Definimos Pulsador como el bit 0, en este caso sera para PORTA

;**** Vector Reset ****
Reset
org     0x00          ; Aquí comienza el micro.-
        goto   inicio         ; Salto a inicio de mi programa.-
        org    0x05           ; Origen del código de programa.-
;**** Programa Principal ****
Inicio
;**** Configuración de puertos ****

        movlw b'00000111'  ; Configuramos PORTA como I/O Digital.
        movwf CMCON        ; Movemos 111 a W, y W a CMCON.-
```

```
bsf     STATUS,RP0          ; Pasamos de Banco 0 a Banco 1.-
                            ; TRISA por default esta en 11111111.-
        movlw b'11111110'   ; Movemos 11111110 a W.-
        movwf TRISB         ; Movemos W a TRISB.-
        bcf   STATUS,RP0        ; Paso del Banco 1 al Banco 0
        bcf   PORTB,Led  ; Apago Led.-

;**** Control de Led ****
Bucle   btfsc   PORTA,Pulsador  ; Preguntamos si esta en 0 lógico.-
        goto    Apagar                  ; Esta a 1 lógico, Apagamos Led.-
        bsf     PORTB,Led       ; Esta a 0 lógico, Encendemos Led.-
        call    Demora_150ms    ; Mantenemos prendido 150 milisegundos
        bcf     PORTB,Led       ; Apagamos Led
        call    Demora_150ms    ; Apagamos durante 150 ms, Ya realizamos un titilo.-
        goto    Bucle           ; Testeamos nuevamente la condición del Pulsador

Apagar      bcf     PORTB,Led           ;Apagamos Led.-
        goto    Bucle           ; Testeamos nuevamente la condicion del Pulsador.-

;**** Demora ****
Demora_150ms
        movlw 0xFF              ;
        movwf Contador1         ; Iniciamos contador1.-
Repeticion1
        movlw 0xC3              ;
        movwf Contador2         ; Iniciamos contador2
Repeticion2
        decfsz Contador2,1      ; Decrementa Contador2 y si es 0 sale.-
        goto   Repeticion2      ; Si no es 0 repetimos ciclo.-
        decfsz Contador1,1      ; Decrementa Contador1.-
        goto   Repeticion1      ; Si no es cero repetimos ciclo.-
        return                  ; Regresa de la subrutina.-

        end                     ; Terminamos código.
```

.Creación de tablas

El PC, direccionamiento del programa: especifica la dirección de la instrucción que se ejecutará. Consta de 13bits, con lo que es posible direccionar hasta 8K palabras, pero en el PIC16F628 solo se implementa 2k.

La parte alta del contador de programa (**PCH**) no se puede acceder directamente, ella debe cargarse desde los 5bits más bajos del registro llamado **PCLATCH** (dirección 0x0A).

En la creación de tablas, la posición a leer de la misma se realiza con el control del registro **PCL**. Este registro es de 8bits, por lo que direcciona solo 256 posiciones, por ello se debe tener en cuenta: la posición de la tabla en la memoria de programa, y el tamaño de la tabla, si nuestra tabla tiene más de 255 posiciones, si o si debemos manejar los bits más significativos de PC [PCLATCH].

Para devolver el valor direccionado se utiliza **retlw**, ésta instrucción devuelve un valor en el acumulador al retornar de una subrutina. La creación de la tabla se hará de la siguiente forma:

```
Tabla
addwf PCL,f
retlw   Valor0
retlw   Valor1
retlw   Valor2
```

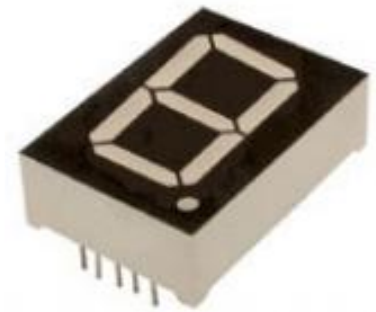
Donde Valor0, Valor1, Valor-2, ..., etc. son los valores que queremos almacenar en la tabla.

La estrategia a seguir para consultar algún valor de la tabla es cargar en el acumulador (W) la dirección de la tabla donde se encuentra el valor que quieres leer y después llamar a la subrutina TABLA (con un CALL). *Advertencia: la carga de W no puede superar el número de valores de la tabla, sino se estará ejecutando una instrucción errónea provocando un mal funcionamiento del programa.*

.Ejemplo #3: control de display 7 segmentos

Para aplicar el uso de las tablas vamos a hacer un ejemplo donde se controle un display de 7 segmentos. Un display es una colección de LEDs ubicados de forma estratégica. Si se los agrupa uniendo sus cátodos será de **cátodo común**, o bien agrupando sus ánodos, un display de **ánodo común**.

Por otro lado estos LEDs pueden ser fabricados en forma de puntos o segmentos, tal es así que se encuentran display de 7 segmentos como los de la figura #9:



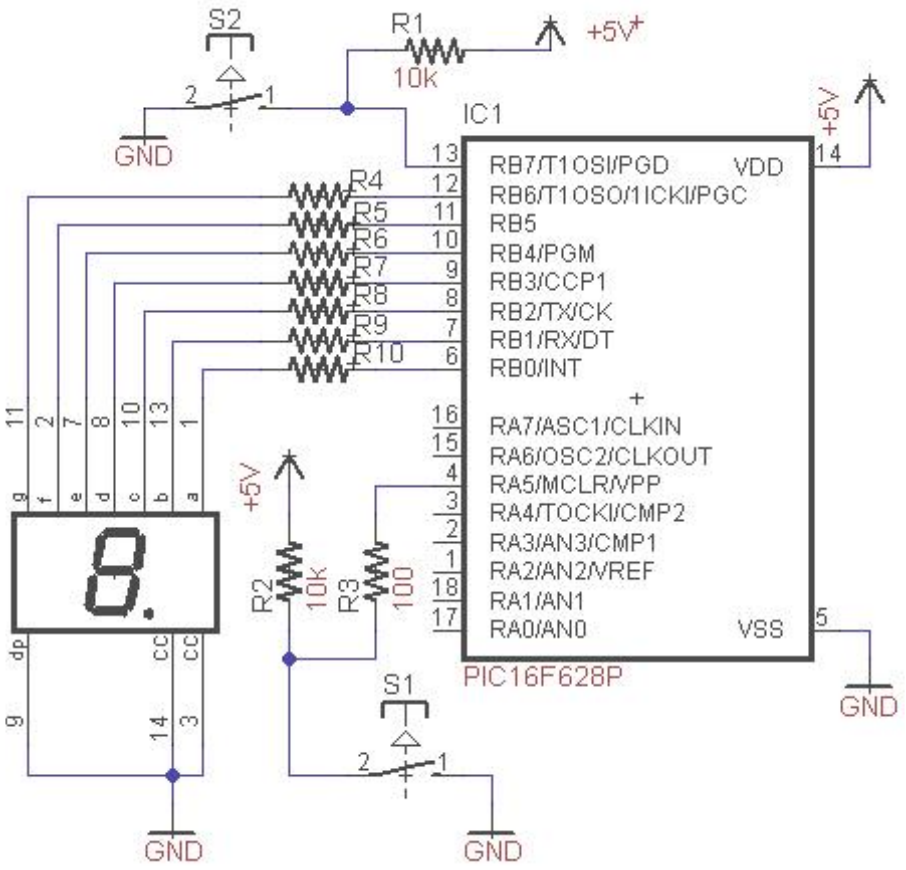


Figura 10. Hardware necesario para el ejemplo #3.

El programa que realizaremos leerá la cantidad de veces que se activa un pulsador y mostraremos el resultado. Conectaremos el display en forma directa, es decir conectando el puerto B del microcontrolador a los pines del display, y luego encender cada uno de los segmentos del display para visualizar el valor correspondiente. Para ello crearemos una tabla que contenga los distintos códigos para el número que necesitemos visualizar. Es obvio que con un solo display solamente podremos contar de 0 a 9.

Una manera más cómoda de escribir la tabla de instrucciones **RETLW** puede lograrse usando la directiva **DT** (Define Table) del ensamblador, la cual nos permite definir una tabla de datos que será sustituida por una lista de instrucciones **RETLW**; *(continua en la página 0x22)...*

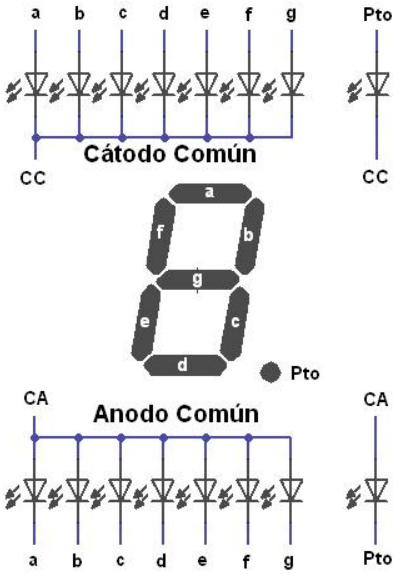


Figura 11. Display por dentro.

```
; **** Encabezado ****
list p=16F628A
#include P16F628A.inc
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTOSC_OSC_NOCLKOUT & _DATA_CP_OFF & _LVP_OFF & _MCLRE_ON

;**** Definición de variables ****
Contador    equ     0x20    ; Registro para almacenar conteo
Contador1   equ     0x21    ; Registro utilizado en demora.-
Contador2   equ     0x22    ; Registro utilizado en demora.-
Pulsador    equ     7       ; Definimos Pulsador como el bit 7, en este caso será
                            ; PORTB.-

;**** Inicio  del Micro ****
Reset org   0x00            ; Aquí comienza el micro.-
      goto  Inicio          ; Salto a inicio de mi programa.-

;**** Tabla de conversión BCD a 7 Segmentos ****
                            ; Se coloca al inicio para asegurar ubicación en Página.-
      org   0x05            ; Origen del código de tabla.-
BCD7SEG:                    ; retlw b'gfedcba'  para display cátodo común
      addwf PCL,1           ; Se incrementa el contador del programa.-
      retlw b'0111111'      ; 0
      retlw b'0000110'      ; 1
      retlw b'1011011'      ; 2
      retlw b'1001111'      ; 3
      retlw b'1100110'      ; 4
      retlw b'1101101'      ; 5
      retlw b'1111101'      ; 6
      retlw b'0000111'      ; 7
      retlw b'1111111'      ; 8
      retlw b'1101111'      ; 9
      clrf  Contador        ; Si llega 10, se resetea contador
      retlw b'0111111'      ; 0

;**** Programa principal ****

;**** Configuración de puertos ****
Inicio bsf  STATUS,RP0      ; Pasamos de Banco 0 a Banco 1.-
      movlw b'10000000'     ; RB7 como entrada y los demás como salida.-
      movwf TRISB
      bcf   STATUS,RP0      ; Paso del Banco 1 al Banco 0
      movlw b'0111111'      ; Comienza en cero.-
      movwf PORTB
      clrf  Contador

;**** Testeo de Pulsador ****
Testeo
      btfsc PORTB,Pulsador  ; Testeamos si esta a 0 lógico.-
      goto  Testeo          ; No, seguimos testeando.-
```

```
        call    Demora_20ms     ; Eliminamos Efecto rebote
        btfsc   PORTB,Pulsador  ; Testeamos nuevamente.-
        goto    Testeo          ; Falsa Alarma, seguimos testeando.-
        incf    Contador,1      ; Se ha pulsado, incrementamos contador.-
        movfw   Contador        ; pasamos contador a W
        call    BCD7SEG         ; Llamamos tabla.-
        movwf   PORTB           ; Cargamos valor recibido por Tabla en PORTB
        btfsc   PORTB,Pulsador  ; Esperamos a que se suelte el pulsador -**-
        goto    $-1             ; No, PCL - 1, --> btfss       PORTA,Pulsador.-
        call    Demora_20ms     ; Eliminamos efecto rebote.-
        btfsc   PORTB,Pulsador  ; Testeamos nuevamente.-
        goto    $-4             ; No, Falsa alarma, volvemos a testear a que se suelte (**).-
goto    Testeo                  ; Si, Testeamos nuevamente.-

;**** Demora ****
Demora_20ms
        movlw   0xFF            ;
        movwf   Contador1       ; Iniciamos contador1.-
Repeticion1
        movlw   0x19            ;
        movwf   Contador2       ; Iniciamos contador2.-
Repeticion2
        decfsz  Contador2,1     ; Decrementa Contador2 y si es 0 sale.-
        goto    Repeticion2     ; Si no es 0 repetimos ciclo.-
        decfsz  Contador1,1     ; Decrementa Contador1.-
        goto    Repeticion1     ; Si no es cero repetimos ciclo.-
        return                  ; Regresa de la subrutina.-

        end
```

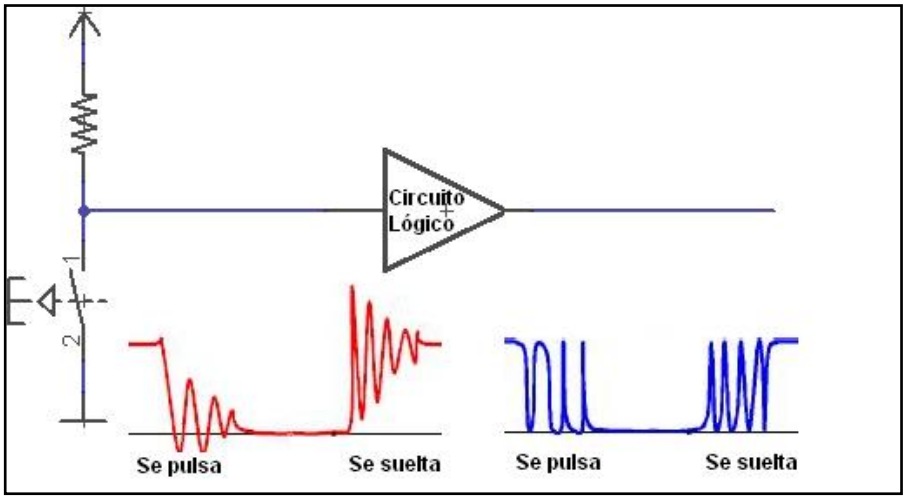
así, la tabla anterior puede quedar como sigue:

```
BCD7SEG:            ; retlw b'gfedcba'  para display
                    ; cátodo común.-
addwf PCL,1         ; Incrementa el contador del programa.-
DT 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,  0xFF, 0x6F
clrf    Contador
retlw   0x3F
```

.Control anti rebote

En el momento de presionar un botón pulsador o cualquier conmutador electromecánico es inevitable que se produzca un pequeño arco eléctrico durante el breve instante en que las placas del contacto se aproximan o se alejan de sus puntos de conexión.

La duración depende de la calidad de los switches y la velocidad de accionamiento, pero no dura más de 20mS. *Seguimos en el próximo número...*




NEOTEO
Ciencia y tecnología al extremo

controlar servos desde el PC

Es muy probable que en algún momento se te haya ocurrido que no debería ser muy complicado controlar los movimientos de un pequeño servomotor desde el ordenador. Y de hecho, no lo es. En este articulo Diego, con su estilo ameno y sencillo, nos explica como hacerlo.

// por: Diego Márquez García-Cuervo //
diego@ucontrol.com.ar

El lenguaje C es el que uso por defecto para el 90% de mis programas. El que vamos a desarrollar es uno de los ejemplos de cómo hacer cosas con este idioma que pueden encontrar en mi página Web. He usado el PCW PICC de CCS (v.3.242). Pero como decía Jack El Destripador: ¡Vayamos por partes!

"Un servo es un motor controlado por una electrónica encargada de moverlo dependiendo de la señal aplicada."

Y para empezar veremos un poco de teoría, que a ninguno de nosotros nos va a venir mal. Un servomotor es un cacharro, entre otros muchos, que puede manejarse inyectándole una señal PWM.

Si me preguntáis qué es esto, os respondo que *"es método de control que consiste en enviar un tren de pulsos, cada uno de ellos con un periodo de tiempo en alto, a 5V, y otro en bajo, a 0V; separados cada uno del siguiente un tiempo constante y que podemos variarle la respectivas duraciones que permanece en alto y bajo"*, o como su propio nombre indica: Pulse Width Modulation, que dicho para entendernos significa **Modulación de Ancho de Pulso**.

Un servo es un motor controlado por una electrónica que lee el **PWM** y que se encarga de mover al motor dependiendo de lo que ha leído. El servo, o mejor dicho la electrónica del servo coloca al motor en cada posición dependiendo del tiempo en que el pulso que le inyectamos permanece en alto. Si el tiempo que dura pulso en estado alto es de exactamente 1.5mS, entonces el servo se coloca en el centro de su recorrido. Si el pulso dura exactamente 0.5mS el servo retrocede desde el punto medio unos 90º y se coloca en su extremo izquierdo; y si, por último, el pulso dura exactamente 2.5mS el servo avanza desde el punto medio unos 90º y se coloca en su extremo derecho. A la relación entre el tiempo en que permanece en alto y bajo un pulso le

llamamos Duty Cycle.

Con duraciones intermedias del tiempo en que permanece el pulso en alto, o sea: con distintos Duty Cycle, el servo se posiciona en puntos intermedios de su recorrido. Para que el servo responda correctamente deben llegarle los pulsos con una periodicidad (o frecuencia) **constante**, uno tras otro, separados 20mS cada uno del siguiente. Además, cada flanco de subida debe estar separado del siguiente flanco de subida los mismos 20mS; por lo tanto cada ciclo alto-bajo dura siempre exactamente 20mS y lo que variamos es la relación entre el tiempo que está en alto y en bajo.

Decir que los pulsos están separados unos de otros 20mS es exactamente lo mismo que decir que se envían con una **frecuencia de 50Hz**, ya que 50Hz son 50 pulsos por segundo y por lo tanto 1000mS (que tiene un segundo) dividido entre 50 son exactamente eso: 20mS.

f (frecuencia en Hertz) = 1 / t (período en Segundos)

En el fondo todo este asunto no es distinto de encender y apagar nuestro famoso LED, que es algo por lo que empezamos todos cuando comenzamos a trastear con los PIC's, pero controlando muy exactamente los tiempos durante los que permanece encendido y apagado. Esto podemos verlo más fácil y claro en la figura #1:

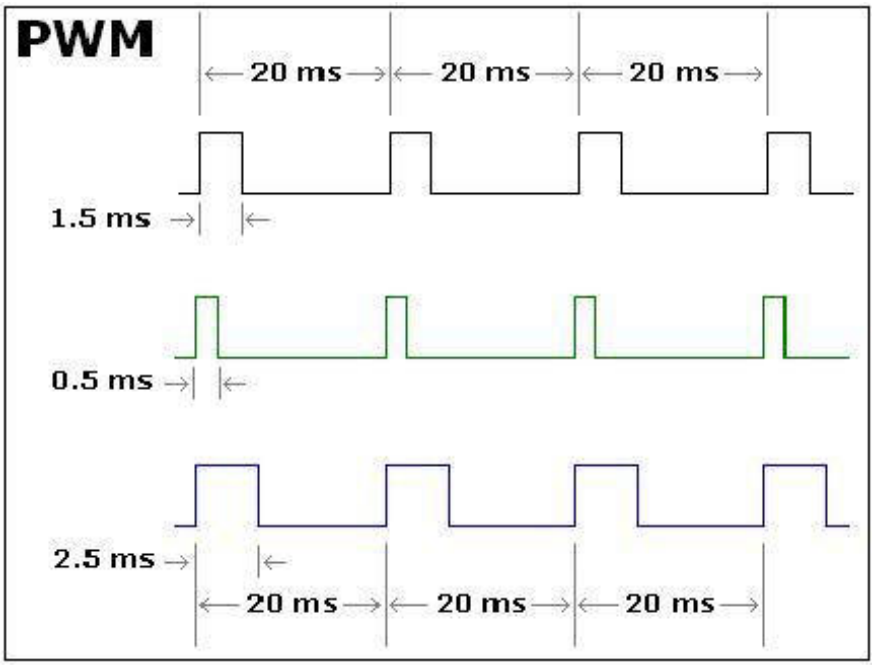


Figura1. Los pulsos se envían con una frecuencia de 50Hz.

Ahora lo que tenemos que hacer es saber cómo podemos controlar estos tiempos en nuestro **PIC**, para poner en alto (disparar el pulso) y en bajo (apagarlo) siguiendo la tabla de tiempos descrita más arriba.

Para ello voy a echar mano del socorrido **TIMER0** del PIC que nos va a servir de reloj para saber cuándo y durante cuánto tiempo tengo que tener mi pulso en alto. Como soy el más listo de la clase he elegido un divisor, o *Preescaler*, del TIMER0 de 1:16 (mas adelante os contaré el por qué de este divisor). El TIMER0 funcionando a 1:16 hace saltar la **Interrupción por Desbordamiento de Timer**, también conocida como **RTCC**, cada 4.096mS.

Esto es lo mismo que decir que TIMER0 tarda 4.096mS en contar desde 0 a 255 y que al llegar a 255 pasar de nuevo a 0 hace saltar la RTCC. Cada paso de contador del TIMER0, a lo que vamos a llamar un **tick de reloj**, tarda 4.096 / 256 = 0.016mS. Esto me da una pauta bastante fácil de calcular que consiste en que cada 5 RTCC completas tengo 5 * 4.096 = 20.48mS que es un poco más de lo que necesito, que son 20mS exactos.

Esto lo podemos conseguir contando **4 RTC's completas**, a 4.096mS cada una, y **otra más un poco mas corta**. No podemos hacer que la RTCC se acabe antes de la cuenta, pero si que podemos, y es lo que vamos a hacer, que **no empiece a contar desde 0 sino desde 30**: esto se explica porque 30 * 0.016 = 0.48mS menos que va contar esta última RTCC al haber empezado desde un valor de 30 en lugar de 0, luego 4.096 - 0.48 = 3.616mS para la última RTCC.

Concluyendo: tengo cuatro RTCC's completas a 4.096mS y una capada a

3.616 luego 4 * 4.096 + 3.616 = 20mS. Lo que realmente voy a hacer es contar 1 RTC completa, 2 RTCC completas, 3 RTCC completas, 4 RTCC completas y pongo el contador de TIMER-0 a 30, 5 RTCC completas luego ya han pasado 20mS exactamente.

A estas 5 RTCC's les llamo **flagRTCC** que solo voy a activar cuando se completen las 5 RTCC's (4 completas y otra mas incompleta).

Además, sabiendo que cada tick de reloj ocupa 0.016mS podemos traducir los tiempos de anchos de pulsos descritos anteriormente en ticks de reloj: así **0.5mS** son lo mismo que esperar **31 ticks** de reloj, **1.5mS** equivalen a **93 ticks** de reloj y **2.5mS** son **155 ticks** de reloj. (Recuerda que un tick de reloj es el tiempo que tarda TIMER0 en contar exactamente 1, o sea 0.016mS).

La imagen anterior podemos ahora convertirla en la figura #2 en las que hemos **cambiado los tiempos por RTCC's y Ticks**.

Como vemos en el nuevo cronograma: cada vez que se produce un **super RTCC**, de 4 RTC's y pico a la que llamamos **flagRTCC**, ponemos en alto el PIN de la señal del servo, debemos recordar que esto siempre va a ocurrir en el tránsito del contador TIMER0 entre los valores de 255 y 0, por lo que **flagRTCC siempre va a coincidir con TIMER0 = 0**.

Ahora entonces solo debemos esperar el número suficiente de ticks para volver a poner nuestro **PIN** a bajo. Si deseamos que el servo se posicione en su centro debemos mantener el **PIN** en alto durante 93 ticks de TIMER0 o, lo que es lo mismo, esperar 1.5 mS para bajar el pulso.

La secuencia queda ahora de la siguiente manera:

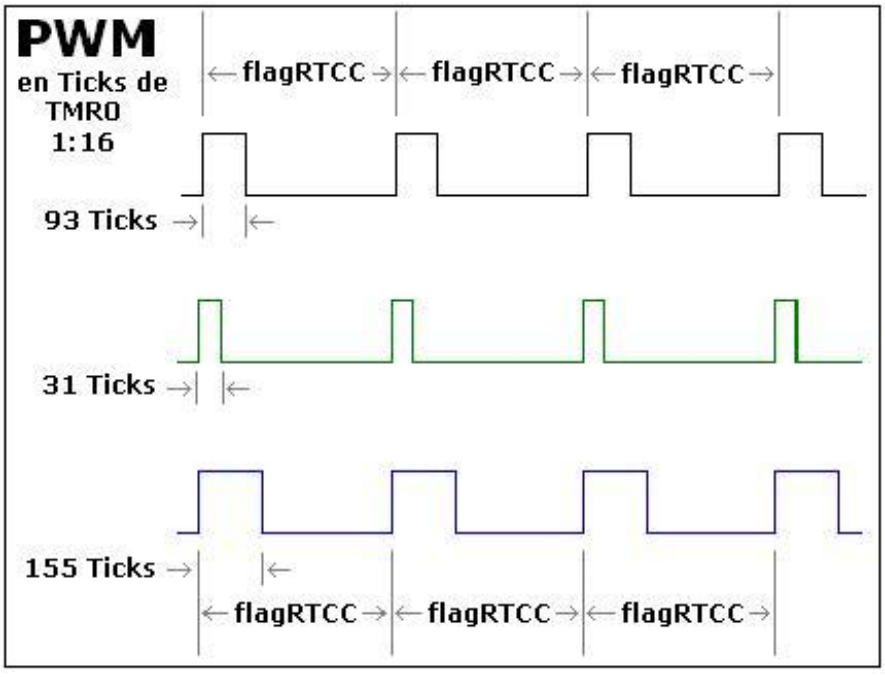


Figura #2. Tiempos de anchos de pulso y ticks de reloj.

"A la relación entre el tiempo en que permanece en alto y bajo un pulso le llamamos Duty Cycle."

- RTCC corre alocadamente, una tras otra, dedicándose exclusivamente a contar cuantas de ellas han pasado, si es la cuarta pone Timer0 a 30 para que la quinta sea más corta, si es la quinta pone en alto flagRTCC para lo que sea necesario y comienza de nuevo.

- En el programa principal detectamos que flagRTCC se ha activado así que lo desactivamos y ponemos en alto el PIN y marcamos, con flagSERVO1, que acabamos de activarlo.

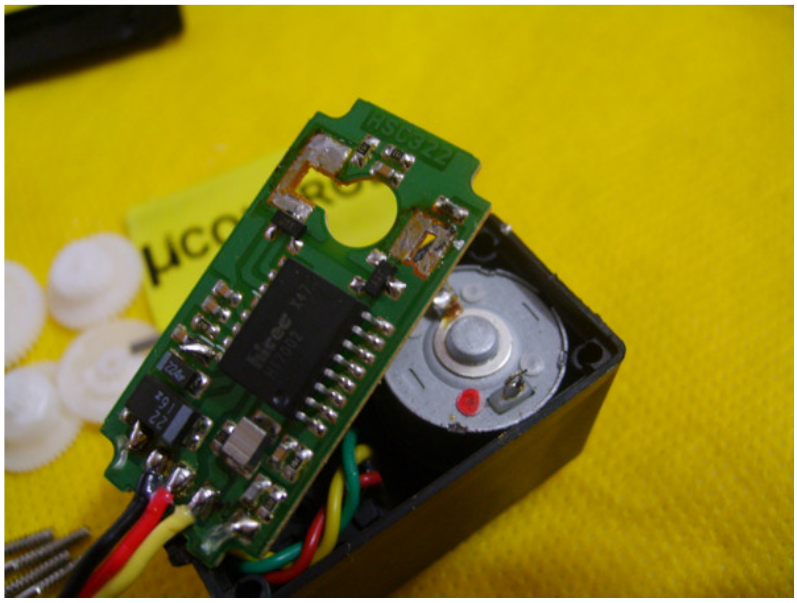
- A continuación, y siempre que flagSERVO1 esté activado, comprobamos el valor de TIMER0 que si es mayor que el que deseamos, en nuestro caso 93, y cuando lo alcancemos ponemos a bajo el PIN y lo marcamos desactivando flagSERVO1. Hemos acabado.

Cada 20mS activamos el pulso, y transcurridos 1.5mS lo desactivamos, que es exactamente lo que queríamos hacer.

El valor de TIMER0 con el que comparamos para controlar la duración de cada pulso está guardado en tSERVO1, que inicialmente

"Con los comandos "1", "2" y "3" podemos cambiar el valor de tSERVO1 a ticks_PULSO_MINIMO, ticks_PULSO_MEDIO y ticks_PULSO_MAXIMO respectivamente."

"HITEC HS-311, un servo muy popular. En las fotos se puede apreciar su formato físico, el pequeño motor y la placa controladora."



cargamos con el número de ticks necesarios para colocar el servo en su punto medio, **ticks_PULSO_MEDIO**, y que podemos cambiar dinámicamente mediante la recepción de comandos a través de la RS232.

De esta forma con los comandos **"1", "2" y "3"** podemos cambiar el valor de **tSERVO1** a **ticks_PULSO_MINIMO**, **ticks_PULSO_MEDIO** y **ticks_PULSO_MAXIMO** respectivamente; y con los comandos "+" y "-" vamos incrementando o decrementando su valor. Con "r" le pedimos al PIC que nos envíe su valor actual.

Ahora solo nos quedaría implementar todo esto en un programa en C que queda de la siguiente forma:

```
// servo_pwm_232

// Ejemplo con un servo FUTABA S3003
// Alimentación y pulsos a 5V
// Cuadro de Tiempos :
// Periodo 20 ms (Frecuencia 50 Hz)
// Ancho Pulso minimo 0.5 ms
// Ancho pulso medio 1.5 ms
// Ancho pulso maximo 2.5 ms
// TMR0 a 1:16 -> 1 RTCC cada 4.096 ms
// -> 1 Tick cada 0.096 / 256 = 0.016 ms
// -> 20 ms = (4 x RTCC completas) + (1 * RTCC - 30 ticks)
// Ancho Pulso minimo 0.5 ms -> 31 ticks de TMR0
// Ancho pulso medio 1.5 ms -> 93 ticks de TMR0
// Ancho pulso maximo 2.5 ms -> 155 ticks de TMR0

#include <16f876a.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)

#define PIN_SERVO1 PIN_B0

const int AJUSTE_FINO_DE_RTCC = 30;
const int ticks_PULSO_MINIMO = 31;
const int ticks_PULSO_MEDIO = 93;
const int ticks_PULSO_MAXIMO = 155;

int1 flagRTCC = 0;
int contRTCC = 0;
int1 flagSERVO1 = 0;
int tSERVO1 = ticks_PULSO_MEDIO;
char Keypress =0x00;

void eco_servos(void);
void ajusta_servo(void);

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

#int_RTCC
RTCC_isr(){
  ++contRTCC;
  if(contRTCC==4){
```

```
    set_TIMER0(AJUSTE_FINO_DE_RTCC);
  }
  if(contRTCC==5){
    flagRTCC=1;
    contRTCC=0x00;
  }
}

void main() {

int ValTIMER0;

  setup_counters(RTCC_INTERNAL,RTCC_DIV_16);
  enable_interrupts(int_rda);
  enable_interrupts(global);
  printf("\r\n\SERVO Commander\r\n\r\n" );
  eco_servos();
  set_TIMER0(0);
  enable_interrupts(INT_RTCC);
  do {
    // DISPARO DEL PULSO PWM
    if(flagRTCC==1){
      flagRTCC=0;
      output_high(PIN_SERVO1);
      flagSERVO1=1;
    }
    // CONTROL DE ANCHO DEL PULSO PWM
    if(flagSERVO1==1){
      valTIMER0 = get_TIMER0();
      if(valTIMER0>tSERVO1){
        flagSERVO1=0;
        output_low(PIN_SERVO1);
      }
    }
    // CONTROL DESDE LA RS-232
    if(Keypress!=0x00){
      ajusta_servo();
      Keypress=0x00;
    }
  } while (TRUE);
}

void ajusta_servo(void){

  switch(Keypress){
    // Periodos Prefijados
    case '1': tSERVO1=ticks_PULSO_MINIMO;
              break;
    case '2': tSERVO1=ticks_PULSO_MEDIO;
              break;
```

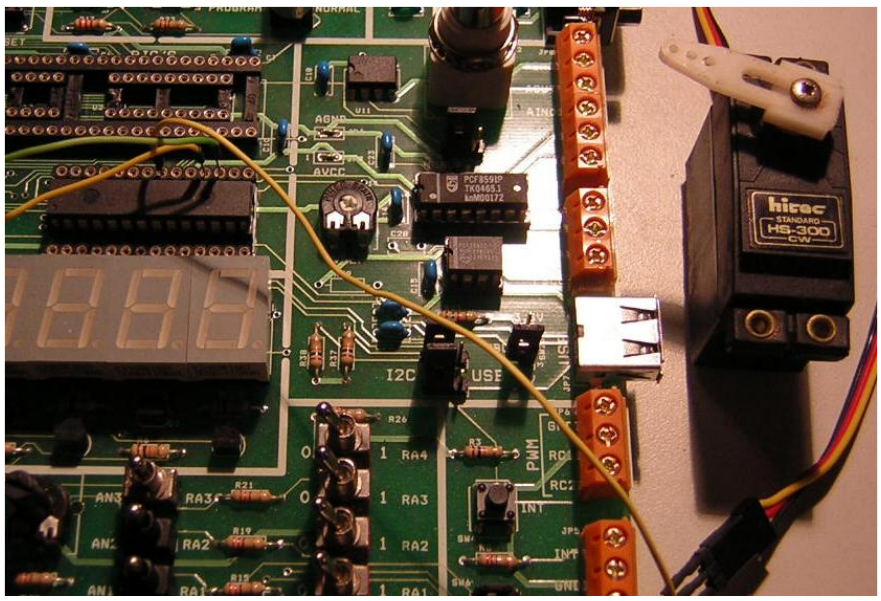
```
   case '3': tSERVO1=ticks_PULSO_MAXIMO;
             break;
   // Inc Dec Periodo
   case '+': if(++tSERVO1>ticks_PULSO_MAXIMO){
               tSERVO1=ticks_PULSO_MAXIMO;
             }
             break;
   case '-': if(--tSERVO1<ticks_PULSO_MINIMO){
               tSERVO1=ticks_PULSO_MINIMO;
             }
             break;
   // Dame Periodo actual
   case 'r': eco_servos();
             break;
  }
}

void eco_servos(void){
  printf("S=%u\r\n",tSERVO1);
}
```

Como podéis ver en el comentario inicial del código, todo esto está montado para la familia de servos compatibles **FUTABA S300-3, HiTec HS-300 CW, HOBBICO COMMAND CS-51,** que son de los más usados por los aficionados al radiocontrol.

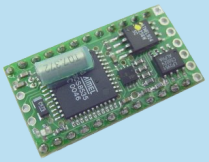
Para ajustar este código a otros servos solo hay que calcular el Preescaler y los ticks necesarios para ajustarse a las características de éste. Hay servos que funcionan a 400Hz en lugar de 50Hz y con anchos de pulso ligeramente distintos a los utilizados aquí. Es normal anchos de pulso en los extremos de 1.00 y 2.00mS respectivamente. El punto medio en 1.5mS es muy común.

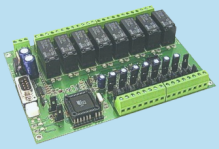
Otro corolario de este ejemplito estriba en la posibilidad de manejar varios servos. Utilizando la misma estructura de **PIN_SERVO1** y **tSERVO1** podemos habilitar el control indistinto de tantos servos como deseemos, teniendo así **PIN_SERVO2** y **tSERVO2**, **PIN_SERVO3** y **tSERVO3,** ..., etc. controlando cada uno de ellos de forma absolutamente similar.

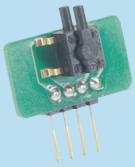
“Para ajustar este código a otros servos solo hay que calcular el Preescaler y los ticks necesarios para ajustarse a las características de éste.”



Servo controlado por una placa entrenadora.





automatismos

MAR DEL PLATA

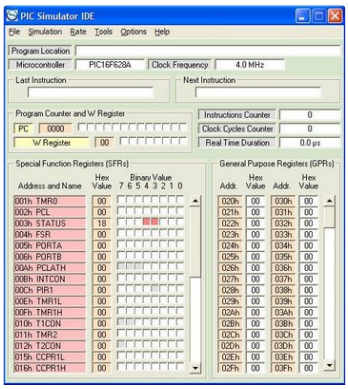
* noticias
* artículos
* proyectos
* ideas de diseño
* recursos didacticos
* y mucho, mucho más...

http://www.automatismos-mdq.com.ar/blog

addons en PIC simulator IDE

Esta guía está destinada a programadores principiantes y/o avanzados que desean crear módulos externos al simulador de PIC de Vladimir Soso, el PIC SIMULATOR IDE.

// por: Sergio Luis Scarnatto //
sergiols@keko.com.ar



El PIC SIMULATOR IDE (PSIDE) es una suerte de suite de desarrollo para la simulación de un microcontrolador real con múltiples herramientas internas y externas llamadas módulos para la prueba de los diferentes dispositivos con los que se comunica el micro. Para realizar la simulación, el PSIDE recrea la memoria de programa, registros, memoria EEPROM, puertos, etc., haciendo posible la construcción de nuestros proyectos en forma virtual y segura antes de lanzarnos a construir el hardware.

.Módulos externos

Bajo el menú **Tools** del PSIDE se listan todos los módulos de la suite. Para la realización de módulos propios es necesario utilizar la opción **"External Modules"** para subscribir nuestro módulo al entorno de desarrollo. En la figura #2 podemos ver dicho menú y uno de los módulos: 8 LEDs con salidas configurable a diferentes puertos, la opción "Always On Top" es útil cuando se nos llena la pantalla de ventanas y queremos que esté siempre visible.

.Creación de un módulo externo

Para la creación de un módulo externo para PSIDE es necesario comprender algunos conceptos de utilización de objetos **COM** desde **.NET**.

.COM (Component Object Model)

Es una plataforma de Microsoft implementado desde Windows 9x como una forma de lograr la comunicación, creación y destrucción de objetos que residen en diferentes contextos de ejecución (entre diferentes aplicaciones y/o sistemas).

Los componentes de COM se programan en cualquier tipo de lenguaje siempre que utilice los servicios y requerimientos de COM, que son bastantes.

Muchos programadores de Visual Basic 6, por ejemplo, conocen muy superficialmente los conceptos de COM, dado que la complejidad está oculta en la pseudo máquina virtual.

Con el tiempo llegó Microsoft .NET y se impuso como un estándar para la creación, comunicación, destrucción y notificación de objetos redefiniendo varios de estos conceptos.

Cuando se construye un objeto COM lo que se está haciendo es definiendo una interface, algo así como el prototipo de una función en C, y se la registra en un catálogo (en este caso el registro de Windows).

Para lograr la comunicación entre un objeto .NET y un objeto COM es necesario crear una envoltura (Wrapper en inglés) que realiza las conversiones necesarias en los tipos de datos de C# y los definidos por el objeto COM.

Esta envoltura o wrapper se encarga además de procesar los temas de manejo de tiempo de vida, punteros a estructuras y funciones del objeto COM.

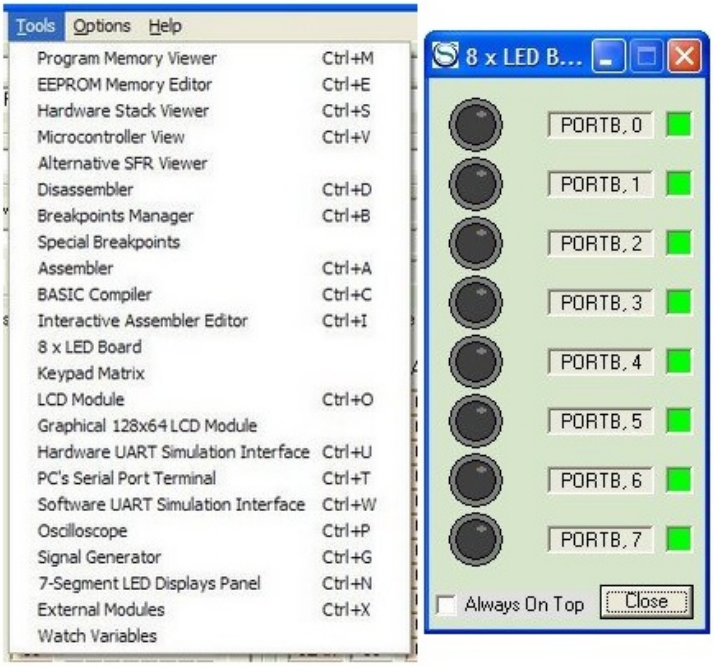
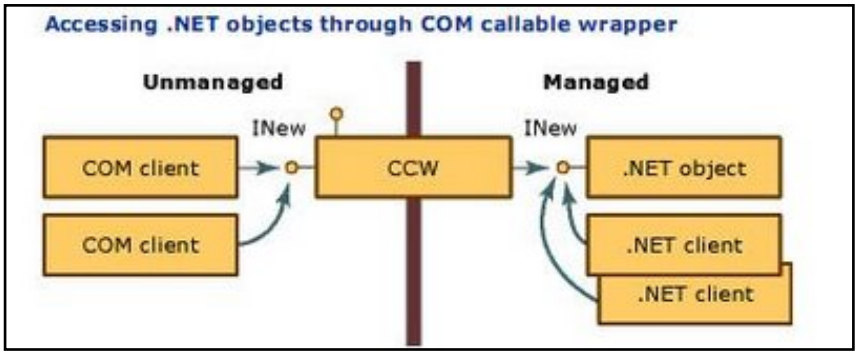


Figura #2. Vista del menú Tools (Herramientas) y del módulo externo "8 x LED".

.Introducción a COM callable wrapper

Cuando un cliente COM llama a un objeto .NET, el CLR (Common Lenguage Runtime) crea el objeto manejado y un CCW (COM Callable Wrapper) para el objeto. Dado que no es posible para un cliente COM hacer una referencia directa de un objeto .NET, deben usar este CCW como un proxy al objeto manejado.

.Creación de un ensamblado "visible" desde PSIDE



Creamos un nuevo proyecto en **Visual Studio** y abrimos el archivo ***AssemblyInfo.cs***. Veremos que el archivo contiene la siguiente línea:

```
// Setting ComVisible to false makes the types in this assembly not visible to COM components.  If you need to access a type in this assembly from COM, set the ComVisible attribute to true on that type.

[assembly: ComVisible(false)]
```

Cambiamos **false** por **true** y de esta manera todas las clases públicas de nuestro proyectos se van a ver como objetos COM.

Luego podríamos crear por ejemplo una nueva clase llamada "Server".

Los atributos de esta clase indican que se desea generar en forma transparente las interfaces que imple-

mentan **IDispatch** (ver Automation en la MSDN), ya que Visual Basic 6 (lenguaje en el que está programado el PSIDE) no puede entender las intefaces simples **IUnknown**.

Según el autor del PSIDE, el entorno nos enviará tres eventos a nuestra clase:

objectinit(): *Invocado por el PSIDE cuando se inicia la simulación. Lo podremos utilizar para inicializar el estado interno de nuestro módulo.*
objectrefresh(): *Invocado luego de cada instrucción simulada.*
objectterm(): *Invocado antes de que PSIDE elimine el módulo de la memoria. Usado normalmente para cerrar recursos utilizados.*

Dado que C# es un lenguaje case-sensitive (diferencia minúsculas y mayúsculas) es importante notar que los métodos se deben escribir en todos minúsculas.

.Programación de un formulario principal y módulo cliente/servidor

Al crear un nuevo formulario o pantalla principal en **VS.NET** agregamos la siguiente referencia al proyecto. Al crear esta referencia a los servicios del PSIDE el Visual Studio nos está creando por detrás la **RWC** (wrapper del cliente .NET al objeto COM).

Luego modificamos el código del formulario agregando la referencia a los servicios de PSIDE y **System.Runtime.InteropServices**.

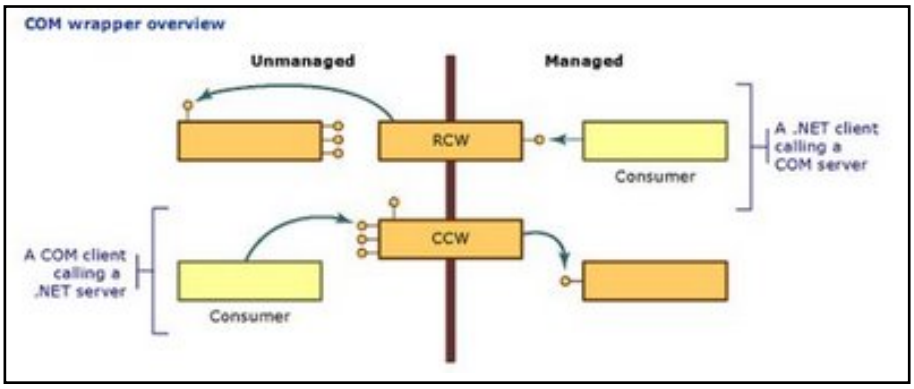


Figura #4. COM wrapper overview.

```
using System.Runtime.InteropServices;

namespace picsimtest
{
  [ClassInterface(ClassInterfaceType.AutoDispatch)]
  [ProgId("PICSimTest.Server")]
  [Guid("FA08D8C2-3616-483c-8B21-B7C72D6B5E7D")]
  public class Server
  {
  }
}
```

Código para la creación de una clase llamada "Server".

```
using pic18simulatoride;

using System.Runtime.InteropServices;

namespace picsimtest
{
  [ComVisible(false)]
  public partial class MSCKeyboard : Form
  {
    private server _server;
```

Agregando la referencia a los servicios de PSIDE.

La variable **_server** contendrá la instancia al objeto COM de los servicios del PSIDE. Marcamos la clase del formulario como ComVisible(false) para no publicar clases que no son necesarias ni útiles para el PSIDE. Luego en el constructor le damos instancia a la variable **_server**:

```
public MSCKeyboard()  //<-- constructor
{
  InitializeComponent();

  _server = new server();
}
```

.Acceso a los servicios de PSIDE

Abrimos el object browser y seleccionamos la librería **Interop.pic18simulatoride**, y luego navegamos hasta la clase **serverClass** y nos mostrará que tiene los siguientes métodos públicos y disponibles para usar e interactuar con PSIDE.

Los métodos son muy sencillos de entender, por ejemplo:

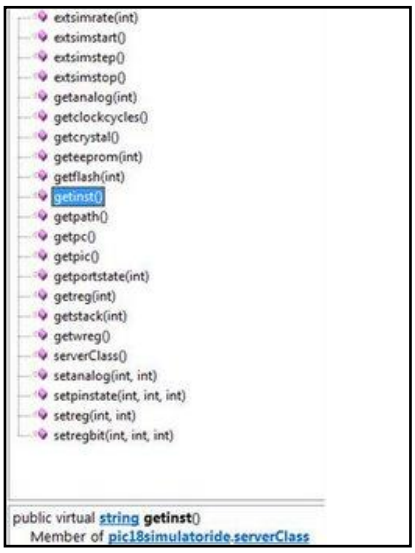
String getpic() *nos devuelve en una cadena el nombre del microcontrolador que seleccionamos en PIC Simulator IDE.*

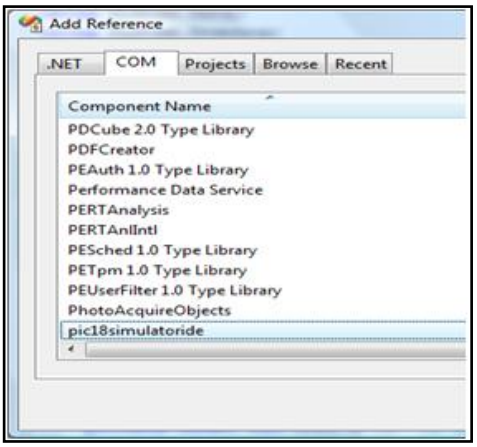
Utilizando **getreg** podremos acceder al valor actual de un determinado puerto que le pasemos como referencia (utilizando la dirección de memoria que figura en la hoja de datos del microcontrolador seleccionado). Por ejemplo:

```
// lee el port B
int portb = s.getreg(0xF81);
```

.Ejemplo: LED indicador 1

Crear un nuevo proyecto llamado **LedTestModule** y configurarlo. Crear un nuevo formulario llamado **LedTest** y configurarlo. Arrastrar un objeto control tipo "Panel" desde la Toolbox al formulario. Cambiar el backcolor del panel a White. Y por último agregar al formulario la "clase server".





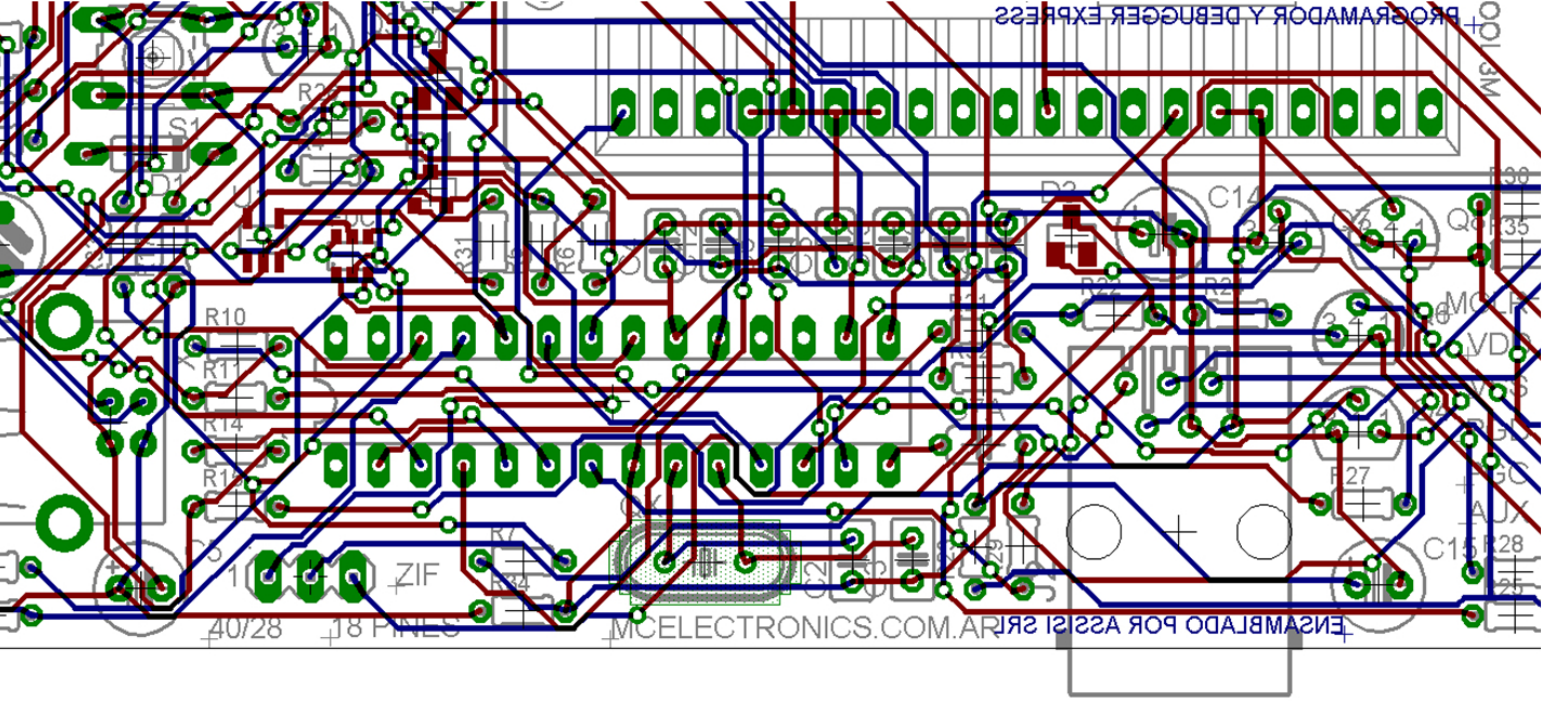
Vista de la ventana de "Agregar referencias" y del "Object browser" de PSIDE.

```
public void RefreshData()
 {
   int ledstate = s.getreg(0xF81) & 0x01;   // lee RB0

   if (ledstate == 1)
     panel1.BackColor = Color.Red;
   else
     panel1.BackColor = Color.White;
 }
```

Ejemplo: LED indicador 1.

```
using System.Runtime.InteropServices;

namespace LedTestModule
{
 [ClassInterface(ClassInterfaceType.AutoDispatch)]
 [ProgId("PICSimTest.Server")]
 [Guid("FA08D8C2-3616-483c-8B21-B7C72D6B5E7D")]
 public class Server
 {
   [ComVisible(false)]
   private LedTest ledTest;

   public Server()
   {
     ledTest = new LedTest();
     ledTest.Show();
   }

   public void objectrefresh()
   {
     ledTest.RefreshData();
   }
 }
}
```

Agregar la clase server.



mc electronics®

Herramientas de Desarrollo para PIC® y dsPIC®.
Training Partner de Microchip en Argentina.

MCELECTRONICS
Austria 1760 - OF 8
Ciudad de Buenos Aires (1425)
BA. Argentina.

(011) 6091-4922/4581
www.mcelectronics.com.ar
info@mcelectronics.com.ar

10% OFF*

En KITs y Programadores Microchip para usuarios de ucontrol.

PIN 79694876633289034567804658741

* Promoción válida para todas las herramientas de desarrollo publicadas en mcelectronics.com.ar. Ingrese el cupón **UC101606** al momento de realizar la compra. Sólo válido para usuarios que se hayan registrado en el foro de ucontrol.com.ar antes del 16 de Junio de 2009. **Promoción vigente hasta el 31/08/09**

conversor IrDA a TTL

Este sencillo circuito esta pensado para formar parte de un proyecto mayor. Se trata de un simple conversor de señales infrarrojas a niveles de tensión TTL. Puede ser empleado, por ejemplo, para comunicar un dispositivo portátil como una PALM con un microcontrolador o incluso con un ordenador hogareño. Las aplicaciones: un sinfín. ¡Manos a la obra!

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

“Con este simple circuito podremos comunicar fácilmente un micro con algún dispositivo portátil como una PALM”

Básicamente, el circuito es un fototransistor infrarrojo cuya salida se amplifica e invierte para convertir los pulsos luminosos que recibe en niveles de tensión compatibles con los niveles TTL, con los cuales trabajan muchos circuitos integrados que normalmente utilizamos en nuestros proyectos.

Se han elegido componentes muy comunes, por lo que no será complicado conseguir todo lo necesario para tener el montaje funcionando en una hora o poco más.

Como puede verse en la figura #1, solo se dispone de un conector de tres pines. El pin superior es el que se encarga de proporcionar los 5V de corriente continúa que necesita el circuito para funcionar. El pin inferior corresponde a GND. Y el pin central es el que entrega la señal equivalente a la recibida por el fototransistor. El circuito funciona muy bien a 9600 baudios, y posiblemente sirva para velocidades mayores.

En caso de emplear este circuito como parte de un proyecto mayor, puede conectarse el pin correspondiente a +V con la linea DTR de un puerto RS-232, que al estar en alto proporcionará la alimentación necesaria al pequeño conversor.

.La placa de circuito impreso

Como siempre, hemos creado un pequeño PCB (por sus siglas en inglés, *Printed Circuito Board*) para que no haya dificultades a la hora de "fabricar" el conversor. El diseño del mismo puede verse en la figura #2.

.Montaje

Para llevar a cabo el mismo basta con soldar los componentes sobre el PCB, utilizando como guía la figura #3.

.Lista de materiales

La lista de componentes necesarios para este proyecto no podría ser mas corta:

2 resistores de 1K, 1/8W
2 resistores de 27K, 1/8W
1 capacitor cerámico de 22nF
2 transistores BC548B
1 conector de tres pines
1 fototransistor BPW40 (o similar)

Recuerden que cualquier consulta sobre el funcionamiento o armado de este circuito la pueden hacer en el **foro uControl**. ¡Hasta la próxima!

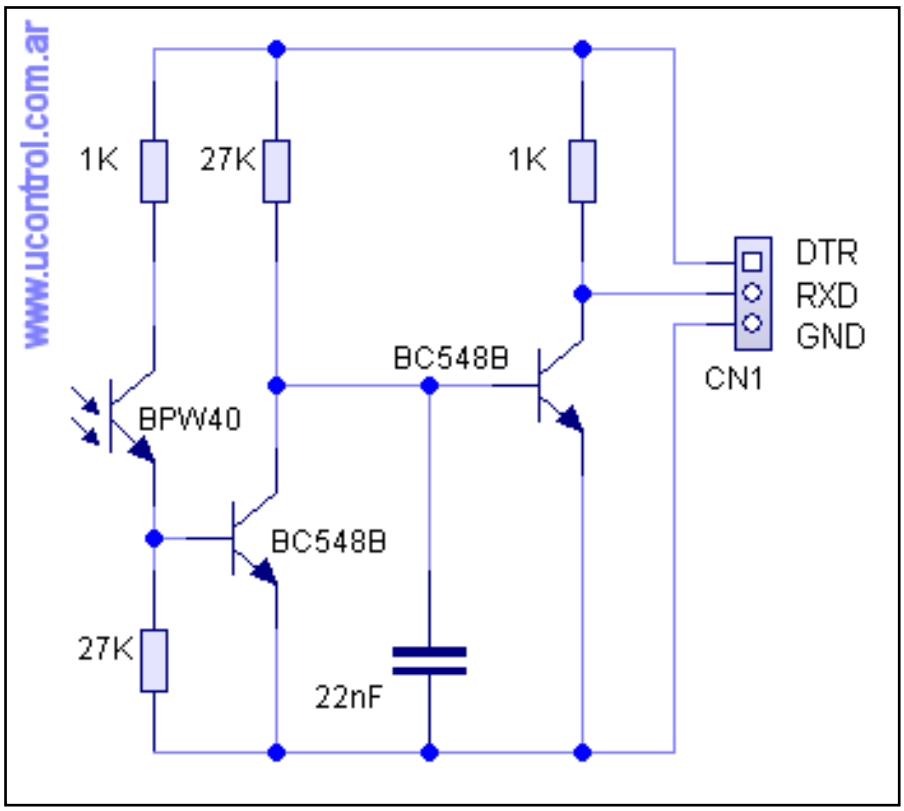


Figura #1. El circuito.

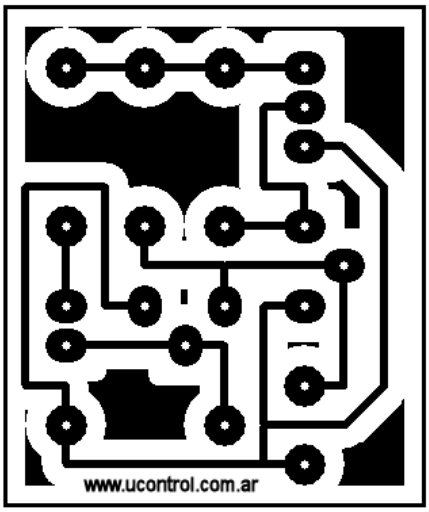


Figura #2. El PCB.

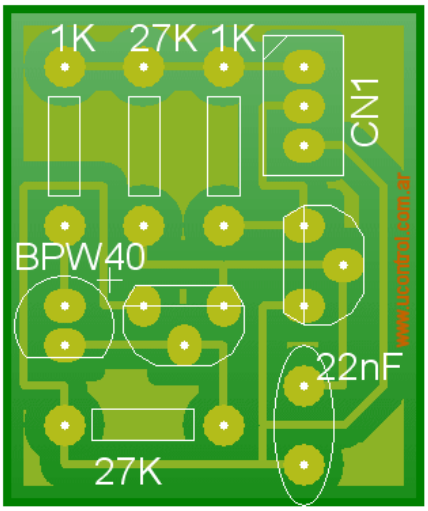


Figura #3. Ubicación de componentes.

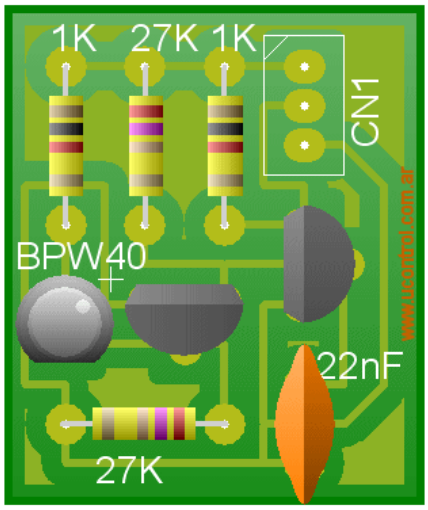


Figura #4. Modelo terminado.

uCONTROL
Electrónica en General Pics en Particular


Toda la Television
el Audio y el Video
en un solo sitio web
SERVISYSTEM
Tutoriales, circuitos
reparaciones, software
tips, samples, consejos,
foros, blog, PICs ...
www.servisystem.com.ar

el apagón analógico llegó

Es como la luz: tocas la llave y se enciende. O como la radio: la enciendes y escuchas tu emisora favorita. La televisión tal como hoy la conocemos (ese integrante más de la familia) comienza a desaparecer. El inicio definitivo de las transmisiones de señales digitales dejó hace un par de semanas a millones de estadounidenses sin televisión. ¿Y tú cómo te imaginas una vida sin televisión? ¿Estás listo para la transición? ¿Cómo es la transición en tu país? Nos interesa tu opinión acerca de este fenómeno que dejará excluida y aislada a mucha gente.

// por: Mario G. Sacco //
servisystem@gmail.com

Millones de hogares estadounidenses dejaron de recepcionar las señales de televisión el 12 de junio de 2009 cuando alrededor de 1000 emisoras de todo el país cesaron sus transmisiones analógicas y completaron la conversión de su programación al sistema de televisión digital ATSC.

El gobierno gastó más de 2000 millones de dólares para facilitar la transición a la televisión digital. Sin embargo, recién en los últimos meses se pudo reducir a la mitad el número de hogares que aún no estaba preparado para la conversión final del 12 de junio. El último estudio realizado por **The Nielsen Company** indica que, a finales de mayo, más del 10 por ciento de los 114 millones de hogares que tienen aparatos de televisión aún no estaba totalmente preparado para recepcionar las señales digitales.

Existe un abanico muy amplio de personas que quedaron totalmente aisladas a partir de este cambio, siendo los segmentos más vulnerables las familias de bajos ingresos, los ancianos, los discapacitados, las zonas rurales y los hogares donde se habla poco o nada de inglés. Los números que se manejan son preocupantes ya que aproximadamente hay más de tres millones de hogares que no tienen posibilidades técnicas o económicas de suscribirse a un servicio de cable o de recepción satelital. A esta cifra se le suman otras 9 millones de familias que, a pesar de estar abonadas a un sistema de cable, no pueden reponer todos los aparatos de TV de la casa debido a la profunda crisis que atraviesan los estadounidenses.

Otro problema que se suma a lo ya enunciado es el factor puramente técnico. Muchos serán propensos a experimentar problemas de mala recepción

por encontrarse en zonas casi marginales para la señal aérea o, en su defecto, por el mal conexionado o instalación de las antenas receptoras. Recordemos que en el caso de la TV analógica, las señales débiles provocan distorsión en la imagen, algo de nieve o ruido en pantalla y defectos causados por las limitaciones de instalación. Con la TV digital, la señal debe tener buena llegada hasta las antenas receptoras ya que el tipo de transmisión no admite una mala o pobre calidad de señal. Con los sistemas digitales se ve perfecto o no se ve nada. Por lo tanto, la topografía, el clima, el ruido eléctrico de una red de alta tensión, y hasta incluso un tránsito intenso de automóviles, pueden alterar (y hasta anular, en el peor de los casos) la recepción correcta de las señales de aire.

Entre los lugares más propensos a presentar altos índices de inconvenientes en los primeros meses se encuentra Nueva York, Los Ángeles, Chicago, Filadelfia, Miami, Boston y Dallas-Fort Worth. El estado de Puerto Rico también puede sufrir inconvenientes ya que es uno de los estados con la tasa más alta de hogares que reciben sus señales de televisión por aire. En Nueva York, el 6 por ciento de los hogares (unas 450.000 viviendas) no se encuentra aún listo para la recepción del nuevo tipo de señal, y esto significa un porcentaje preocupante de la región.

A principios de este año el Congreso estadounidense aplazó para junio la transición que tenía prevista para febrero, con el fin de insertar en el mercado otros 650 millones de dólares destinados principalmente a la compra de las cajas convertidoras (Set Top Box) que se agregan a los equipos tradicionales de TV (los analógicos) y que permiten visualizar las nuevas transmisiones digitales en los viejos aparatos. Este dinero se suma a los 1500 millones de dólares que ya había gastado la administración Bush.

Preocupado por una posible reacción política, el Presidente Obama emitió una declaración instando a los consumidores a tomar medidas para que no pierdan la recepción de televisión. *"Hemos trabajado mano a mano con los funcionarios estatales y locales, los organismos de radiodifusión y los grupos comunitarios para educar y ayudar a millones de estadounidenses con la transición"* declaró el Sr. Obama.

Hay funcionarios que levantan voces optimistas para esta transición y otros, en cambio, no tanto. *"Hay personas que esperan a último momento para todo, como si fuesen estudiantes universitarios antes de presentar sus trabajos, o las personas que pagan sus impuestos el último día de plazo". "Incluso personas como yo que esperan a último momento para hacer sus compras de Navidad"*, dijo el Secretario de Comercio, Gary F. Locke, en una reciente entrevista. Si bien aplaude los esfuerzos del gobierno, Locke dijo que estaba preocupado y apenado debido a que los primeros anuncios no proporcionaron suficiente información específica sobre los problemas que iban a llegar de la mano de la transición. Además, admitió que no se informó debidamente el hecho de que el cambio estaba llegando inexorablemente. Por último, agregó: *"Hay demasiadas personas que no saben la diferencia entre digital y analógico. Ni siquiera lo sabía yo hasta hace unos meses cuando un familiar me lo explicó".*

La conversión es el último paso en un largo plan para un uso más eficiente del espectro radioeléctrico en el país. La reestructuración vislumbra la reasignación de muchas frecuencias a una importante diversidad de servicios que están dispuestos a pagar muchos millones de dólares por los espacios que dejen libres los viejos canales analógicos. Un ejemplo de ello son las frecuencias que fueron vendidas por miles de millones de dólares, principalmente a las grandes compañías de telefonía celular, cuya demanda de espectro ha aumentado con la proliferación de dispositivos portátiles que pueden navegar por Internet y enviar y recibir correo electrónico.

.La situación actual en España

Los españoles adquieren diariamente unos 21.000 equipos de Televisión Digital Terrestre (TDT), según explica hoy Impulsa TDT (Asociación para la Implantación y Desarrollo de la Televisión Digital Terrestre) quien ha presentado los datos del último informe generado por su Observatorio que analiza la situación actual de la Televisión Digital Terrestre en España.

Según estos datos, las ventas de equipos TDT durante el primer cuatrimestre de 2009 se han elevado hasta los casi tres millones de sintonizadores o los ya mencionados Set Top Box, lo que representa un 16,2% del total de equipos comercializados hasta la fecha (18,2 millones). Tan importante como lo elevado de estas cifras, es la regularidad detectada en la adquisición de decodificadores de los últimos meses. Desde febrero de 2009 el número de dispositivos de TDT integrados se ha mantenido prácticamente estable en torno a las 400.000 unidades, de las cuales más de 300.000 son televisores con TDT integrado.

A esta constancia en las ventas de equipos TDT se le une en los primeros meses de 2009, una ampliación del nivel de cobertura que ya alcanza al 95,09% de la población española. No obstante, a menos de un mes para que empiecen a hacerse efectivos los primeros ceses de transmisiones analógicas y en un momento en el que la cobertura es efectiva para unos 43,9 millones de personas, se sigue manteniendo la comercialización de televisores analógicos, es decir, que no poseen la estructura de recepción TDT en forma nativa.

Según los datos del Observatorio, durante el primer cuatrimestre de 2009 se vendieron casi 56.000 unidades, hecho que desde Impulsa TDT se califica como incomprensible, más aún teniendo en cuenta la obligación los comercios de informar adecuadamente al comprador de un televisor analógico de su pronta caducidad.

A menos de un mes para los ceses analógicos, de los 22 Proyectos Técnicos de la Fase I en los que ya se ha confirmado la fecha (27 de junio, 30 de junio y 22 de julio) y que beneficiarán a 4,14 millones de habitantes de más de 556 municipios de toda España, el contacto medio diario con la TDT se establece en 13,9 millones de individuos, de los cuales 4,1 se han incorporado en los últimos cinco meses.

"Ciudadanos, administración y medios de comunicación, es decir TODOS debemos continuar implicados en este proceso para conseguir que el 30 de junio el proceso de transición se inicie de forma satisfactoria y España continúe su irreversible camino a la digitalización", afirmaba Andrés Armas, direc-

tor General de Impulsa TDT.

.La inalcanzable Televisión Digital en Argentina

Desde hace 10 años la discusión sobre la norma es relativa en la Argentina. Tiene algún sentido en el espectro radioeléctrico porque pueden ocurrir limitaciones. Pero teniendo un ancho de banda definido, la clave se encuentra en las conversiones digitales de la norma, que ocurre en la caja del decodificador, que es donde se concentra la verdadera guerra de tecnologías. Los funcionarios afirman estar debatiendo entre:

- La norma japonesa (ISDB-Integrated Services Digital Broadcasting), de la que Brasil hizo una adaptación local (ISDB-T).

- La norma estadounidense (ATSC, Advanced Televisión Systems Committee).

- La norma europea (DVB, Digital Video Broadcasting), que es la que más países han incorporado, comenzando por la Unión Europea que ya son 27 países, y ese volumen abarata el precio final de los decodificadores.

Mientras el debate burocrático interminable se desarrolla, DirecTV (satélite) y CableVisión (cable) pusieron en oferta 2 canales de tecnología digital adaptada a sus propias necesidades pero basadas en ATSC (el sistema americano). Por ahora es una programación minima (son servicios de alta definición de las señales Movie City y HBO), pero a 1.920x1.080 pixeles (la televisión convencional tiene 720x480 pixeles), sonido Dolby Digital 5.1 Surround, formato 16:9 (el de la TV convencional es 4:3, más cuadrada), y un decodificador (Set Top Box) con posibilidad de grabación de 100 horas HDT (DVR). Todo con salida HDMI directo para Plasma o LCD.

Además de pagar el abono que permite acceder al decodificador DVR, es necesario contar con un televisor LCD o plasma (HD Ready / Full). No es un producto masivo todavía pero permite comenzar a promover el cambio de los decodificadores, popularizar la tecnología, fidelizar al cliente. Por su parte, DirecTV se propone que el 20% de la base de abonados tenga ese servicio en los próximos 3 años, en el marco de un aumento promedio anual del 16% en su clientela.

"La oferta de contenido en Alta Definición (HD) para Latinoamérica, es un segmento en pleno desarrollo", declaró Jacopo Bracco, ejecutivo de DirecTV Panamericana. Básicamente el deporte, y la clave de esa estampida será el Mundial Sudáfrica 2010.

.Los números mandan

Hay cuestiones que no pueden discutirse, acerca de cómo es la integración socioeconómica del mercado televisivo argentino y del resto de latinoamérica. Por ejemplo, el 30% ó 20% de los hogares que no acceden hoy día a la TV por abono no interesa a la industria publicitaria, y tampoco al negocio de la TV HDTV porque, en teoría, no pueden adquirir el nuevo televisor plasma y el decodificador necesario, que difícilmente pueda subsidiar el Estado, en especial en el nuevo contexto fiscal global.

En cuanto al universo alcanzado por la TV por abono (ya sea la TV por cable o el sistema satelital de DirecTV), quienes pueden pagar el acceso a la nueva tecnología, ya han optado sin importarle qué ocurrirá con el ente regulador de las emisiones radiotelevisivas.

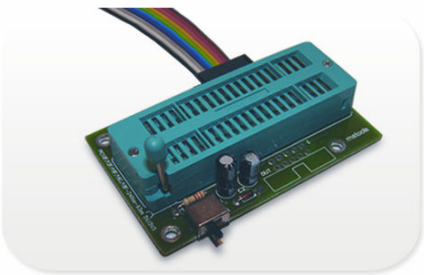
Es una demostración más de que en los países en vías de desarrollo, el mercado se impone a los burócratas, en especial cuando ellos se dilatan en debates interminables.

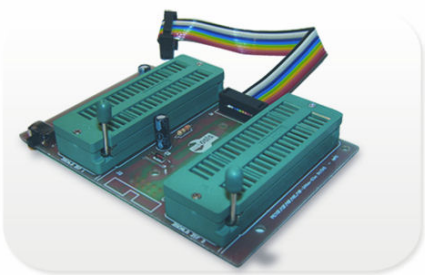
GTP-USB+

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

primer congreso virtual de microcontroladores

Sin mesas, sillas, micrófonos, cañón proyector o coffe break. Para participar del Primer Congreso Virtual de Microcontroladores no hará falta pagar pasajes, viajar durante horas, ni alojarse en ningún hotel. En definitiva, si de tecnología se trata, no podían estar ausentes las herramientas que la informática pone al alcance de la mano, todo al servicio de una original iniciativa.

// por: Germán Reula //
gerreula@yahoo.com.ar

"La idea es conformar una gran comunidad de desarrolladores, unir a las empresas con las universidades y desarrolladores independientes"

"La principal ventaja que se obtiene al desarrollar un congreso virtual, es que cualquiera puede participar, sin las limitaciones que supone en tiempo y dinero el tener que trasladarse. De hecho, tenemos inscriptos de España y de 18 países de toda América, además por supuesto de los representantes de Argentina" explican los impulsores de la novedosa iniciativa, gestada desde las Cátedras de "Técnicas Digitales" de la carrera de Ingeniería en Electrónica de la Facultad Regional Paraná de la Universidad Tecnológica Nacional (UTN).

.El congreso

Si bien en la actualidad se realizan otros congresos sobre tecnología, proyectos de investigación y microcontroladores, los organizadores evaluaron que muchas veces estos están fuera del alcance de los estudiantes, docentes y desarrolladores independientes, de aplicaciones con microcontroladores.

"Intentamos entonces, a través de esta propuesta, romper las barreras geográficas y unir en este congreso a desarrolladores de cualquier región, abriendo canales de participación para la comunicación de las numerosísimas experiencias realizadas en las distintas instituciones educativas, empresas y por desarrolladores particulares" se explicó.

. Eureka

La idea nació en diciembre pasado. El Ingeniero Raúl Manuel Caballero, quien tiene a cargo la cátedra, planteó la idea de organizar un congreso virtual, basado en su experiencia al haber participado en otras iniciativas de estas características.

“Siempre buscamos hacer cosas nuevas, porque la asignatura es propicia para el desarrollo de proyectos. No es común unir en un único ámbito a empresas, universidades o aficionados. Por eso nos planteamos que la participación debía ser sin costos, pues de lo contrario habría una limitación para muchos participantes”.

La idea es conformar una gran comunidad de desarrolladores, unir a las empresas con las universidades y desarrolladores independientes. Se pretende que, como su nombre lo dice, este sea el primero de muchos congresos virtuales y que año a año más desarrolladores y empresas se sumen a la propuesta.

Como todos los congresos, existe un plazo para presentar los trabajos. Solo que en este caso, son ponencias virtuales. Los usuarios deben preinscribirse ingresando a la página *www.frp.utn.edu.ar/congreso*, una vez registrados podrán enviar sus ponencias, teniendo como fecha limite para esto el 30 de Julio. Un comité evaluará las mismas y comunicara a los autores su publicación.

.Quienes pueden participar

Cualquier persona, desde cualquier país o región, puede participar de este evento, solo es necesario tener acceso a Internet. A la fecha el número de inscritos es de 500, entre los cuales hay Ingenieros, Técnicos, investigadores, docentes, estudiantes, empresas y desarrolladores independientes de 19 países de América y Europa. Se espera que para la apertura del congreso el número de inscriptos supere los 800.

La participación puede ser en calidad de Asistente o de Ponente y para ello deberán realizar su Pre-Inscripción y esperar la confirmación de parte de la Organización para el acceso al área de debates.

.Avales universitarios

El Congreso ha sido declarado de Interés Universitario por Universidad Tecnológica Nacional según resolución 285/09 de Consejo Superior. También cuenta a la fecha con el aval institucional de los Consejos Académicos de la Facultad Regional Paraná de la Universidad Tecnológica Nacional, según Resolución 059/09, de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral, según Resolución 141/09, del Decanato de la Facultad de Ciencia y Tecnología de la Universidad Autónoma de Entre Ríos y recientemente se sumo el apoyo de la Universidad Abierta Interamericana.

.Metodologia

Se decidió establecer seis líneas temáticas sobre las que se centraran las aplicaciones. Comunicaciones, Control de Potencia e Industrial, Transporte, Aplicaciones Hogareñas, Aplicaciones Ecológicas y por supuesto Aplicaciones Académicas son las áreas sobre las cuales los ponentes basaran sus publicaciones.

El 14 de setiembre, a las 19 (horario de Argentina), se realizará la apertura del

congreso. Los inscriptos tendrán acceso a las publicaciones y quedarán abiertos los foros de debate, uno por cada línea temática.

Cada trabajo presentado tendrá un tópico específico en los foros de debate. El los autores de las ponencias responderán a las consultas que le realicen los participantes. Eso es un compromiso que deben asumir todos los ponentes. Todos los asistentes y ponentes, tendrán la posibilidad de participar de todos los espacios de debates así creados.

El 30 de Setiembre se producirá el cierre del congreso. Se entregarán certificados de asistencia a los asistentes, y de participación a los ponentes.

.Preinscripción abierta

El congreso se realizará desde el 14 al 30 de septiembre del corriente año. Durante estas dos semanas, instituciones, docentes, alumnos, empresas y desarrolladores intercambiaran experiencias sobre diseño, uso e implementación de aplicaciones y programas académicos realizados con microcontroladores.

La preinscripción se encuentra abierta, pudiendo recabarse más datos *en la página web del congreso www.frp.utn.edu.ar/congreso* o via correo electrónico a *congreso.microcontroladores@gmail.com*.

Una vez registrados en el sitio, se podrán enviar las ponencias al congreso. Los asistentes y ponentes deberán esperar la confirmación de parte de la organización para el acceso al área de debates.

µCONTROL
Electrónica en General Pics en Particular

Commodore Amiga: un ordenador legendario

El Commodore Amiga fue un ordenador personal con extraordinarias capacidades multimedia de gran éxito en las ultimas dos décadas del siglo pasado. Fueron comercializados entre 1985 y 1994, y su bajo precio sumado a sus características multimedia mucho más avanzadas que los PC de la época lo hicieron el favorito de los amantes de los videojuegos.

// por: Ariel Palazzesi //
arielpalazzesi@gmail.com

La historia del Commodore Amiga comienza a principios de los años 80s, con la creación de una empresa llamada Hi Toro en Los Gatos, California, financiada con fondos aportados -según la leyenda- por un grupo de adinerados dentistas de Texas. Su primer presidente fue Dave Morse, y el objetivo de Hi toro era crear "la maquina de videojuegos definitiva", una consola que les permitiese quedarse con la parte del león del mercado de los videojuegos, que en aquella época estaba en manos de Atari y su consola de 8 bits "2600".

"Las características básicas del Amiga 1000 de 1985 incluían un Motorola 68000 de 32 bits funcionando a 7Mhz, 512KB de RAM (ampliables a 8MB), interfaz gráfica y multitarea preemptiva"

Justamente, el creador de la consola Atari 2600 y del ordenador Atari 800, Jay Miner, que recientemente había renunciado a esa empresa por que sus directivos no querían abandonar el microprocesador 6502 como "cerebro" de sus nuevos productos, fue contratado por Hi Toro para llevar adelante el proyecto.

Miner sugirió que se diseñase una maquina basada en un microprocesador de 32 / 16 bits -el potente Motorola 68000- y que eventualmente pudiese expandirse para ser utilizada como un ordenador. Se formó un equipo de trabajo, y poco tiempo después -por sugerencia de los inversores- se cambio en nombre de la empresa a "Amiga". Algunos creen que se decidieron por este nombre en español por que, además de su significado, estaba alfabéticamente ubicado antes que Apple o Atari, sus competidores.

Mientras que Amiga diseñaba su consola, el mercado de los videojuegos comenzaba a hacer agua frente a la expansión de los nuevos ordenadores domésticos (LINK). Eran épocas en que hacían furor los Apple-II (el primer ordenador personal con gráficos a color), el Commodore PET y el Radio

Shack (o Tandy) TRS-80, todos con 4Kb de RAM. Muchos fabricantes de equipos electrónicos "tradicionales" lanzaban modelos propios, como Texas Instruments con el TI 99/4, Sinclair con su ZX-80 y ZX-81, Commodore con el VIC-20. Uno o dos años más tarde todo explotaría con la aparición del Sinclair Spectrum, los MSX y el Commodore 64. Los ordenadores ya tenían color y sonido, y sus juegos eran tan buenos -o incluso mejores- que los de muchas consolas.

Frente a este panorama, los inversores de Amiga comenzaron a pensar que quizás fuese una buena idea reformar el diseño de su videojuego original para convertirlo en un ordenador. De esta manera, el equipo de Jay Miner empezó el diseño del auténtico Amiga, de nombre clave "Lorraine" (que "casualmente" era el nombre de la mujer de Dave Morse). Entre los usuarios ya comenzaba a sonar insistentemente un rumor que vaticinaba la aparición de un ordenador con increíbles capacidades gráficas, sonoras y una cantidad de memoria nunca vista. A finales de 1983 se terminaron de diseñar los tres chips de apoyo del 68000, y una primer versión del ordenador se presenta en el Consumer Electronics Show (CES) que tendría lugar en Las Vegas en Enero de 1984. En ese momento, se terminaron los 7 millones de dólares de la inversión inicial.

El ordenador en realidad era un amasijo enorme de cables y chips. Lo que finalmente serian tres circuitos integrados "custom" -Agnus, Daphne y Portia- eran todavía tres grupos de ocho placas cada uno, cableadas entre si y colocadas dentro de tres torres. Durante el CES solo se mostró el prototipo en privado, haciendo exhibiciones dentro de una pequeña habitación. Quienes tuvieron la suerte de verlo en funcionamiento quedaron alucinados por sus prestaciones, y la empresa recuperó la esperanza de poder sobrevivir económicamente. Consiguieron el dinero necesario para fabricar versiones de silicio de Agnus, Daphne y Portia, y volvieron en junio al CES de Chicago. Esta vez, el ordenador tenia pinta de ordenador y se mostró al público en general. Algunos aseguran que la gente miraba debajo de la mesa en que estaba el Amiga buscando el "gigantesco ordenador que estaba generando esos gráficos y sonidos". Los rumores sobre el "super ordenador para juegos" eran ciertos.

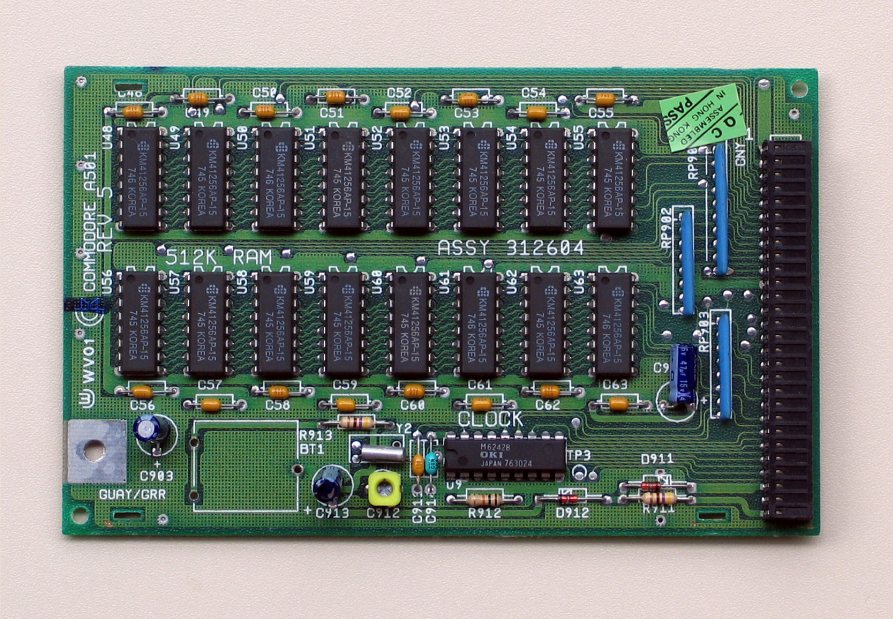
Commodore Amiga 500.

Lamentablemente, la situación económica de Amiga luego del esfuerzo de fabricar el primer modelo era pésima. Después de intentar un acuerdo como gigantes como Sony, Philips, Apple, Hewlett Packard o Silicon Graphics, y un fracasado intento de compra por parte del recientemente expulsado director de Commodore Jack Tramiel, la empresa Commodore Bussiness Machines se quedó con Amiga. Era el año en que IBM deslumbraba al mundo con su PC AT, basado en el chip 80286 de Intel, una unidad de disco 5"1/4 de 1'2 MB y 256 KB de RAM, que sin monitor ni tarjeta de vídeo costaba 4.000 dólares.

El 23 de junio de 1985 se presenta en el Lincoln Center de Nueva York el

Amiga 1000, primer modelo comercial de esta estupenda máquina, con el lema "¡Sólo el Amiga lo hace posible!". Andy Warhol, uno de sus más famosos y apasionados usuarios, creo delante del publico presente el retrato de Debbie Harry, la vocalista del grupo "Blondie". B.B.King, Arthur C. Clarke, y una larguísima lista de artistas utilizaron modelos de Commodore Amiga para desarrollar sus trabajos.

Las características básicas del Amiga 1000 de 1985 incluían un Motorola 68000 de 32 bits funcionando a 7Mhz, 512KB de RAM (ampliables a 8MB), interfaz gráfica y multitarea preemptiva (algo que los usuarios de IBM PC conocerían recién en 1991 gracias a Linux). Todo esto por solo 1295 dólares, la tercera parte del costo de un “IBM AT”. El secreto de Amiga estaba en los tres chips de soporte creados especialmente por Jay Miner. El chip de vídeo era capaz de manejar 32 colores (de una paleta de 4096) con una resolución de 320×200, que podían mejorarse aprovechando características poco documentadas de estos chips. Poseía aceleración de vídeo por hardware (copiar bloques, dibujar líneas y rellenar sólidos). Todo esto en una época que el mundo del PC utilizaba casi exclusivamente monitores de fósforo verde.



Ampliacion de memoria Commodore A501.

En cuanto al sonido, presente en los IBM compatibles mediante un pequeño parlante capaz de hacer poco más que “beep”, el Amiga disponía cuatro vías sonidos digitalizados en estéreo (2 por canal), que funcionaban de forma totalmente independiente del procesador, incluso accediendo a la RAM por sus propios medios. Esto le permitía proporcionar musica de calidad y efectos de sonido a los videojuegos sin sacrificar velocidad.

Aún hoy muchas empresas dedicadas a la tratamiento de imágenes para la TV emplean Commodore Amiga para subtitular sus producciones. Es que este ordenador se diseñó tomando como base la señal de TV, y los modelos equipados con AGA (Advanced Graphic Architecture, o arquitectura gráfica avanzada). Como las

Commodore Amiga 1200.

Amiga A1200 o A4000 -que se vendieron en 1992- soportaban gráficos de hasta 724x566 píxeles en PAL o 1472x566 píxeles con hasta 256 colores reales. Mediante algunos trucos de programación, como la modificando la paleta de colores en cada línea de barrido, Amiga puede mostrar 262144 colores de una paleta de 16 millones.

Todo esto hizo del Commodore Amiga un ordenador que marcó una época. Aún hoy existen usuarios que cada día encienden su A500 y juegan una partida de su videojuego favorito, y en los sitios de subastas luchan por conseguir una de estas máquinas. ¿Tu eres uno de ellos?

Commodore Amiga CDTV.

Fuente: Neoteo

AmigaOS4, una de las últimas versiones de este sistema operativo.


µCONTROL
Electrónica en General Pics en Particular