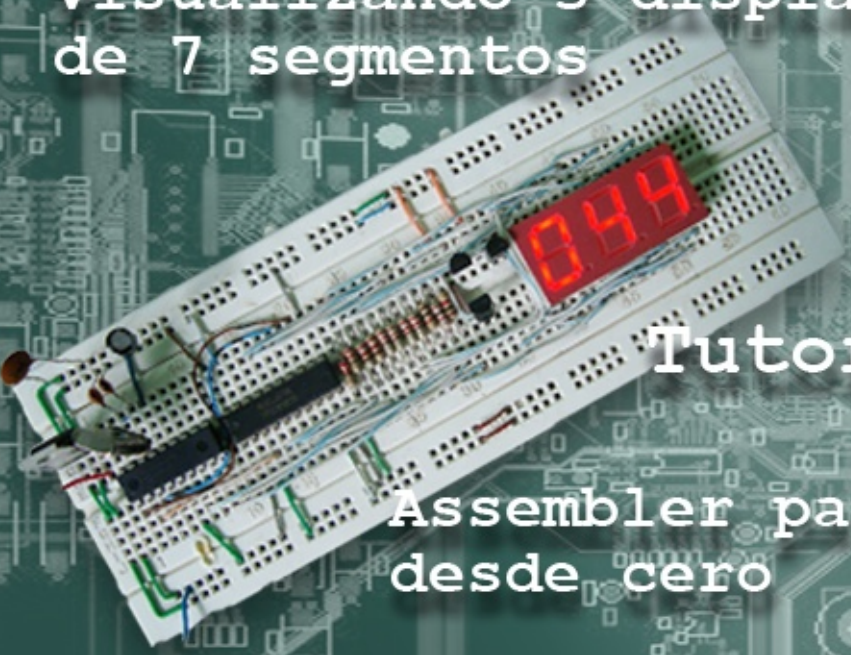
// DESCARGA GRATUITA //

"int año = 3, número 10 / DICIEMBRE 2010

µCONTROL

Electrónica en General Pics en Particular

Visualizando 3 displays de 7 segmentos

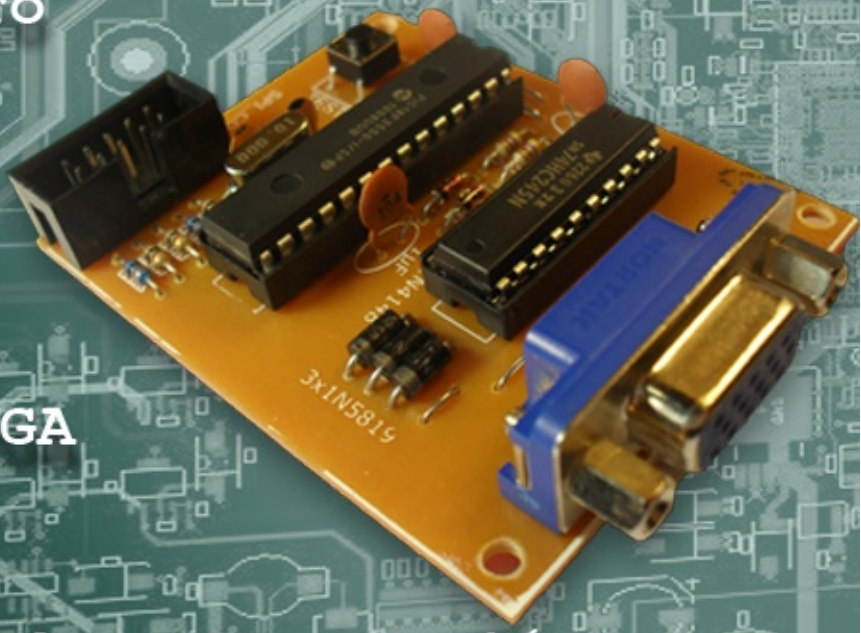


Control PID Nivel básico

Tutorial MPLAB C18

Assembler para PIC desde cero

Como desechar cloruro férrico



Generar señal VGA con un PIC

Tips de programación en assembler para PIC

Receptor de mandos infrarrojos

.staff

número = 10; año = 3;

Dirección, Redacción y Corrección:
uControl
www.ucontrol.com.ar

Diseño y Diagramación:
Alejandro Casanova,
Lucas M. Treser
y xocas
inf.pic.suky@live.com.ar
lmtreser@gmail.com

Consejo Editorial:
Ariel Palazzesi
arielpalazzesi@gmail.com

David (Leon Pic)
david@meteorologiafacil.com.ar

Emiliano Safarik
fluf@adinet.com.uy

Felixls
sergiols@keko.com.ar

Gabriel Gabarain
gabriel@glab.com.ar

Hector Javier
hj.0x00@gmail.com

Mario Sacco
service.servisystem@gmail.com

Maximiliano Martin Simonazzi
admin@maxisimonazzi.com.ar

Miguel Ángel Borrego Trujillo
jimjim17@gmail.com

Miguel A. Lopez O.
miguel_626@hotmail.com

Rodrigo Flores Calle
rodrigo@ielectronica.com

.indice

.editorial

Es la décima vez que me siento a escribir un editorial para que sirva de “prólogo” de la Revista uControl. Y al igual que las otras nueve veces, no se por donde empezar. Afortunadamente, este dilema se debe a la abundancia de buenos temas que deberían ser resaltados en estas lineas, y no al bochornoso hecho de disponer solamente de “material de segunda” que no me permitiría escribir nada bueno sobre el contenido de estas páginas.

Esta vez -tarde, como ya es nuestra costumbre- hemos armado una selección de artículos que estamos seguros van a gustarte. Empezamos por una nota ecológicamente impecable: ¿Como debemos desechar el cloruro férrico que utilizamos para construir nuestros PCBs? Si haces tus propias placas de circuito impreso, ese es tu artículo. Los que han intentado realizar algún proyecto en el que se muestren datos mediante displays LEDs de 7 segmentos habrán luchado con los problemas relacionados con su multiplexado. El artículo de Rodrigo Flores Calle les evitará dolores de cabeza en el futuro. Y si programas tus microcontroladores PIC en ASM, los “tips” propuestos por Hector Javier y la nueva entrega del tutorial de David te resultarán muy útiles.

Pero hay mucho mas para leer: Miguel Angel Borrego Trujillo ha preparado un receptor de mandos infrarrojos, Miguel A. Lopez O. nos explica en que consiste un Control P.I.D Básico, y Alejandro Casanova nos trae la segunda parte del Tutorial MPLAB C18. El último articulo incluido en este número permite dotar a nuestros proyectos de una salida directa a un monitor VGA, mostrando texto en 16 colores. Todo material de primera, sin una sola página de desperdicio.

La calidad del contenido de esta precaria revista, construida mediante muchas horas de esfuerzo desinteresado de los integrantes del staff, no es casualidad. Las personas que participan de este proyecto no solo tienen los conocimientos necesarios como para exponer temas relacionados con la electrónica de forma simple y amena, sino que son excelentes personas con la que es un verdadero placer trabajar. A todos ellos, nuevamente, muchas gracias.

Nos vemos en el número 11.

Cómo desechar cloruro férrico

Mucho se habla en internet sobre como desechar el cloruro férrico o como no hacerlo, sin embargo, casi ninguna de las voces explica idealmente como debe hacerse o por qué, es por ello que nos pusimos a investigar para saber como desechar este compuesto adecuadamente tratando de causar el menor impacto posible al medio ambiente.

// por: Maximiliano Martin Simonazzi //
admin@maxisimonazzi.com.ar

El cloruro de hierro (III) o tricloruro de hierro (comúnmente llamado cloruro férrico o mal llamado percloruro férrico o percloruro de hierro) es un compuesto químico utilizado industrialmente en muchas aplicaciones. En electrónica se lo utiliza para la realización de plaquetas y generalmente se lo utiliza en su estado líquido o si no como un polvo diluido en agua. Su color es rojo purpúreo cuando trasmite la luz y sus cristales tienen un color verde oscuro cuando reflejan la luz.

Es muy probable que toda persona que se haya dedicado a la creación de plaquetas electrónicas haya tenido contacto o esté usando este químico para realizarlas. Luego de un tiempo de usarlo este líquido pierde sus propiedades y ya no puede ser usado para la tarea que se usaba originalmente por lo cual debe ser desechado, y es aquí cuando todos nos preguntamos ¿Cuál es el método adecuado para desechar estos residuos?

Es importante saber antes que nada como interactúa el Cloruro Férrico en presencia de otros compuestos. Por su carácter covalente este es soluble en disolventes orgánicos. En disolución alcohólica se lo conoce como tintura de hierro. Cuando se disuelve en agua, el cloruro férrico sufre hidrólisis y libera calor en una reacción exotérmica. De ello resulta una solución acida y corrosiva de color marrón que se utiliza como coagulante en el tratamiento de aguas residuales, para la potabilización del agua, y en la industria electrónica para el grabado químico de plaquetas de circuito impreso. Al disolverse en agua, debería precipitar formando hidróxido de hierro (III), insoluble; sin embargo, forma una disolución coloidal de ese compuesto, que presenta el típico color pardo de las disoluciones de sales de hierro (III). El cloruro férrico anhidro es un ácido de Lewis bastante fuerte, y se emplea como catalizador en síntesis orgánicas.

Visto lo anterior solo vamos a tomar el ejemplo del cloruro férrico disuelto en agua que es el que realmente nos interesa porque es el que se usa en electrónica.

Precauciones:

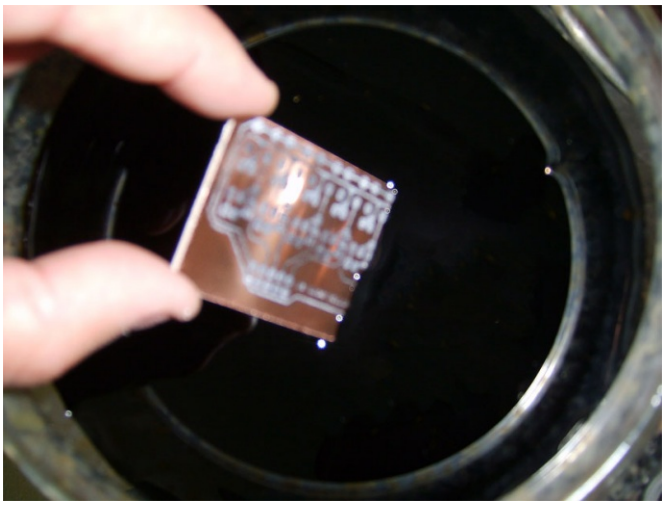
En este estado no es muy fuerte y no quema al contacto con la piel, aun así su contacto prolongado puede irritar levemente las capas superiores de la piel. Al contacto con heridas o los ojos se siente un ardor intenso debido a las quemaduras que produce, urgentemente hay que limpiar la

zona con abundante agua durante 10 minutos o más. Una vez que la piel fue expuesta al cloruro férrico es muy probable que durante un tiempo (1 semana aproximadamente) la misma quede coloreada de un color marrón bien característico, sobre todo en la parte de la unión de las uñas con la piel. Es muy recomendable el uso de guantes durante su manipulación.

Hay que tener cuidado con la ropa ya que la tiñe al instante y por más que se la lave va a quedar la mancha en la tela.

Es muy corrosivo para los metales ferrosos y algunos metales no ferrosos como el cobre y sus aleaciones, es por ello que se debe evitar el contacto del mismo con cualquier tipo de metal.

Cuando lo hacemos reaccionar con el cobre de una plaqueta, este reacciona formando Cloruro Ferroso y Cloruro Cúprico. Para obtener mejores resultados, la solución debe ser calentada a 25º previamente o simultáneamente en un baño calefactor. Si la solución se encuentra en buen estado la plaqueta debería flotar sobre la solución. También es recomendable usar una fuente de aire en el recipiente donde vaya la disolución para que de esta manera se oxigene y el ataque sea más efectivo.



Luego de haber sido usado durante un tiempo, veremos un precipitado en el recipiente donde se alojaba el cloruro férrico, esto es cloruro cúprico. Mientras más cloruro cúprico haya en el recipiente menor será el poder de ataque de la disolución. Llegado cierto punto debe ser desechado adecuadamente.

Para no dar más vueltas sobre el tema, fuimos directamente a contactar a un responsable de una de las empresas más conocidas en Argentina que comercializa este producto, Electroquímica Delta. En esta ocasión nos pusimos en contacto con el Ing. Leonardo Eidelson el cual nos explicó detalladamente como debe ser la disposición del compuesto:

"La explicación es general, si bien cada distrito o provincia tiene su propia reglamentación sobre el tratamiento de desechos; tratándose de un producto poco peligroso, le sugerimos lo siguiente:

El desecho del cloruro férrico usado puede hacerse por vía de desagüe de líquidos diluyéndolo en gran cantidad de agua, no menor a 50 litros, por litro de cloruro férrico.

Si las cantidades son mayores, agregar al recipiente con el producto usado, una pequeña cantidad de hidróxido de calcio (cal) para neutralizar el producto, con mucho cuidado y con guantes y gafas por si se generan salpicaduras.

El agregado de la cal debe hacerse de a poco, ya que se eleva la temperatura y se debe esperar entre agregado y agregado, mezclando con cuidado con varilla de plástico o madera.

La cantidad que se debe añadir es hasta conseguir que el líquido se decolore. Cuando termina el agregado se deja reposar hasta que decante (se va a depositar un sólido gelatinoso, mezcla de hidróxidos de hierro y cobre). El líquido se vierte en el sumidero haciendo circular abundante agua. El resto se desecha como residuo sólido."

Sin más que decirles, esperamos que esta nota les haya sido útil a todos ustedes y que utilicen estos métodos para cuidar un poco más el lugar en el que vivimos.

Visualizando 3 displays de 7 segmentos mediante registros de desplazamiento.

Utilizar display de 7 segmentos para representar datos numéricos de forma visual puede llegar a ser algo imprescindible en nuestro proyecto. En esta nota presentamos una de las tantas formas de hacerlo, utilizando registro de desplazamiento y multiplexado, con el fin de ahorrar pines utilizados por el microcontrolador. Además evaluamos entre 3 posibilidades de registros de desplazamientos.

// por:Rodrigo Flores Calle //
rodrigo@ielectronica.com

Normalmente utilizamos un display de siete segmentos para visualizar datos en forma de números, ya sea un display de ánodo o cátodo común (colocando el pin común a donde corresponde) encendemos o apagamos los segmentos correspondientes para visualizar un número en particular.

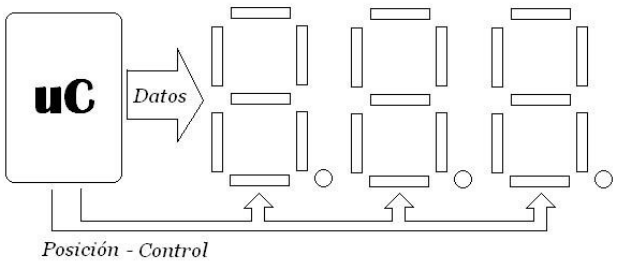
Podemos usar un decodificador de BCD a 7-segmentos tal como el 74LS47, 48 o un CMOS CD4511 y obtener el número que deseemos en el display.

Lo que en éste artículo pretendemos, es mostrar otra forma de poder visualizar varios displays de 7 segmentos, utilizando menos pines del microcontrolador. Para este fin recurriremos a un registro de desplazamiento, existen muchos en el mercado, tales como el 74LS164, CD4094, 74HC595, etc. Aquí trataremos principalmente el TPIC6595 de Texas Instruments y luego se comparará con otros, porque la mayoría de los registros no pueden hacer circular la corriente necesaria para encender led's a 20-25mA, que es la corriente óptima para poder tener un buen brillo en ellos. Algunos registros entonces necesitarán de una circuitería adicional para suministrar a los leds la corriente óptima para el mejor brillo, tales como transistores o arreglos de transistores como los ULNxxxx.

A la vez visualizaremos varios displays, serán 3 en realidad, para ello multiplexaremos los datos de tal manera que podamos ver los 3 displays encendidos, utilizando solamente 6 pines de un microcontrolador. Vamos a trabajar con el PIC16F88, por ser uno de mis preferidos, en mi opinión, el mejor de los 16F de 18pines.

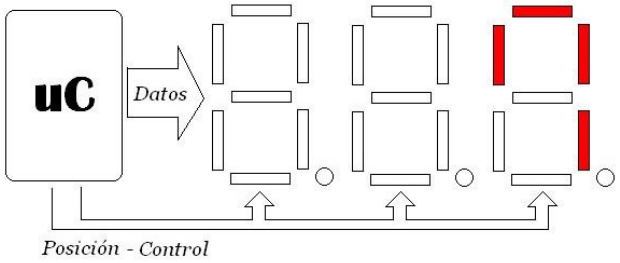
Para aquellos que no conozcan muy bien el efecto del multiplexado que buscamos acá, lo explicaremos brevemente:

Multiplexar, para este nuestro caso, significará enviar por el mismo bus de datos, información para 2 o más dispositivos, entonces tendremos un hardware como este:



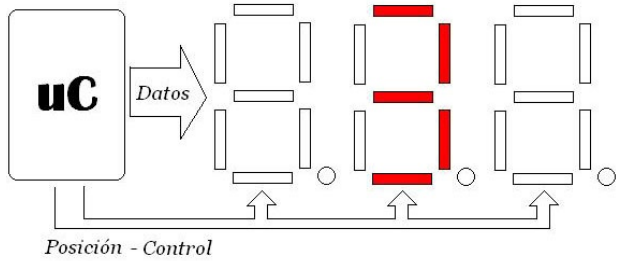
Lo que podemos interpretar de la imagen es que tenemos un bus de datos que sale del microcontrolador hacia los displays, y que es el mismo, para todos los displays. De la misma manera tenemos un bus de control, que nos encenderá un display a la vez, a una velocidad que el ojo humano no percibe

parpadeos y ve los tres displays encendidos y mostrando cualquier número o dato independientemente del otro, por ejemplo para visualizar el número 1.37 hacemos:

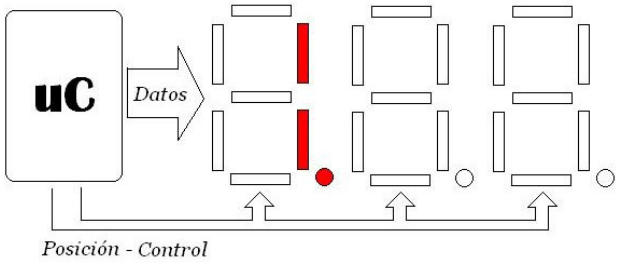


Mediante el bus de control o posición activamos el primer display de la derecha (posición 1), por tanto los otros dos están desactivados, enviamos los segmentos a encender por el bus de datos, estos datos los reciben todos los displays, pero solamente uno lo mostrará.

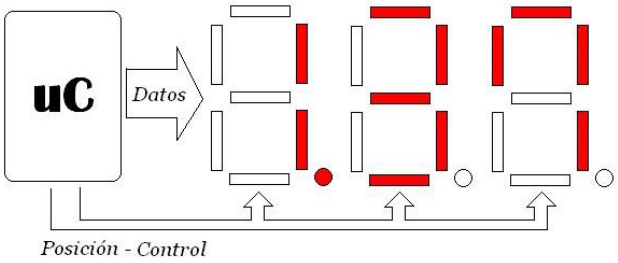
Hacemos lo mismo para el siguiente número a mostrar, será el display del medio (posición 2), por el bus de control solo se activará éste:



Por último el display de la izquierda (posición 3), con el dato y la posición en el bus de datos y control respectivamente.



Este proceso el microcontrolador lo hace a una frecuencia dada que nos dará como resultado visualmente en el ojo humano los tres displays encendidos, también tiene que ver con un efecto en los leds, y es que no tienen el suficiente tiempo para apagarse completamente.

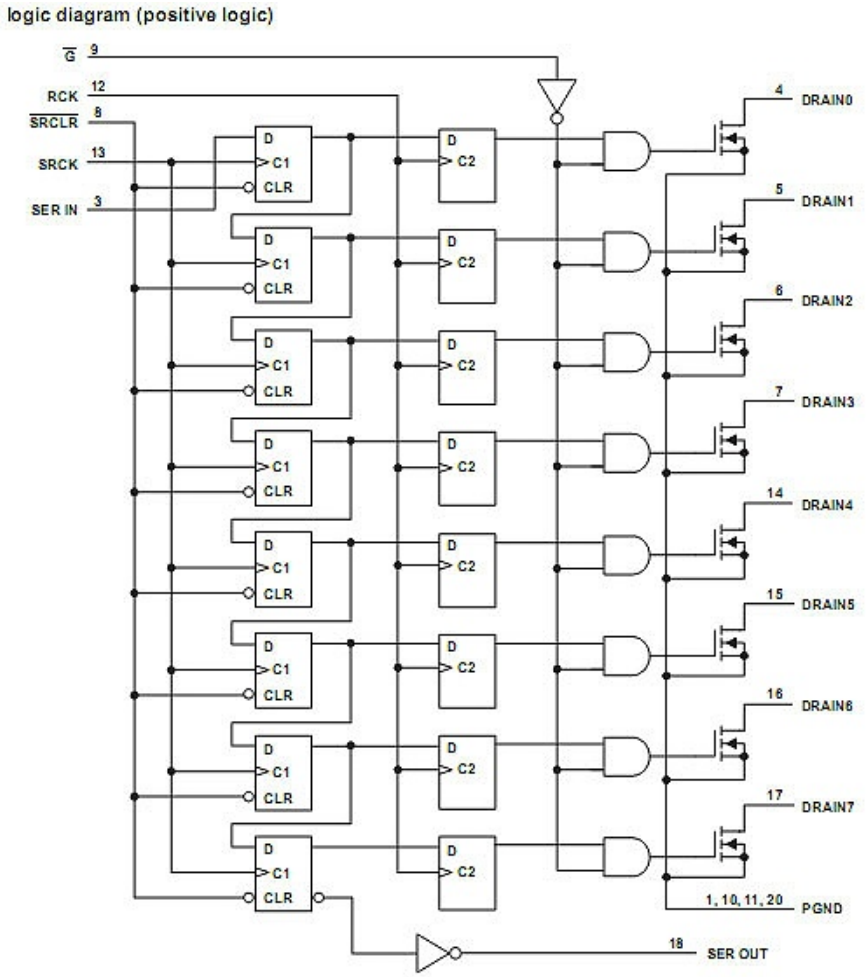


Nuestro propósito ahora es el ahorro de pines, ya que, para un fin como este, podemos utilizar solamente el microcontrolador para las salidas del bus de datos hacia los displays directamente, lo que nos lleva a usar un puerto completo de 8bits, porque un bit corresponderá a un segmento del display (8 por adicionar el punto decimal), y más un bit por cada display existente, para el bus de control o posición.

Tenemos entonces, en nuestro caso, 8 + 3 = 11 salidas necesarias de nuestro microcontrolador, que si disponemos de un micro con pocas salidas y que quizás adicionalmente se trabaje con algunos otros componentes, como pulsadores, sensores, etc. los pines ahorrados son muy valiosos.

Para ello recurrimos a un registro de desplazamiento, recibe los datos serialmente y los muestra en forma paralela, es por eso que usaremos el TPIC6595 de Texas mencionado anteriormente, pues este es un registro de desplazamiento, con las salidas con MOSFET's que pueden entregar la corriente necesaria para tener un brillo óptimo en los leds del display, ya que muchos de los registros de desplazamiento no son capaces de suministrar la corriente necesaria, como describimos anteriormente.

Es muy importante tener la hoja de datos del TPIC6595 a mano, para poder observar sus características y funcionamiento. Podemos ver entonces, el principal diagrama para poder manipular el integrado.



El **Pin 9**, de nombre **G**, es el Enable, se activan las salidas del registro, con un estado bajo en este pin.

El **Pin 12**, de nombre **RCK**, se encarga de pasar los datos cargados serialmente en registros anteriores, mediante un pulso a nivel alto.

El **Pin 8**, **SRCLR**, es el Reset de los registros donde se ha ingresado previamente los datos serialmente y esperan a ser pasados a las salidas mediante el pulso en el pin descrito anteriormente. SRCLR coloca a cero todos estos datos previos (mediante un pulso a nivel bajo) antes de ser pasados a las salidas.

El **Pin 3**, **SER IN**, es la entrada de datos seriales.

El **Pin 13**, **SRCK**, es el reloj para el ingreso de datos por el pin 3, detecta los datos en el flanco de subida.

Los **Pines 4 a 7 y 14 a 17**, **DRAIN0 – DRAIN7**, son las salidas a drenador abierto de los mosfets correspondientes a cada salida

El **Pin 18**, **SER OUT**, es el mismo valor del bit7 ingresado, que sirve para conectar en cascada varios registros de este u otro tipo.

Por último los pines de alimentación y la particularidad de los pines de tierra **LGND y PGND** (LogicGND y PowerGND respectivamente), y como su nombre dice, Logic se refiere a las tierras de los flip-flop y compuertas internas que tiene el registro, y Power a la tierra de las salidas de Mosfet. Se puede trabajar entonces con 2 tierras distintas para aislar eléctricamente ambas etapas, esto puede ayudar a manipular quizá cargas que requieran algo extra de consumo de corriente, como Relés por ejemplo u otro dispositivo. O se puede trabajar con una misma tierra cuando no es indispensable, es decir las cargas de las salidas no son pesadas, como en nuestro caso, entonces se realizan los puentes respectivos entre estos pines; LGND y PGND.

El lenguaje de programación que usaremos es C de CCS Compiler. Una posible rutina que podemos utilizar para enviar los datos de forma serial al registro de desplazamiento puede ser la siguiente:

```
void cargar(int datob){
  int contador;

  output_low(srck);
  delay_cycles(2);
  output_low(rck);
  for(contador=0;contador<8;contador++){
    output_bit(ser_in,(datob & 0x1));
    datob>>=1;
    output_high(srck);
    delay_cycles(1);
    output_low(srck);
  }
  output_high(rck);
  delay_cycles(1);
  output_low(rck);
}
```

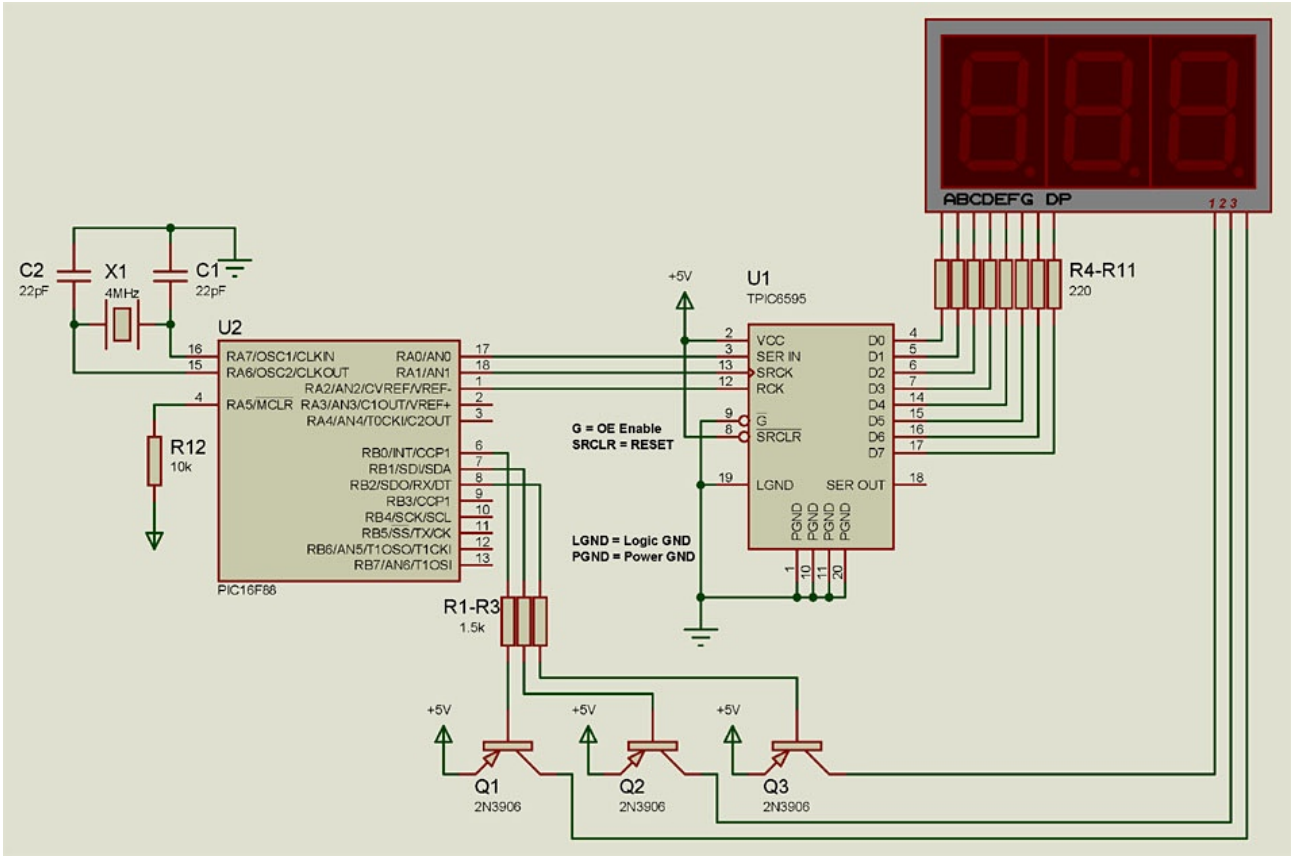
En la cabecera de nuestro programa se definirán los pines correspondientes del micro para el bus de control hacia el TPIC6595.

Se utilizará displays de ánodo común, con las respectivas resistencias para protección de los Leds de los displays, y con transistores en los ánodos para encender cada display, ya que no se lo puede tratar directamente desde pines del microcontrolador, por la corriente que éste debería suministrar. Un display que posee todos sus segmentos encendidos consumirá 25mAx8=200mA, y si lo conectamos directamente al microcontrolador dará fin a su vida. Entonces el transistor de propósito general PNP 2N3906 nos garantizará la corriente adecuada para los segmentos sin estropear el microcontrolador.

La aplicación que haremos es sencilla, consistirá en un contador que desbordará el timer0 a un preescaler de 256 y cada cuatro veces que ocurra tal desborde la cuenta irá ascendiendo. El contador entonces se incrementará aproximadamente cada 262.144ms:

```
256(timer0, de 8
bits)x256(preescaler)x4(variable
count)x1us(Fosc) = 262.144ms
```

El hardware que usaremos es el siguiente:



El código definitivo será:

```
#include <16f88.h>
#fuses
XT,NOWDT,PUT,MCLR,NOBROWNOUT,NOLVP,NO
FCMEN,NOIESO,NODEBUG,NOPROTECT,
#use delay(clock=4M)
#use fast_io(a)
#use fast_io(b)

#define srck PIN_A1
#define rck PIN_A2
#define ser_in PIN_A0

unsigned int const LED
[10]={0x03,0x9f,0x25,0x0d,0x99,0x49,0x41,
0x1F,0x01,0x09};
int uni=0,dec=0,mil=0,count=0;

void cargar(int datob);
void mostrar(void);

#int_timer0
void isr_timer0(void){
  count++;
  if(count>3){
    uni++;
    if(uni>9){
      uni=0;
      dec++;
      if(dec>9){
        dec=0;
        mil++;
        if(mil>9)
          mil=0;
      }
    }
    count=0;
  }
}

void main(void){
  setup_adc(adc_off);
  set_tris_a(0b111000);
  set_tris_b(0b11111000);
  output_a(0);
  output_b(0b00000111);
  setup_timer_0(rtcc_internal | rtcc_div_256);
  set_timer0(0);
  clear_interrupt(int_timer0);
  enable_interrupts(int_timer0);
  enable_interrupts(global);
  while(true){
    mostrar();
  }
}

void mostrar(void){
  cargar(LED[uni]);
  output_low(pin_b2);
  delay_ms(1);
  output_high(pin_b2);
  delay_ms(1);
  cargar(LED[dec]);
  output_low(pin_b1);
  delay_ms(1);
  output_high(pin_b1);
  delay_ms(1);
  cargar(LED[mil]);
  output_low(pin_b0);
  delay_ms(1);
  output_high(pin_b0);
  delay_ms(1);
}

void cargar(int datob){
  int contador;

  output_low(srck);
  delay_cycles(2);
  output_low(rck);
  for(contador=0;contador<8;contador++){
    output_bit(ser_in,(datob & 0x1));
    datob>>=1;
    output_high(srck);
    delay_cycles(1);
    output_low(srck);
  }
  output_high(rck);
  delay_cycles(1);
  output_low(rck);
}
```

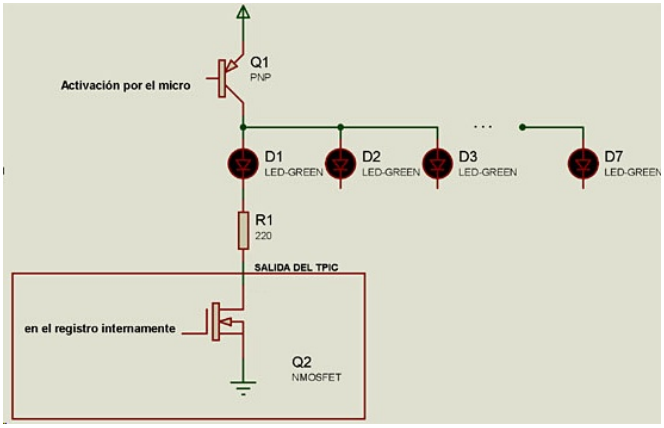
Ahora explicaremos un poco sobre el display, en particular sobre los datos del display. Como ya sabemos, los displays utilizados son de ánodo común. El ánodo se conecta a Vcc o en nuestro caso a un transistor PNP con el emisor a Vcc y el colector al ánodo (+).

Esto quiere decir que, para encender un segmento debemos enviar un cero a éste

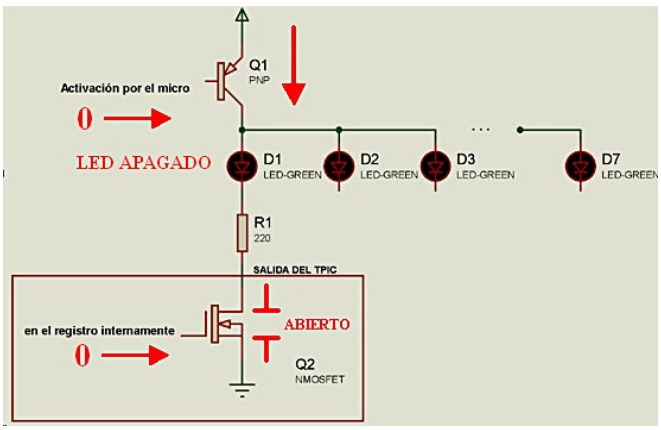
para que se establezca la polarización del led del segmento correspondiente y de esta manera veamos el led de ese segmento encendido. Pero hay que considerar que el TPIC posee salidas MOSFET a drenador abierto y son de canal N, por tanto estas salidas las estaremos colocando con el led y resistencia de 220 Ohm correspondiente, como pull-up a través del PNP con el emisor a Vcc. Y como al trabajar con mosfets de canal N en las salidas del TPIC, debemos invertir la lógica de entrada, como si trabajáramos con displays de cátodo común.

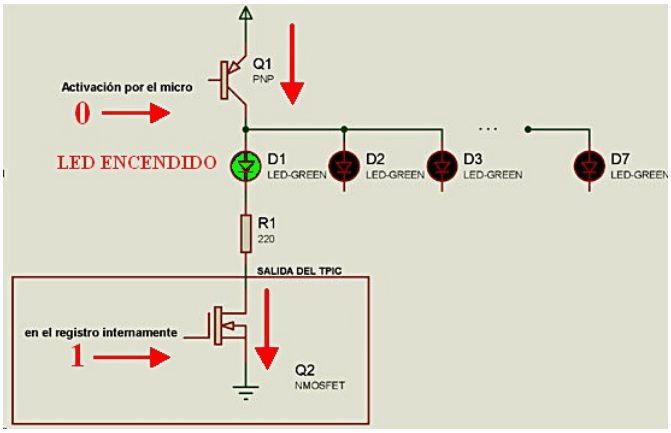
Ya que para obtener el cero que necesitamos en la salida del TPIC, en este circuito en particular, el transistor mosfet deberá recibir un 1 lógico en su gate.

Por ejemplo, para ver un 8 en el display, todos los segmentos deben estar encendidos, entonces debemos mandar a las entradas de los Mosfet 1s lógicos. Los datos que guardaremos en el programa para el micro serán como si fueran para displays de cátodo común. Esta es la esquematización de lo que tendremos:

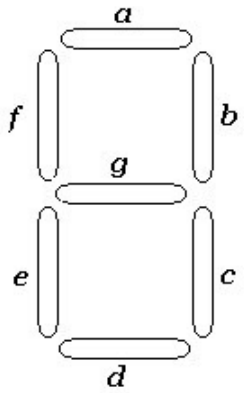


Esto es lo que sucede si en la entrada del gate del mosfet de salida tenemos un ‘0’ o un ‘1’ lógico, la entrada del gate se corresponde directamente con el dato serial ingresado.





Como bien sabemos un display de led’s de siete segmentos nombra a cada led como a, b, c, d, e, f, g y DP o h al punto en caso de existir.



Tomaremos al segmento ‘a’ como el más significativo, y nuestros datos para este caso serán:

Dato a ver	a	b	c	d	e	f	g	h	en hexa	en dec
0	1	1	1	1	1	1	0	0	0xfc	252
1	0	1	1	0	0	0	0	0	0x60	96
2	1	1	0	1	1	0	1	0	0xda	218
3	1	1	1	1	0	0	1	0	0xf2	242
4	0	1	1	0	0	1	1	0	0x66	102
5	1	0	1	1	0	1	1	0	0xb6	182
6	1	0	1	1	1	1	1	0	0xbe	190
7	1	1	1	0	0	0	0	0	0xe0	224
8	1	1	1	1	1	1	1	0	0xfe	254
9	1	1	1	1	0	1	1	0	0xf6	246

Que lo tenemos definido como constante en el código del programa para el micro:

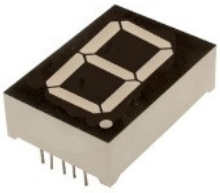
```
unsigned int const LED [10]={0xfc, 0x60,
0xda, 0xf2, 0x66, 0xb6, 0xbe, 0xe0, 0xfe,
0xf6};
```

Y con eso ya tenemos los datos para nuestros displays de ánodo común listos para ser enviados y mostrados.

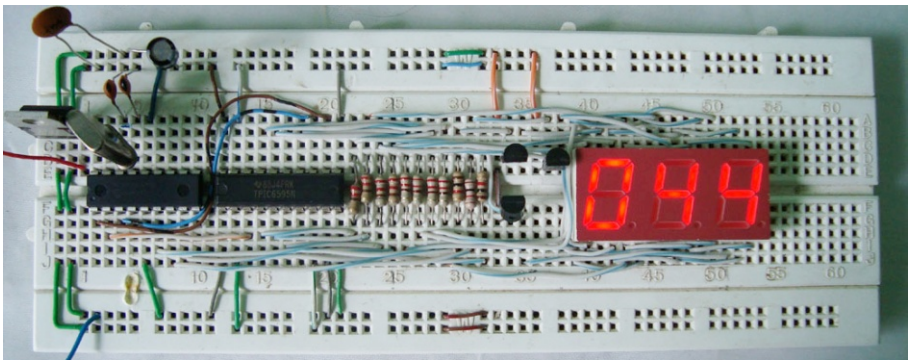
Ahora una vez explicado todo lo anterior nos falta detallar un poco el programa, comprendiendo todas las anteriores ideas, podemos resumir el funcionamiento del programa para el microcontrolador como:

1. Inicialización del micro, puertos I/O, timer0, interrupciones.
2. Cargamos valor de unidades al registro (función cargar)
3. Encendemos el display de unidades, mediante su transistor
4. Esperamos un tiempo para que sea visible al ojo humano
5. Apagamos el display de unidades, mediante su transistor
6. El transistor toma un tiempo en apagarse, entonces aguardamos
7. Repito pasos del 2 al 6 pero con decenas y su display correspondiente
8. Repito pasos del 2 al 6 pero con centenas y su display correspondiente
9. Vuelvo al paso 2

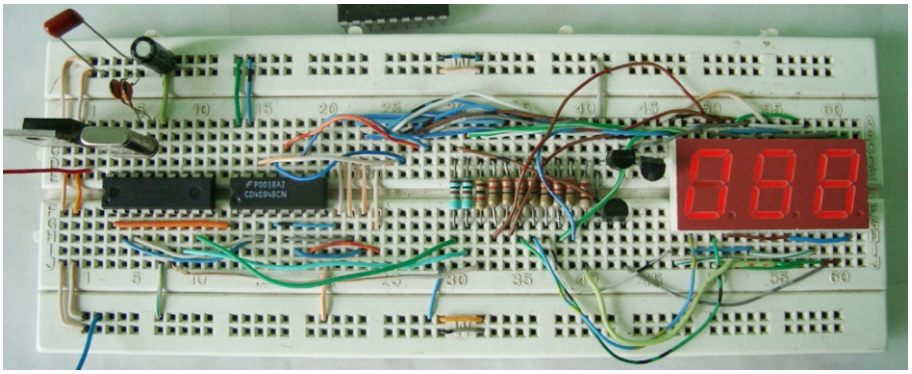
Continúo mostrándoles unas fotos del circuito armado y funcionando, y una comparación del mismo circuito armado con diferentes componentes:



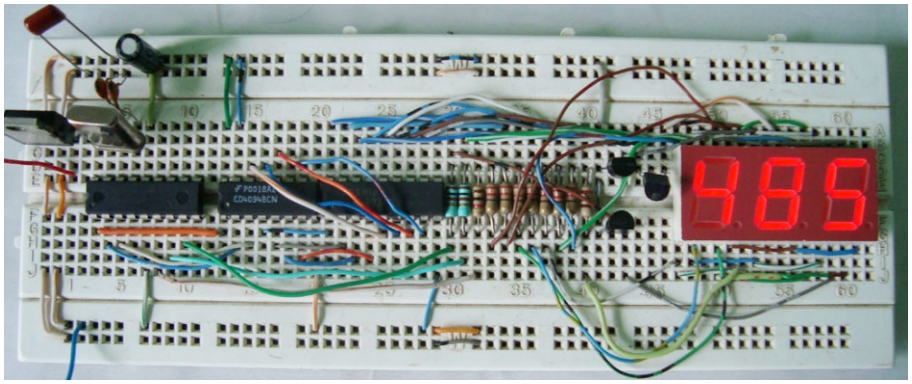
1. Primero nos vamos con el TPIC6595 de texas:



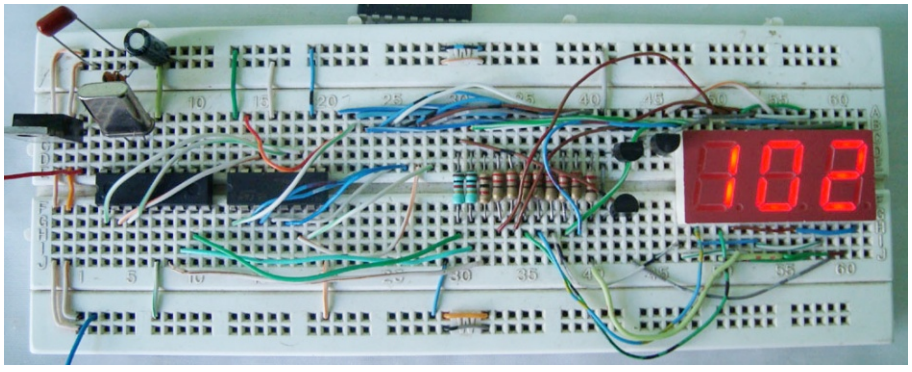
2. Ahora cambiamos el TPIC por un registro CD4094



3. Como puede verse los leds no tienen el brillo suficiente. Podemos solucionar esto agregando un ULN2803:

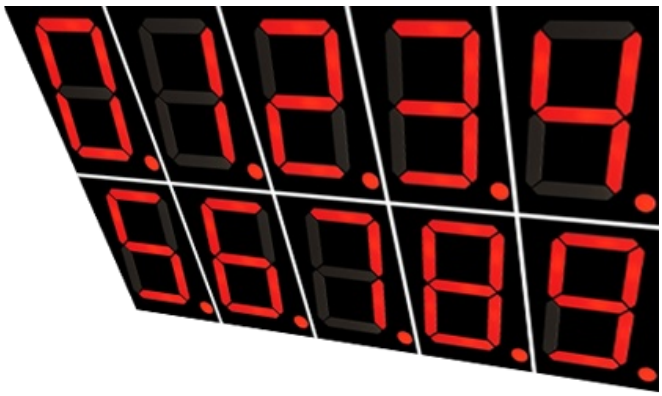


4. Los registros 74HC595 se caracterizan por tener una mejor salida de corriente en sus salidas, y eso lo demuestra la siguiente imagen:



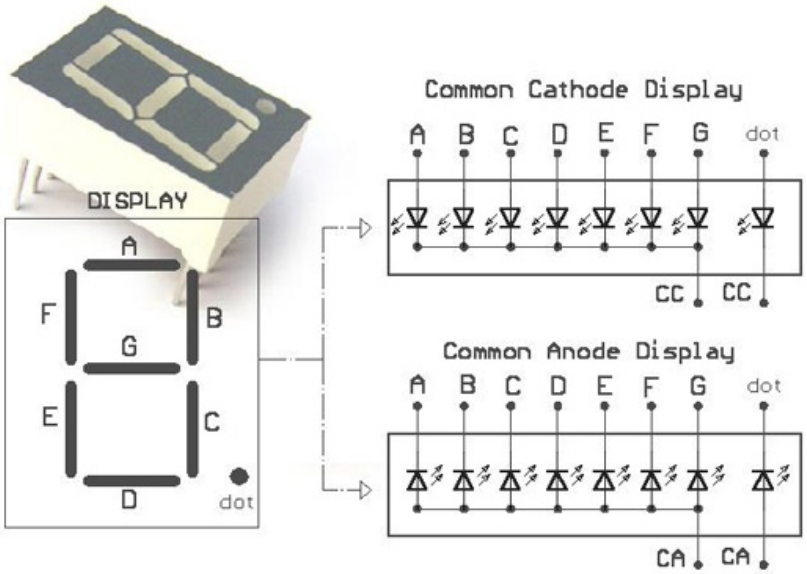
5. Si se coloca un ULN2803 junto al 74HC595, se ve idéntico a la anterior imagen. Pero probablemente con un circuito sin el ULN se exige un poco en corriente al chip y no dura mucho,

dado que la hoja de datos nos dice que no soporta tanta corriente.



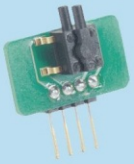
Entonces entre estos 3 registros analizados, la diferencia es la cantidad de corriente que pueden entregar o recibir, el CD4094 siempre va a necesitar de un ULN para darle más brillo a los led's, en cambio el 74HC595 puede no necesitar del ULN. El TPIC puede que sea difícil de conseguir en las tiendas de electrónica cerca de casa, hay que, o bien comprarlo de una web o directamente del fabricante Texas Instruments. En mi caso lo obtuve mediante muestras gratuitas que puede enviar el fabricante. Los TPIC si garantizan una buena corriente que pueda circular por sus pines de salida.

Debo aclarar que para los 3 registros se usa el mismo código, solo que, cuando el 4094 y el 595 están sin ULN, se debe invertir la lógica de los datos que se mostrarán en los displays.







automatismos

MAR DEL PLATA

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

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

TIPS de programación en ASSEMBLER para PIC

En el presente artículo, presentaremos una serie de “Tips” de programación en lenguaje ensamblador o assembler para los microprocesadores PIC® de Microchip con palabras de instrucción de 12 y 14 bits.

// por: Hector Javier (HJ) //
hj.0x00@gmail.com

Introducción

Son pequeños trucos que ayudan a realizar operaciones no previstas originalmente dentro del lenguaje como instrucciones, complementando de este modo las instrucciones y pseudo-instrucciones que reconoce el lenguaje ensamblador de Microchip.

Unas de las carencias a la hora de programar en assembler, es que no encontramos instrucciones para trabajar directamente con W, siendo este registro el más utilizado, como por ejemplo incrementar o decrementar W.

Algunos de estos Tips pueden ser guardados como macros con un nombre más representativo, lo que deriva en un código más entendible. El nombre de dichas macros se erigirán de tal forma que coincidan en el formato con la terminología de utilizada por Microchip. Empecemos!

TIP#1

Incrementar W: addlw 0x01

```
; Definimos el nombre de la macro para
; incrementar W
incw  macro
      addlw  0x01
      endm          ;Fin de la macro
```

En nuestro programa si necesitamos incrementar W, simplemente escribimos incw en lugar de “addlw 0x01”, lo que hace más entendible nuestro programa. Esto es aplicable al resto de los Tips.

TIP#2

Decrementar W: addlw 0xFF

```
; Definimos el nombre de la macro para
; decrementar W
decw        macro
      addlw  0xFF
      endm          ;Fin de la macro
```

TIP#3

Invertir o complementar W: xorlw 0xFF

```
; Definimos el nombre de la macro para
; invertir W
comw        macro
      xorlw  0xFF
      endm          ;Fin de la macro
```

TIP#4

Igualdad de W con un literal: Con esta rutina determinamos si el contenido de W es igual o

distinto a un literal. Realizamos una OR Exclusiva entre W y el literal, para luego comparar el bit Z del registro STATUS.

```
;Definimos el nombre de la macro
tstwkse     macro
      xorlw  k
      btfss  STATUS,Z ;Salta si son iguales
      endm           ;Fin de la macro
```

Test W=K skip Equals: Verificar si W=k, saltar si son iguales

Si no deseamos utilizar una macro, podemos escribir el código que mostramos a continuación:

```
      xorlw  k
      btfss  STATUS,Z
      goto   rutina_distintos
      goto   rutina_iguales
```

TIP#5

Verificar si el contenido de W es 0x00: Esta rutina nos permite determinar si el contenido de W es “cero”, sin necesidad de utilizar otro registro. En primer lugar le adicionamos “0x00” a W y luego verificamos si el bit Z del registro STATUS está a “1”

```
;Definimos el nombre de la macro
tstwsz      macro
      addlw  0x00
      btfss  STATUS,Z;Salta si W = "0x00"
      endm           ;Fin de la macro
```

test w skip zero: Verificar W, saltar si es cero

```
;Definimos el nombre de la macro
 tstwsnz     macro
      addlw  0x00
      btfsc  STATUS,Z ;Salta si W <> “0x00”
      endm           ;Fin de la macro
```

test w skip no zero: Verificar W, saltar si es distinto de cero

Aquellos lectores que no deseen utilizar la macro propuesta, pueden usar el siguiente código:

```
      addlw 0x00
      btfss  STATUS,Z
      goto   w_distinto_a_cero
      goto   w_igual_a_cero
```

TIP#6

Poner a “1” algunos bits de W: Para poner a “1” algunos bits determinados de W, se hace un “iorlw” con un literal que tenga a “uno” los bits a modificar y a “cero” los que deben permanecer igual.

Por ejemplo, si se desean poner a “1” el bit 0 y el bit 4 del registro W, se debe realizar escribir:

```
iorlw 0x11 ( o lo que es lo mismo:  iorlw
B’00010001’)
```

Con lo que nos aseguramos que los bits 0 y 4 de W tendrán un “1” sin afectar al resto de los bits.

TIP#7

Poner a “0” algunos bits de W: Para poner a “0” algunos bits determinados de W, se hace un “andlw” con un literal que tenga a “cero” los bits a modificar y a “uno” los que deben permanecer igual.

Por ejemplo, si se desean poner a “0” la parte alta registro W sin afectar el contenido de la parte baja, se debe realizar escribir:

```
andlw 0x0F ( que es equivalente a: andlw
B’00001111’)
```

TIP#8

Hallar el complemento a 2 de W: En este caso primero invertimos W y luego le adicionamos 1, logrando el complemento a 2 de W.

```
; Definimos el nombre para complementar a 2
W.
com2w       macro
      xorlw  0xFF
      addlw  0x01
      endm           ;Fin de la macro
```

TIP#9

Invertir un BIT: Algo que notamos al trabajar con las instrucciones de BIT, es que solamente tenemos las dos que nos permiten establecer el valor del bit y las dos que nos permiten conocer su estado. Sin duda extrañamos una que nos permita invertir un bit determinado, sin importar el estado inicial y sin modificar el resto de los bits en el proceso.

El invertir el estado de un bit es simple, es más complicado explicarlo que hacerlo, se testea si vale “1”, de ser así se salta a una línea de código que lo pone en “0”, caso contrario, se salta a una línea de código que lo pone a “1”

```
; Definimos el nombre de la macro
invb  macro bit_a_invertir
      btfss   bit_a_invertir
      goto    $+3
      bcf     bit_a_invertir
      goto    $+2
      bsf     bit_a_invertir
      endm          ;Fin de la macro
```

Para utilizar esta macro, debemos llamarla desde nuestro programa, con su nombre seguido del bit a invertir, por ejemplo si deseamos invertir el bit 3 del registro “PORTB”, debemos escribir:

```
      invb    PORTB,3
```

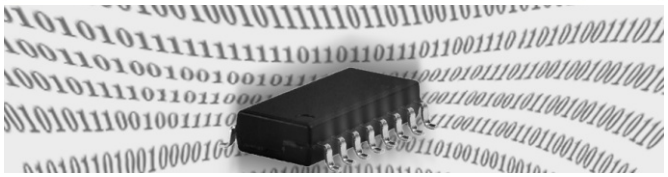
Conviene aclarar para los que no están familiarizados con las MACROS, que estas no suponen un ahorro en la memoria de programa, solamente hacen más legible el programa, ya que al ser ensamblado, el nombre de la macro es reemplazado por su contenido. Esto significa que en el ejemplo anterior “invb PORTB,3” ocupa 5 posiciones en la memoria de programa.

Si no deseamos utilizar una macro, utilizando el mismo ejemplo, debemos escribir lo siguiente:

```
      btfss   PORTB,3
      goto    $+3
      bcf     PORTB,3
      goto    $+2
      bsf     PORTB,3
```

Para finalizar, incluimos la tabla de pseudo-instrucciones para Microcontroladores PIC con palabras de instrucción de 12 y 14 bits.

Menmonic	Description		Equivalent Operation(s)		Status
ADDCF	f,d	Add Carry to File	BTFSC INCF	3,0 f,d	Z
ADDDCF	f,d	Add Digit Carry to File	BTFSC INCF	3,1 f,d	Z
B	k	Branch	GOTO	k	-
BC	k	Branch on Carry	BTFSC GOTO	3,0 k	-
BDC	k	Branch on Digit Carry	BTFSC GOTO	3,1 k	-
BNC	k	Branch on No Carry	BTFSS GOTO	3,0 k	-
BNDC	k	Branch on No Digit Carry	BTFSS GOTO	3,1 k	-
BNZ	k	Branch on No Zero	BTFSS GOTO	3,2 k	-
BZ	k	Branch on Zero	BTFSC GOTO	3,2 k	-
CLRC		Clear Carry	BCF	3,0	-
CLRDC		Clear Digit Carry	BCF	3,1	-
CLRZ		Clear Zero	BCF	3,2	-
LCALL	k	Long Call	BCF/BSF BCF/BSF CALL	0x0A,3 0x0A,4 k	
LGOTO	k	Long GOTO	BCF/BSF BCF/BSF GOTO	0x0A,3 0x0A,4 k	
MOVFW	f	Move File to W	MOVF	f,0	Z
NEGF	f,d	Negate File	COMF INCF	f,1 f,d	Z
SETC		Set Carry	BSF	3,0	-
SETDC		Set Digit Carry	BSF	3,1	-
SETZ		Set Zero	BSF	3,2	-
SKPC		Skip on Carry	BTFSS	3,0	-
SKPDC		Skip on Digit Carry	BTFSS	3,1	-
SKPNC		Skip on No Carry	BTFSC	3,0	-
SKPNDC		Skip on No Digit Carry	BTFSC	3,1	-
SKPNZ		Skip on Non Zero	BTFSC	3,2	-
SKPZ		Skip on Zero	BTFSS	3,2	-
SUBCF	f,d	Subtract Carry from File	BTFSC DECF	3,0 f,d	Z
SUBDCF	f,d	Subtract Digit Carry from File	BTFSC DECF	3,1 f,d	Z
TSTF	f	Test File	MOVF	f,1	Z



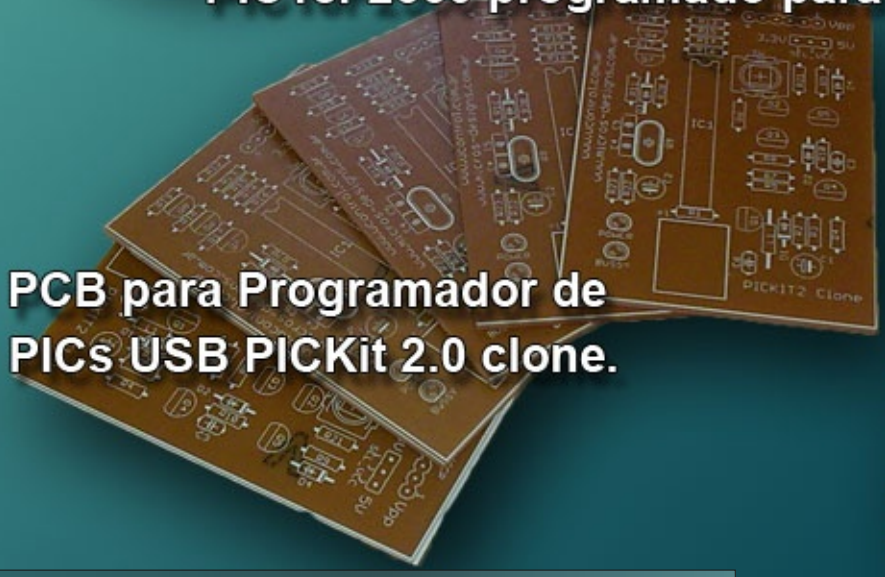
TIENDA µCONTROL

PCB para el entrenador PIC TRAINER
para micros de 18 pines.

PCB para el entrenador PIC TRAINER
para micros de 40 pines.

"El Relojito": PIC16F628A programado,
listo para que montes tu propio reloj.

"Programador de PICs USB PICKit 2.0 clone"
PIC18F2550 programado para construir éste programador.



PCB para Programador de
PICs USB PICKit 2.0 clone.

http://www.ucontrol.com.ar/forosmf/tienda-ucontrol/

Los envíos se hacen mediante Correo Argentino

Receptor de mandos infrarrojos

La intención del artículo es la de explicar cómo realizar un sistema receptor de infrarrojos, que sea capaz de descodificar un protocolo específico de cualquier mando a distancia común en cualquier hogar. También implementaremos un pequeño sistema por contraseña.

// por: Miguel Ángel Borrego Trujillo //
correo autor

Para realizar dicho receptor, utilizaremos un microcontrolador de la empresa Microchip, concretamente el PIC16F877. Podríamos haber utilizado cualquier otro microcontrolador, pero escogí este microcontrolador porque es el que tenía a disposición.

Una vez determinado el protocolo y habiendo estudiado las características del PIC, estableceremos qué elementos (pines, registros, interrupciones...) deberemos utilizar y qué requerimientos de hardware adicional nos harán falta para desarrollar las tareas requeridas.

Introducción a la solución:

Sabiendo cuál es el proyecto que debemos realizar, tenemos que fijar unos objetivos para llevar a cabo el trabajo de una manera organizada.

Los objetivos para el correcto del desarrollo del trabajo son:

1. Concretar el protocolo de comunicación.
2. Estudio del protocolo de comunicación.
3. Definir el hardware y circuitería necesaria.
4. Implementar el programa.
5. Simulación del circuito.
6. Test del circuito en protoboard.
7. Desarrollo final en una placa agujereada o pcb.

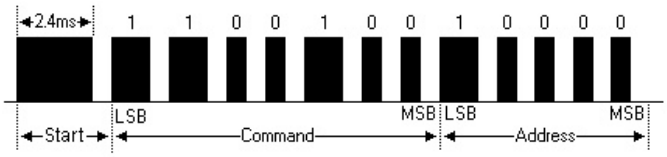
Concretar el protocolo de comunicación.

El protocolo de comunicación que usaremos para realizar a cabo la comunicación entre el emisor y el receptor, es el código SIRC, que ha desarrollado la empresa SONY y es el mismo protocolo que encontramos en los mandos de infrarrojos que esta empresa fabrica y distribuye.

Estudio del protocolo de comunicación

SIRC 12-bits:

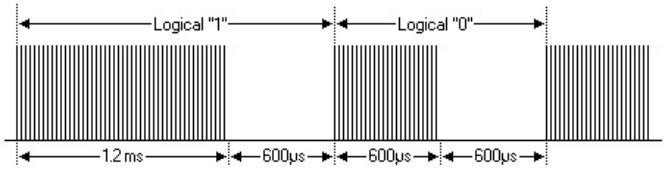
Podemos observar cómo está estructurado



el código SIRC, que es un código que distribuye los bits que envía en dos partes, el comando y la dirección. Para empezar la comunicación se envía una señal de 'START' que consiste en un pulso en alto de 2400 us y un pulso en bajo de 600 us, seguidamente se envían los datos. Para enviar un 1 se genera una señal en alto de 1200 us y uno en bajo de 600 us, para enviar un 0 se genera una señal

en alto de 600 us y uno en bajo de 600 us.

En la imagen anterior podemos observar cómo se envían los bits menos significativos (LSB) hacia los más significativos (MSB) del comando y la dirección. Utilizando la imagen de ejemplo Adress = 00001 i Command = 001001. Éste protocolo es de 12 bits de datos, 7 bits de comando y 5 bits de dirección. El comando nos indica la acción a realizar y la dirección nos indica el dispositivo que debe recibir el comando.

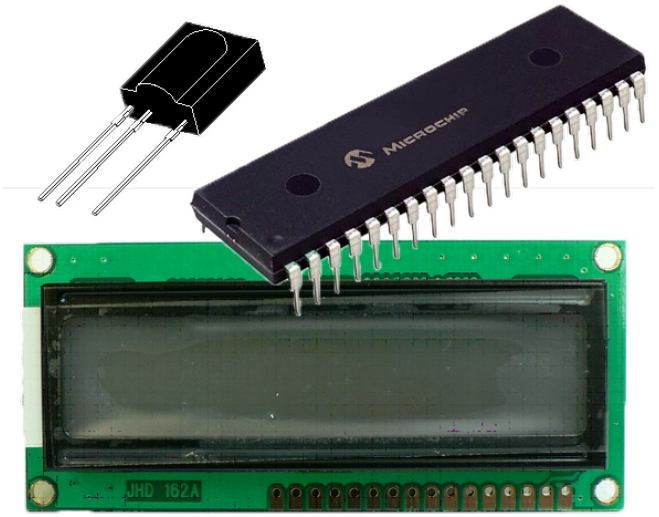


En la generación del señal, éste se modula a una frecuencia de 40 Khz según la norma del protocolo y posteriormente se demodula con el sensor de infrarojos como podemos ver en la imagen.

Definición del Hardware y circuitería necesaria.

Para poder recibir correctamente los datos se necesita un sensor de infrarrojos, que nos acondicione la señal enviada por el emisor. El sensor elegido para hacer este trabajo es el TSOP1740 aunque este podría ser sustituido por cualquier otro sensor que reciba infrarojos. Los dos últimos números del nombre del sensor corresponden a la frecuencia de demodulación en la que trabaja el sensor. Otro elemento importante es el PIC. En este caso decidí utilizar el PIC16F877 ya que es el microcontrolador que tenía. Para acabar, el último elemento importante del hardware sería el LCD ya que por éste mostraremos los datos que enviaremos a través del emisor.

Otros elementos necesarios son los siguientes: las pilas, el portapilas, los bornes de alimentación, el cristal de cuarzo, los condensadores, los botones, el potenciómetro para el display y las resistencias de PULL-UP.



El siguiente paso después de definir el hardware necesario corresponde a decidir cómo conectaremos los diferentes elementos entre ellos. De los diferentes pines de entrada / salida que tenemos, decidiremos qué pines utilizaremos, ya que el 16F877 tiene 5 puertos.

El sensor TSOP1740 tiene 3 pines, alimentación, masa y salida. Como quiero determinar el tiempo de los diferentes pulsos que nos llegan del sensor, he decidido conectar la salida del sensor a la pata RB0 del PORTB, ya que este pin nos genera una interrupción cada vez que nos llega un flanco de subida o de bajada, según el flanco configurado en el programa. Cabe destacar que el sensor TSOP1740 invierte la señal de salida, cuando recibe señal infrarroja nos pone un 0 a la salida y cuando no recibe nos pone un 1 (lo tendremos que tener en cuenta en el programa).

El display LCD que disponemos, está constituido por 16 pines que son alimentación, masa, contraste, habilitación, lectura / escritura, comando / dirección, 8 pines de datos y 2 pines del backlight. De los 8 pines de datos sólo utilizaremos los 4 pines de mayor peso y los conectaremos al PORTD, debido a que la librería que el programa CCS incorpora hace funcionar el LCD a 4 bits con el PORTD.

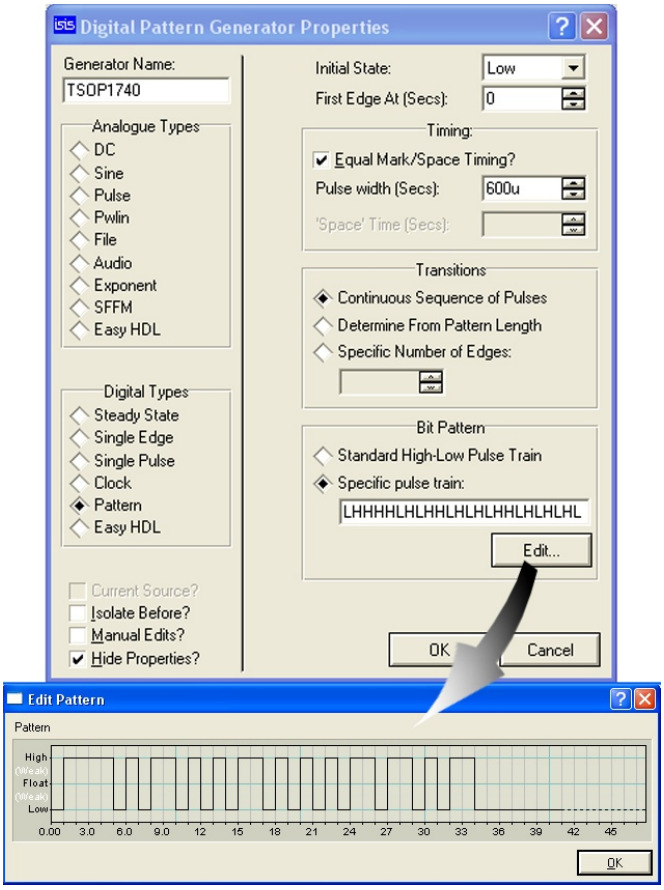
De los 5 botones, 4 los utilizaré para hacer una protección mediante contraseña

conectados a los 4 bits altos del PORTB, ya que utilizaré la interrupción por cambio de estado que estos 4 pines me generan al haber algún cambio en estos. El botón que queda lo conectaré al pin de reset (MCLR). Estos pulsadores estarán conectados a los pines correspondientes mediante resistencias de PULL-UP de 10k.

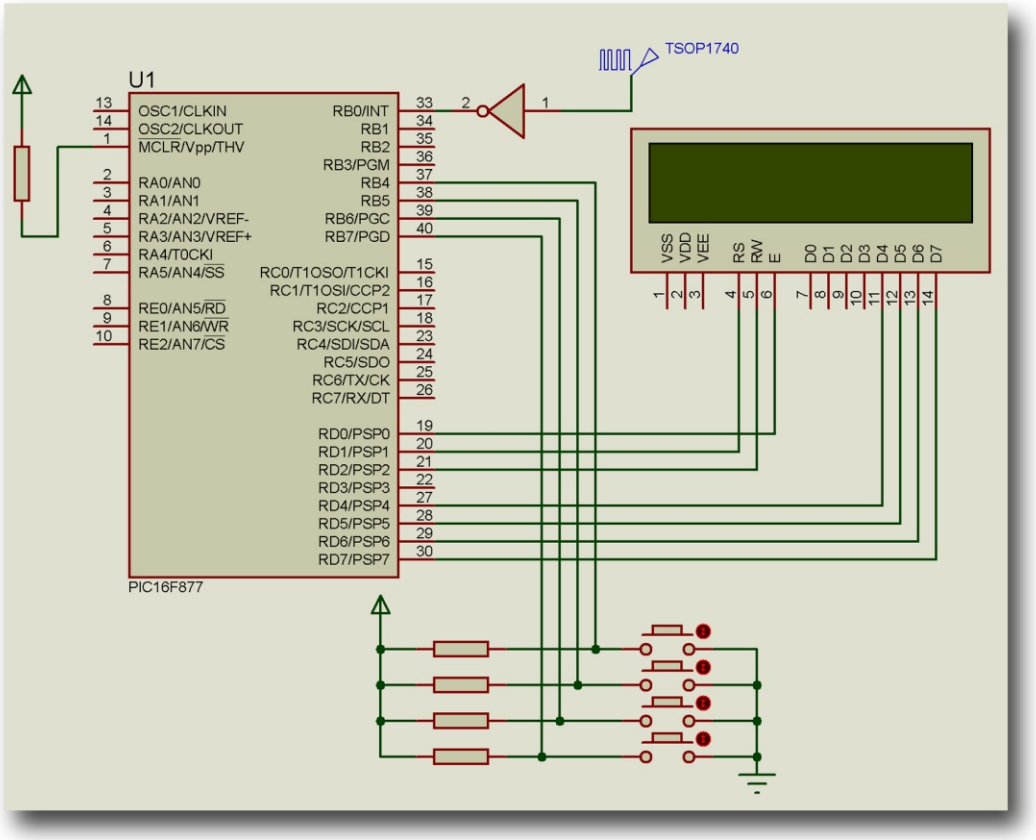
El resto de conexiones son las conexiones básicas del oscilador, los condensadores y las alimentaciones de los diferentes elementos.

Lista del hardware necesario:
1 LCD 16x2
1 PIC16F877
1 Cristal 4MHz
5 Resistencias 10K ohm
1 Condensador 100nf (Alimentación)
5 Pulsadores
1 Resistencia 330 ohm
1 Led rojo (On / Off)
1 Receptor TSOP1740
1 Potenciómetro (Contraste LCD)
1 Portapilas

Configuración para emulación de código SIRC en simulación:



Ejemplo de conexionado en Proteus:



Implementación del programa.

El programa lo deberemos de estructurar muy bien para no perdernos durante la programación y para posteriormente poder detectar los errores de programación más rápido y poder hacer los retoques oportunos. La estructura del programa está basada en diferentes funciones que iré explicando.

1) Rutina de interrupción por flanco.

Lo que hacemos en esta rutina es configurar la interrupción para que se active con un flanco de bajada (ya que el sensor nos invierte el señal), si hemos recibido un flanco de bajada ponemos el TIMER0 a 0 y establecemos una interrupción para detectar un flanco de subida. Una vez se recibe el flanco de subida guardamos el tiempo entre los dos flancos en una variable llamada tiempo. Después de esto detectamos si el valor de la variable tiempo cumple los requisitos de START dentro de un intervalo. Si cumple ponemos un flag llamado START a 1. A partir de aquí ya podemos rellenar la palabra de código de 12 bits con unos y ceros comparando los valores de la variable tiempo. Si obtenemos un valor de 600 us le corresponde un 0, si obtenemos un tiempo de 1200 us le corresponde un 1. Al TIMER0 se le ha puesto un divisor de 16 para no sobrepasar su valor máximo de 255 cuando nos llegue el START (que es el pulso que más dura).

```
#INT_EXT
void ext_isr(){

  if(cambio){                 // Hemos recibido flanco de subida
    tiempo=get_timer0();          // Miramos de cuanto tiempo es el pulso
    EXT_INT_EDGE(H_TO_L);         //Configuramos para recibir flanco de bajada
     cambio=0;
  }else {                      // Hemos recibido flanco de bajada
    set_timer0(0);               // Timer0 a 0
    tiempo=0;                   // Tiempo a 0
    EXT_INT_EDGE(L_TO_H);         //Configurramos para recibir flanco de subida
     cambio=1;
     }
  if (tiempo>140 && tiempo<160){   // Comprobamos START
    start=1;                    // flag start a 1
    i=0;                       // Reseteamos el contador de bits
  }
  if(start){                   // Si hemos detectado start...
    if(tiempo>27 && tiempo<55){     // es un 0 ?
    bit_clear(word,i);           // Añadimos el bit a la palabra
    i++;                                // incrementa contador bits
    }
    if(tiempo>65 && tiempo<90){   // es un 1 ?
    bit_set(word,i);             // Añadimos el bit a la palabra
    i++;                        // incrementa contador
    }
    if(i==longitud){             // Si contador = ntotalbits
    dato_recibido=1;              // flag dato redibido activado
    EXT_INT_EDGE(H_TO_L);         // Flanco de subida para el START
    start=0;                    // Inicialización para la siguiente palabra
    }
  }
```

2) Rutinas de descodificación de comando y dirección.

Una vez terminada la transmisión de datos se activa un flag de dato_recibido y seguidamente hacemos las modificaciones oportunas para separar el comando y la dirección. Para hacer esto lo que hacemos es coger la palabra de 12 bits y separar las 2 partes, la dirección que es de 5 bits y comando que es de 7 bits teniendo en consideración que en el protocolo se envían los datos del bit menos significativo al bit más significativo (al revés).

```
void take_adress(void){           // Separamos la dirección de la palabra
    i=longitud-1;
    adress=0;
    for(j=0;j<longitud-7;j++){    // Hacemos un recorrido al revés
      adress=adress<<1;           //al revés para separar la direccion
      adress=adress+bit_test(word,i);
      i--;
    }
}

void take_command(void){          //Separamos el comando de la palabra
    i=6;
    command=0;
    for(j=0;j<7;j++){             // Hacemos un recorrido para separar el comando
      command=command<<1;
      command=command+bit_test(word,i);
      i--;
    }
}
```

3) Rutina del programa principal

En esta rutina lo que hacemos es esperar a que la recepción de datos haya finalizado, deshabilitamos la interrupción externa para poder extraer la dirección y el comando. Seguidamente escribimos en el LCD qué comando hemos enviado y volvemos a activar la interrupción externa de RB0.

```
if(start==1){                    // Si hem rebut START, esperem dades
      while(!dato_recibido);              // Esperamos mientras recibimos datos
      dato_recibido=0;                 // Ponemos datos recibidos a 0
      DISABLE_INTERRUPTS(INT_EXT);     // Deshabilitamos INT EXTERNA
                                 // Tratamos los datos
      take_adress();                 // Cogemos la Dirección
      take_command();                  // Cogemos el Comando
                                             // En este caso no se utiliza adress
      switch (command){                      // Mostramos por el LCD la tecla
                                     // del mando pulsada
        case 0: printf(lcd_putc,"\fCanal 1");break;
        case 1: printf(lcd_putc,"\fCanal 2");break;
        case 2: printf(lcd_putc,"\fCanal 3");break;
        case 3: printf(lcd_putc,"\fCanal 4");break;
        case 4: printf(lcd_putc,"\fCanal 5");break;
        case 5: printf(lcd_putc,"\fCanal 6");break;
        case 6: printf(lcd_putc,"\fCanal 7");break;
```

```
        case 7: printf(lcd_putc,"\fCanal 8");break;
        case 8: printf(lcd_putc,"\fCanal 9");break;
        case 9: printf(lcd_putc,"\fCanal 0");break;
        case 16: printf(lcd_putc,"\fCanal +");break;
        case 17: printf(lcd_putc,"\fCanal -");break;
        case 18: printf(lcd_putc,"\fVolumen +");break;
        case 19: printf(lcd_putc,"\fVolumen -");break;
        case 20: printf(lcd_putc,"\fSilencio");break;
        case 47: printf(lcd_putc,"\fStandby");break;
        default: printf(lcd_putc,"\fCom:%3U Adr:%2U",command,adress);
      }
      ENABLE_INTERRUPTS(INT_EXT);      // Habilitamos la interrupción externa
    }
```

4) Rutina de interrupción por cambio de estado

Lo que hago en esta rutina es leer el estado del teclado al inicio del programa para implementar una pequeña contraseña. Cuando llega la interrupción quiere decir que hubo un cambio de estado en los 4 pines altos del PORTB, primero hago una máscara para descartar el nibble bajo del PORTB, una vez he hecho esto comparo si los bits del nibble alto del PORTB están a 1 (ninguna tecla pulsada) o no lo están (alguna tecla pulsada), si no están todos a 1 procedo a hacer la lectura del PORTB y retorno el valor de las teclas que he apretado.

```
#INT_RB

void rb_interrupt(){
  disable_interrupts(global);       // Deshabilitamos las interrupciones
  delay_ms(100);                    // Esperamos 100 ms para no tener ruido
  lectura_portb=PORTB;                      //
  lectura_portb&=0xF0;              // Nos quedamos con los bits altos
  if(!(lectura_portb==0xF0)){       // Comprobamos si se ha pulsado una tecla


  tecla1=bit_test(lectura_portb,7); // guardamos estado tecla1
  tecla2=bit_test(lectura_portb,6); // guardamos estado tecla2
  tecla3=bit_test(lectura_portb,5); // guardamos estado tecla3
  tecla4=bit_test(lectura_portb,4); // guardamos estado tecla4
  tecla_pulsada=1;                                  //flag de telca pulsada
  }
  enable_interrupts(global);        // Volvemos a habilitar interrupciones
}
```

5) Trozo de código de detección de contraseña

Esta rutina sirve para detectar si la contraseña introducida es válida o no. Lo que hacemos para saber si la contraseña que hemos enviado es correcta, es habilitar las interrupciones del PORTB por cambio de estado y así poder obtener las teclas pulsadas. Una vez hemos obtenido las teclas pulsadas, determinamos qué tecla ha sido pulsada y seguidamente hacemos la acción correspondiente, incrementar, decrementar, enviar y resetear la contraseña respectivamente Password = 3.

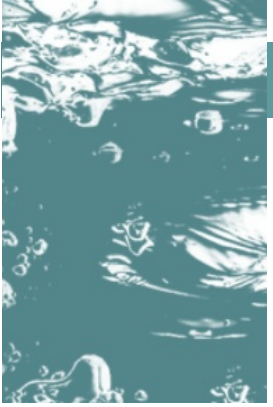
```
while (!password_ok){      // Comprobamos password

   ENABLE_INTERRUPTS(INT_RB);        // Habilitamos interrupcion PORTB
   if(tecla_pulsada){                // Si se ha pulsado miramos que tecla es
     if(!tecla1) {                   // Si es la tecla1
     password++;                     // Aumentamos valor de password
     }
     else if(!tecla2){               // Si es tecla2
     password--;                     // Disminuimos valor de password
     }
     else if(!tecla3){         // Si es tecla3
       if(password==0x03){           // Comprobamos contrasenya
         password_ok = 1;        // Password Okey !
       }
     }
     else if(!tecla4) {        // Si es tecla4
     password=0;                     // Reseteamos el password
     }
     tecla_pulsada=0;
     printf(lcd_putc,"\fIntroduce el \nPassword: %U",password); //Mostramos Lcd
   }
   if (password_ok)printf(lcd_putc,"\fPassword Correcto\nRecibiendo
                          Datos");//Password correcto
   }
```

APLICACIONES

La aplicación más importante que podemos implementar en el receptor de infrarrojos es la del control de dispositivos a distancia. Una vez hemos podido descifrar el código que envía el emisor, podemos controlar dispositivos como bombillas, motores, relés, ect.


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



Control P.I.D - Nivel básico

En esta primer parte del artículo, se pretende dar a conocer los conceptos básicos acerca de los controladores, y hacer un énfasis sobre los controladores PID. No se enfocará puntualmente la parte de aplicación, pero si se realizará un ejemplo simple para que las personas que recién comenzamos con esto del control, tengamos algo de teoría respecto al control y no solo a la aplicación.

// por: Miguel A. Lopez O. //
miguel_626@hotmail.com

Introducción

Al hombre lo mueve, casi siempre, el deseo de conocer mas a fondo las cosas para poder predecir el comportamiento de ellas (determinismo - física clásica); es inevitable pasar a querer manipular, de alguna manera, este conocimiento aplicado para obtener un resultado óptimo, o acorde a lo que queremos.

El control es algo innato en el ser humano y lo aplicamos en un sin-número de actividades cotidianas, por ejemplo: a la hora de bañarnos, y seleccionar la temperatura adecuada a la que sale el agua de la ducha, a la hora de cruzar una calle con tráfico (para aquellos temerarios) y definir la velocidad con la que cruzamos (o corremos), etc.; nos convertimos en un instrumento de control ya que juzgamos la marcha del proceso según las variables que rodean al mismo y decidimos la mejor opción para, de esta manera, obtener el mejor resultado para nosotros, ya sea, una temperatura agradable o pasar con éxito la calle.

El conocer y controlar un proceso, implica un largo recorrido, si se hace por ensayo y error para obtener el resultado que se quiere; se adquiere experiencia, la cual nutre al instrumento de control y hace que éste obtenga un resultado cada vez mejor, pero se convierte en un proceso intuitivo y no aplicable a unas nuevas condiciones. Al crecer la demanda de productos, cualesquiera que sean, se corre el riesgo de afectar la calidad por aumentar la productividad; para mantener la calidad constante (buena calidad) se hace necesario desarrollar teorías para explicar el funcionamiento del proceso, para que así, se puedan vislumbrar las variables de interés y de esta manera, se pueda controlar el proceso para nuestro beneficio.

Características del proceso

Proceso es un sistema que ha sido desarrollado para llevar a cabo una tarea determinada. Los procesos siempre estarán sujetos a variables, que, como cuyo nombre indica, cambian su valor en el tiempo. Es fundamental entonces, el medir estas variables para cuantificar el valor de esa variable y de esa manera saber en cuanto afecta a nuestro proceso en cuestión.

Un bucle de control típico, está conformado por el proceso, el transmisor, el controlador y el elemento de control.

El controlador, permite al proceso cumplir su objetivo y realiza dos funciones esenciales:

Compara la variable medida con una variable de referencia (o valor deseado - Set Point) para determinar el error (lazo cerrado).
Estabiliza el funcionamiento dinámico del bucle reduciendo el error

Como ejemplo, supongamos a un operario controlando la salida de agua caliente. El operario sensa la salida del agua con la mano y acciona una válvula de vapor para mantener el agua caliente. Ante un cambio en el flujo de agua de entrada, como la apertura de la válvula es la misma, el operario notará que el agua que sale esta mas fría que antes, por lo que, luego de la comparación mental entre la temperatura sensada y la que quiere, decide abrir mas la válvula de vapor para lograr nuevamente la temperatura que desea o que se necesita. El cambio de la temperatura no es inmediato, si no que toma un tiempo.

El conjunto de elementos en circuito cerrado que hacen posible este control reciben el nombre de bucle de control.

Los procesos presentan dos características importantes que deben considerarse:

1. Cambios en la variable controlada, a lo que se conoce como cambios de carga.
2. Tiempo necesario para que la variable del proceso alcance un nuevo valor. Este tiempo es función de propiedades como: capacitancia, resistencia y tiempo de transporte.

Las operaciones básicas que se realizan en el sistema de control son, entonces:

Medición: Se hace generalmente mediante la combinación de sensor y transmisor.
Decisión: El controlador decide que hacer con base a la medición.
Acción: Resultado de la decisión del controlador, generalmente realizada por el elemento final de control.

Es evidente que si quisiéramos quitar al operario humano y colocar en su lugar un controlador automático, éste último, debería poseer toda la información correspondiente para que tome la decisión correcta en cuanto al número de vueltas que debe dar a la válvula de vapor para lograr el valor seleccionado, es decir, debe conocer la dinámica del sistema, como se comporta éste respecto al tiempo ante cambios en la variable de control.

Sistemas Térmicos

Para comenzar, vamos a tomar como ejemplo un sistema térmico. Tal como se dijo en la introducción al tema, nos interesa el comportamiento de la variable, en este caso la temperatura, con respecto al tiempo. Como base, tenemos que la transferencia de calor se realiza, generalmente, por tres métodos:

Conducción: El intercambio de energía se realiza por el movimiento cinético o el impacto directo de las moléculas.
Convección: El intercambio de energía se relaciona con el movimiento relativo de un fluido respecto a otro.
Radiación: El intercambio de energía se realiza por emisión

Así, que en un proceso térmico, estarán presentes, en mayor o menor magnitud, estas tres formas de transferencia de calor.

Como ejemplo práctico, supongamos que tenemos inmerso en un fluido un elemento (sensorial), el cual posee ciertas características como lo son: Un área superficial A, un calor específico c y una masa m. Al estar inmerso en el fluido, el sistema está en equilibrio térmico, es decir, ha alcanzado una temperatura uniforme y no hay transferencia de calor en ningún sentido. En este caso:

$$t = 0' \quad T = T_F$$

Supongamos ahora que la temperatura del fluido TF aumenta (función paso):

$$t = 0 \quad T_F > T$$

Esta variación, hace que el equilibrio anterior se rompa y por ende nos hace pensar en que pasa dentro del sistema. Para simplificar las cosas, pensamos en lo mas básico que tiene un sistema térmico: el calor. De ahí podemos deducir que una ecuación que puede describir el equilibrio en el sistema es:

Calor de Entrada - Calor de Salida = Grado de cambio calórico en el sensor

Suponemos que el calor de salida del sistema es nulo, ya que es el sensor el que esta inmerso en el fluido, así que nos interesa conocer el intercambio calórico entre el sensor y el fluido.

El calor de entrada, está definido como:

$$Q = h.A(T_F - T)$$

En donde h es el coeficiente de transferencia de calor entre el fluido y el sensor.

El aumento calórico del sensor está determinado por:

$$m.c.[T - T(0')]$$

Nos interesa es el cambio en el tiempo del aumento calórico del sensor, es decir la tasa de aumento del contenido calórico del sensor, ésto estará dado por:

$$m.c.\frac{d[T - T(0')]}{dt}$$

Definiendo:

$$\Delta T = T - T(0') \quad y \quad \Delta T_F = T_F\_T_F(0')$$

Tenemos, reemplazando en la ecuación de equilibrio:

$$h.A.(\Delta T_F - \Delta T) = m.c.\frac{d\Delta T}{dt}$$

O, lo que es mas ordenado:

$$\frac{m.c}{h.A}\frac{d\Delta T}{dt} + \Delta T = \Delta T_F$$

Que es una ecuación diferencial de primer orden, de esas que son muy fáciles de resolver.

Recordemos que:

$$\Delta T$$

es la variable que nos indica el cambio de temperatura del nuestro sensor inmerso en un fluido. Si resolvemos la ecuación diferencial, obtendremos una función, a la cual se le da el nombre de función de transferencia y es con esta función con la que se puede predecir el comportamiento de nuestra variable en función del tiempo y en relación a un cambio de temperatura, en este caso:

$$\Delta T_F$$

Para resolver la ecuación diferencial, usamos la Transformada de Laplace, con la cual, se vuelve tan simple como lo es multiplicar y dividir. Tenemos entonces:

$$\gamma.[s\Delta T(s) - \Delta T(0')] + \Delta T(s) = \Delta T_F(s)$$

$$donde\ \gamma = \frac{m.c}{h.A}$$

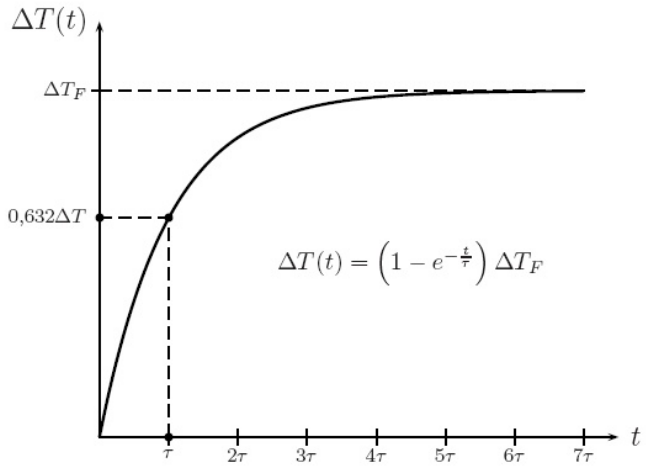
Recordemos, que en t=0', teniamos que la temperatura del elemento sensorial es igual a la temperatura del fluido, por lo tanto:

$$\Delta T(0') = 0$$

Reemplazando, hallamos la función de transferencia, la cual relaciona la salida con la entrada de un sistema, en este caso nuestro elemento sensorial.

$$\frac{\Delta T(s)}{\Delta T_F(s)} = G(s) = \frac{1}{\gamma.s + 1}$$

Si graficamos la función de solución de nuestra ecuación:



Vemos que el número:

$$\gamma$$

es muy importante, ya que justo cuando el tiempo lo iguala, el valor de la función es el 63.2% del valor final. También nos da información acerca de lo confiable que es la medida respecto al tiempo, ya que, idealmente el sistema se estabilizará en un tiempo infinito, pero, como no podemos pasarnos toda una vida midiendo, basta con para estar en un 90% del valor de estado estable.

Lo ideal, sería que todos los sistemas térmicos se comportaran de igual manera, pero siempre hay otras variables que afecten al sistema. Sin embargo éste modelo es aplicable a sistemas como los hornos, ya que describe muy bien el comportamiento dinámico de estos.

Tiempo Muerto

En los sistemas reales, se tiene que los cambios dentro del mismo sistema, no son inmediatos, si no que toman cierto tiempo, debido a la inercia del mismo proceso. En el momento de efectuarse el aumento de temperatura:

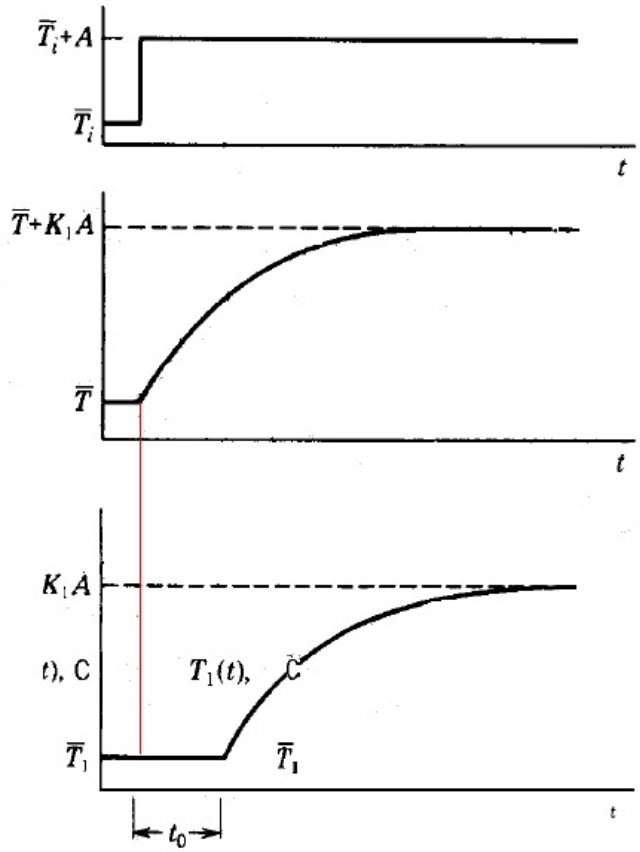
$$T_F > T$$

hay un tiempo de espera hasta que el sistema comienza a reaccionar, es como una clase de juego en la reacción.

El tiempo muerto es parte integral del proceso y se debe tomar en cuenta en la función de transferencia de los procesos.

Matemáticamente modifica nuestra función de transferencia:

$$\frac{\Delta T(s)}{\Delta T_F(s)} = \frac{K_1 e^{-t_0 . s}}{\gamma . s + 1}$$



Componentes de los sistemas de control

Como se mencionó en la Introducción, las operaciones básicas de un sistema de control son:

Medición
Decisión
Acción

Con los sensores y transmisores, se realizan las operaciones de medición en el

sistema de control. En la actualidad hay muchos tipos de sensores que nos permiten tomar medidas muy confiables en precisión y exactitud, con muy buena resolución y algunos hasta ya vienen calibrados desde fábrica. La etapa de medición es muy importante, ya que la acción del controlador se calcula en función del error entre el valor de consigna (Set Point) y el valor medido en el proceso.

La etapa de acción la realizan los actuadores, que pueden ser válvulas, interruptores, TRIAC´s, etc. La señal de salida del controlador debe estar relacionada de alguna manera con el actuador en la variable de control. Por ejemplo: para un horno, la temperatura que genera la resistencia eléctrica, está relacionada con la cantidad de corriente que fluye por ella, gracias a esta relación se puede cuantificar que cantidad de calor se introduce al sistema en función de la cantidad de corriente que se permite circular por la resistencia.

En la etapa de la decisión, está involucrado el concepto propio del control, puede ser por el simple paso del valor de consigna o por un valor de razón proporcional, etc.

Tipos de controladores por retroalimentación

Como se ha venido mencionando, los controladores por retroalimentación, toman la decisión para mantener el punto de control o Set Point, según el cálculo de la salida con base en la diferencia que hay entre la variable de salida que se está controlando y el punto de consigna, diferencia a la que se denomina error.

Control Proporcional: Es el tipo mas simple de controlador y la decisión de control hace gala a su nombre, es decir: produce una acción de control proporcional al error. A mayor valor de error, mayor sera la señal de corrección, si el valor del error disminuye, la señal de corrección disminuirá también:

$$OUT_{control}(t) = K_p . error(t)$$

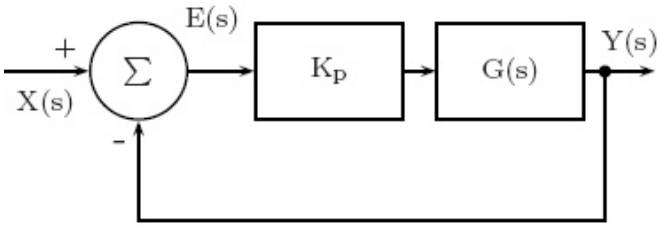
Si aplicamos Laplace -para simplificar las cosas-:

$$OUT_{control}(s) = K_p . E(s)$$

La función de transferencia del controlador proporcional corresponde a una constante:

$$K_p$$

Si aplicáramos esta señal a nuestra planta, nos quedaría un diagrama de bloques como el siguiente:



En donde:
X(s) : Set Point
Y(s) : Valor de Salida, con el cual calculo el error en función de X(s)
E(s) : Valor del error, resultado de la diferencia entre X(s) y Y(s)
G(s) : Función de la planta o del proceso

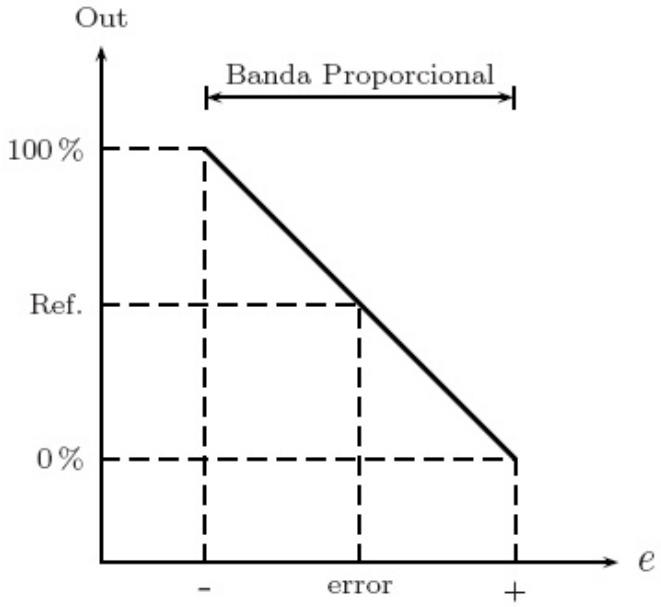
En este tipo de control, siempre va a haber error en estado estable, es decir, cuando el tiempo tiende a infinito y se supone que el proceso se estabiliza para el valor de Set Point fijado por nosotros. Esto es debido a la naturaleza del controlador, ya que es prácticamente imposible que el elemento de acción regrese al estado inicial, que fue justamente cuando se inició el error.

Matemáticamente, el error en estado estable se calcula con :

$$Error_{ss} = \lim_{s \to 0} s . E(s)$$

Este valor interesa mucho, ya que se supone que debería tender a cero (de hecho, eso es lo que se busca al implementar un controlador) cuando ha pasado el tiempo suficiente.

En casos reales, la linealidad entre el error y la salida del controlador, solo existe en un rango determinado, al cual se le denomina banda proporcional.



Para los valores por fuera de ese rango, el valor de salida se saturará y no habrá señal de control mayor, por ejemplo: en el caso de la apertura de una válvula, si el valor ya está en el máximo, no hay forma de seguir abriendo mas la válvula, o si un actuador necesita mas voltaje del que suministra la fuente.

Control Derivativo: En este tipo de control, el cambio de la salida del controlador respecto al Set Point es proporcional a la rapidez de cambio de la señal de error en el tiempo.

$$I_{sal} - I_0 = K_d . \frac{d\ error(t)}{dt}$$

En donde:

Io : Valor de salida del controlador para el valor de referencia, es decir, cuando el error es cero.
Isal : Valor de salida cuando el error cambia a una tasa d error(t)/dt .

Siguiendo el mismo método anterior, aplicamos transformada de Laplace para hallar la función de transferencia de un controlador derivativo:

$$I_{sal} - I_0 = K_d . s . E(s)$$

La función de transferencia del controlador es derivativo, entonces: s.Kd.

En esta parte, se puede apreciar la gran ventaja de trabajar con la transformada de Laplace, ya que la derivada de la función error en el tiempo, pasa a ser un producto aplicando la transformada.

Resulta lógico que este controlador no tiene efecto si el error permanece constante, por lo que no es muy utilizado solo, si no en combinación con el proporcional y el integral.

Control Integral: En este tipo de control, la tasa de cambio en la señal de salida del controlador respecto al tiempo es proporcional al error:

$$\frac{d\ I}{dt} = K_i . error(t)$$

Para hallar nuestra función de respuesta del controlador, integramos:

$$I_{sal} - I_0 = \int_0^t K_i . error(t) dt$$

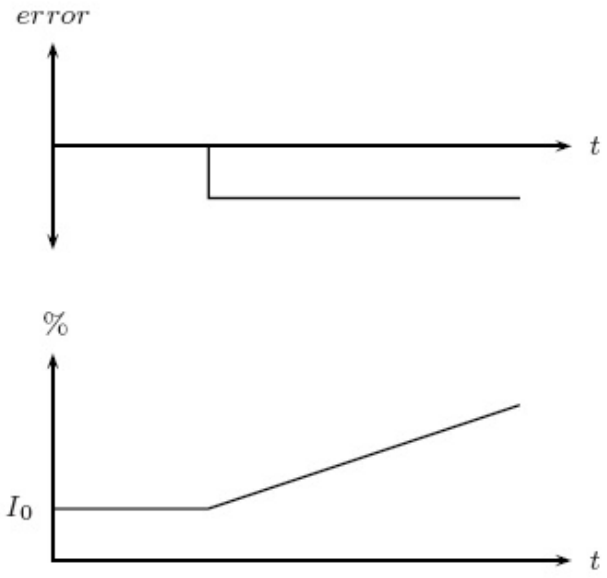
a la cual, le aplicamos Laplace -para variar-:

$$I_{sal} - I_0 = \frac{1}{s} K_i . E(s)$$

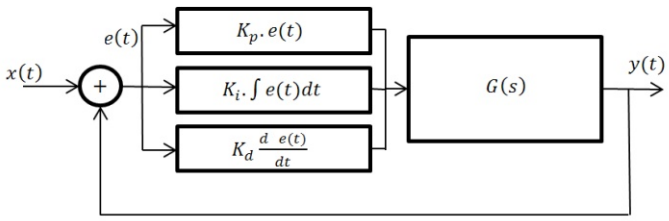
La función de transferencia del control Integral es, entonces 1/s .Ki:

Este tipo de control es muy útil para atacar el error de estado estable u OffSet, ya que para valores constantes de error, la acción del controlador aumentará en el tiempo.

Este tipo de controlador no se utiliza solo, si no en combinación con los otros anteriores.



Controlador P+I+D : Es la combinación de la acción de los tres controladores anteriores en una sola función de transferencia. De esta forma obtenemos un controlador que no tiene desviación en el error y que disminuye la probabilidad a producir oscilaciones.



Tenemos entonces:

$$I_{sal}(s) - I_0(s) = \left[K_p + \frac{1}{s}K_i + s.K_d\right].E(s)$$

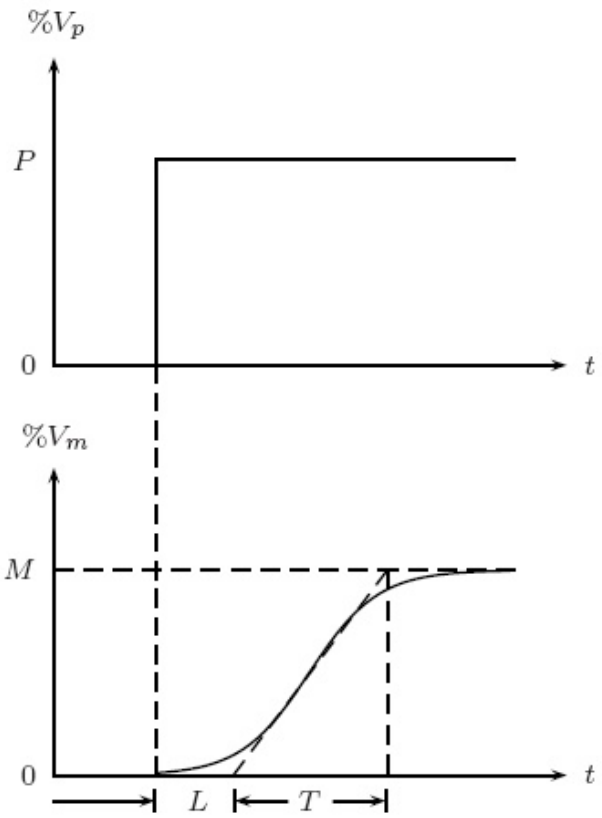
Como bien se aprecia en las ecuaciones anteriores, hay ciertas constantes de proporción en las funciones de transferencia para los diferentes tipos de controladores; si hallamos los valores adecuados para esas constantes, garantizaremos que la señal de acción del controlador sea la adecuada, que se mantenga dentro de los límites y por ende, el buen funcionamiento del proceso.

Ajuste de controladores por retroalimentación

El ajuste del controlador es el procedimiento mediante el cual se adecuan los parámetros del controlador para obtener una respuesta específica del circuito cerrado. En el caso de un controlador Proporcional-Integral (PI) habría que ajustar los dos modos de control para obtener una respuesta óptima en nuestra respuesta. En el caso de un controlador PID habría que ajustar los tres modos, el Proporcional (la ganancia), el Integral (tiempo de reajuste) y el Derivativo (tiempo de derivación), es algo así como hallar el valor RGB óptimo para un color específico.

Lógicamente, el sistema tendrá un tiempo de respuesta ante los efectos del controlador, no es el caso del color RGB, que se tendrá una respuesta inmediata. Aún con esa demora en la reacción (que puede ser de horas según el sistema) el ajuste por realimentación es muy utilizado.

Para hallar estos ajustes vamos a recurrir a un método llamado Método de la curva de reacción del proceso.



Para aplicar este método, lo que se hace es obtener la curva de reacción de nuestro sistema a una entrada escalón y verificamos que el sistema sea de orden 1. Como bien vimos anteriormente, el sistema térmico, va a presentar una respuesta típica de un sistema de orden 1 -no todos los sistemas presentan este tipo de respuesta- o al menos se va a acercar bastante.

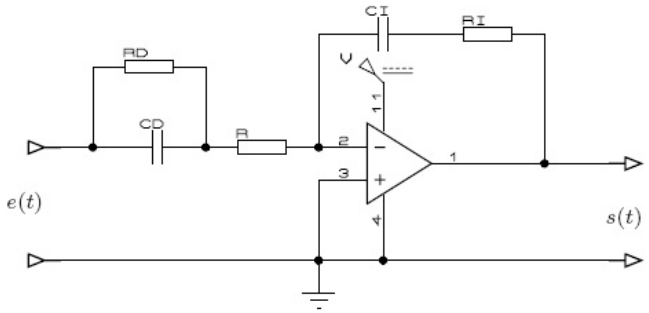
Hay unas formulas que se aplican para calcular los valores de las constantes:

Modo de control	K_p	T_i	T_d
P	P/RL		
PI	$0{,}9P/RL$	$3.33L$	
PID	$1{,}2P/RL$	$2L$	$0{,}5L$

Donde R = M / T

Estas fórmulas son empíricas y se las debemos a Ziegler y Nichols. Hay restricciones para el uso de estas ecuaciones en relación a la razón existente entre el tiempo muerto y la constante de tiempo del sistema. Si el valor de la razón entre estos dos valores está fuera de un rango de 0.10 y 1.0, no se debe usar éste método y es mejor pensar en otro

Para construir un controlador basta con un amplificador operacional configurado de la siguiente manera:



En donde tendríamos:

$$K_p = \frac{RI}{R + RD}$$

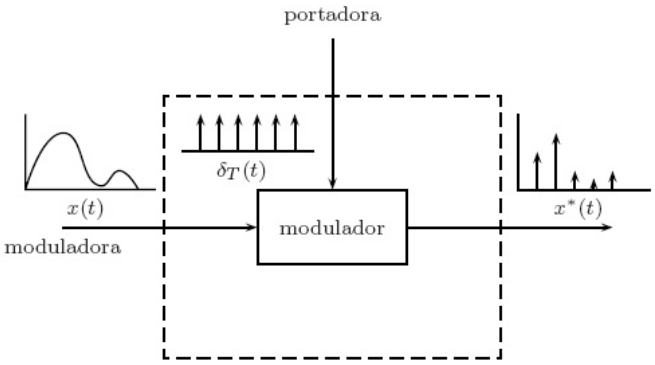
$$K_d = RD \,.\, CD$$

$$K_i = \frac{1}{RI \,.\, CI}$$

Si queremos digitalizar nuestro controlador, hay que tener en cuenta que ya no se estaría trabajando en tiempo continuo, si no que hay de por medio un tiempo entre la toma de una muestra y otra de nuestra variable, a este tiempo se le denomina Tiempo de Muestreo e influye en el comportamiento de nuestro controlador.

Su influencia radica en el concepto de frecuencia, ya que, si el tiempo con el que muestreo la señal es muy prolongado, la señal que obtendré no será la real, si no que estará modificada su frecuencia. Para saber a que tiempo se debe muestrear se usa un teorema denominado Teorema de Nyquist, que en sí nos determina cuanto debe ser la frecuencia mínima de muestreo en función de la frecuencia de nuestra señal a muestrear.

Matemáticamente se debe recurrir a una transformada para que nos facilite mas las cosas a la hora de diseñar. Esa transformada es la transformada Z. Esta transformada Z es en si, una transformada de Laplace de la señal muestreada mediante una función impulso.



Ya en la realidad, se tiene es una función escalonada, ya que el valor del último muestreo se mantiene hasta el siguiente.

Para trabajar en tiempo discreto, se debe entonces, discretizar la ecuación del controlador. Una forma sencilla y rápida es hacer una aproximación de las partes derivativa e integral. Para la parte derivativa, se aproxima a la diferencia de dos puntos y para la parte integral, se aproxima a la suma trapezoidal. Estas aproximaciones hacen que se sume un error a nuestro controlador.

Tenemos entonces:

$$m(kT) = K\left\{e(kT) + \frac{T}{T_i}\sum_{h=1}^{k}\frac{e((h-1)T + e(ht))}{2} + \frac{T_d}{T}[e(kT) - e((k-1)T)]\right\}$$

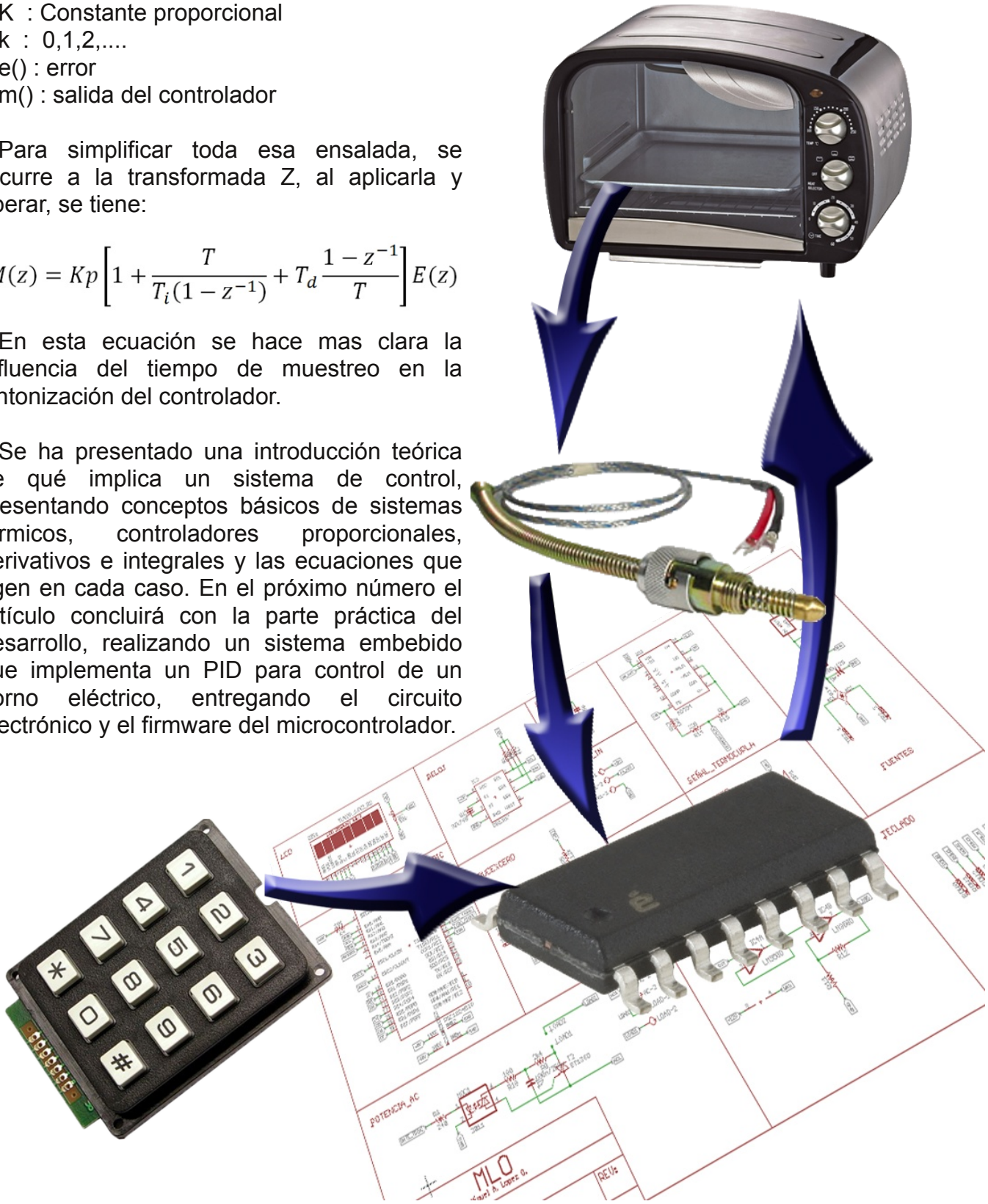
en donde:
T : Tiempo de muestreo
Ti : Tiempo integral
Td : Tiempo derivativo
K : Constante proporcional
k : 0,1,2,....
e() : error
m() : salida del controlador

Para simplificar toda esa ensalada, se recurre a la transformada Z, al aplicarla y operar, se tiene:

$$M(z) = Kp\left[1 + \frac{T}{T_i(1-z^{-1})} + T_d\frac{1-z^{-1}}{T}\right]E(z)$$

En esta ecuación se hace mas clara la influencia del tiempo de muestreo en la sintonización del controlador.

Se ha presentado una introducción teórica de qué implica un sistema de control, presentando conceptos básicos de sistemas térmicos, controladores proporcionales, derivativos e integrales y las ecuaciones que rigen en cada caso. En el próximo número el artículo concluirá con la parte práctica del desarrollo, realizando un sistema embebido que implementa un PID para control de un horno eléctrico, entregando el circuito electrónico y el firmware del microcontrolador.




NEOTEO
Ciencia y tecnología al extremo

Tutorial MPLAB C18

En esta segunda parte del tutorial de programación utilizando el compilador C18, trataremos como son almacenados los datos en la memoria, arreglos de variables, declaración y definición de funciones, y como siempre algunos ejemplos. A trabajar!

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

¿Cómo se guardan los datos en la memoria?

MPLAB C18 implementa el almacenamiento de los datos little-endian. Cuando una variable ocupa más de un byte, el byte menos significativo ocupa la posición de memoria más baja. Entonces al definir una variable de 32 bits:

```
long k=0 x59359712 ;
```

El resultado en la memoria de datos será:

Dirección	0x0100	0x0101	0x0102	0x0103
Contenido	0x12	0x97	0x35	0x59

Operaciones con variables de distintos tipos.

Cuando se evalúa una expresión donde las variables implicadas son de distinto tipos ocurre una conversión, ya sea implícita o explicita, para llevar ambos operando a un tipo común de datos con el que se pueda operar.

En la asignación de una expresión de un tipo dado a una variable de un tipo menor, la conversión se hace en forma automática. Por ejemplo:

```
unsigned char k,
float p =30.56;
k=p; // k= 30, p = 30.56. -
```

Aquí tenemos miembros de diferentes tamaños, por lo que habría un truncamiento del valor entero a la cantidad de bit que lo permita k. Si la parte entera excede el rango establecido por la variable k, el resultado no tendría lógica aparente.

Reglas de promoción automática de expresiones

Estas reglas dicen que el compilador haría estrictamente las conversiones necesarias para llevar todos los operandos al tipo del mayor. El resultado de evaluar una operación aritmética seria del tipo del mayor de sus operandos, en el sentido del tamaño en bits de cada objeto de datos. Por ejemplo:

```
unsigned char k;
float p;
k =5;
p=k /2; // p = 2
```

Por más que indiquemos que el resultado es float el truncamiento se produce en la evaluación del miembro derecho de la asignación.

Para resolver este problema existen dos

formas, una es escribir cualquiera de las contantes en punto flotante o utilizar el operador cast.

```
p= k /2.0; // p = 2.5
p =(( float )k /2); // p = 2.5
```

No es útil implementar el cast de la siguiente forma:

```
p= ( float )(k /2); // p = 2
```

Dado que primero se realiza la operación, y al resultado se aplica el cast, lo que no soluciona el problema.-

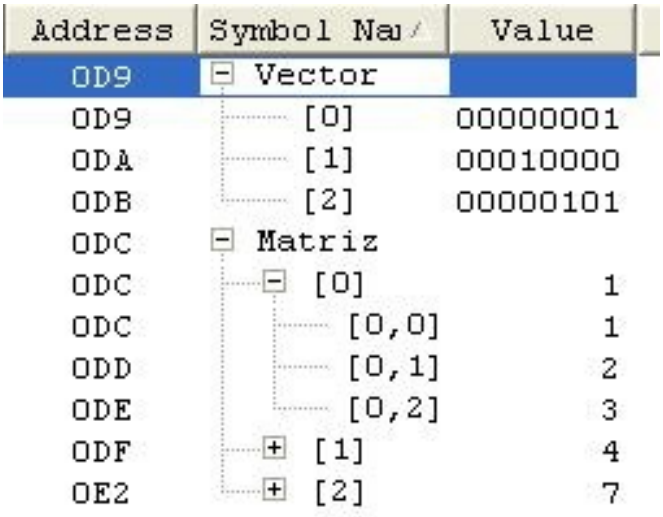
Arreglos de Variables

Nos permite trabajar con un conjunto de variables y acceder a cada una mediante un índice único que lo idéntifica. Todos los valores que contienen deben ser del mismo tipo. Los espacios de memoria que ocupan son contiguos y el primer elemento ocupa la dirección más baja. Algunos ejemplos de declaración y asignación de datos:

```
Vector de 5 elementos:
unsigned char Vector [5];
Matriz de 3 x 3:
unsigned char Matriz [3][3];
.
// Cargamos vector y matriz :
Vector [0]=156; // 0 es primer elemento
Matriz [1][1]=85;
// Leemos vector y matriz :
PORTB = Vector [4];
PORTB = Matriz [0][0];
```

También tenemos la posibilidad de asignar sus valores al momento de declarar los arreglos, eso se haría, por ejemplo, de la siguiente manera:

```
unsigned char Vector [3]={1 ,0 x10 ,0
b000101 };
unsigned char Matriz [3][3]={1 ,2 ,3 ,4 ,5 ,6
,7 ,8 ,9};
```



Funciones

La manera más elegante de construir nuestro programa es dividir la tarea a ejecutar en varias tareas más simples, de modo de facilitar el desarrollo y el entendimiento de la estructura del mismo. Otra ventaja que conlleva este proceder, es la reutilización de estos módulos creados con anterioridad, además de facilitar el trabajo en equipo.

Declaración y definición de funciones

La declaración da a conocer la función al compilador, a partir de su declaración ya se pueden realizar invocaciones a las mismas. La declaración de una función se conoce también como prototipo de la función. En el prototipo de una función se tienen que especificar los parámetros de la función, así como el tipo de dato que devuelve.

La definición estará en algún otro punto del programa, aquí se especifican las instrucciones que forman parte de la misma y que se utilizan para llevar a cabo la tarea específica de la función.

```
Tipo de retorno Nombre(Lista de parámetros)
```

Tipo de retorno: Representa el tipo del valor que devuelve la función. Si no devuelve ninguno de debe colocar void.

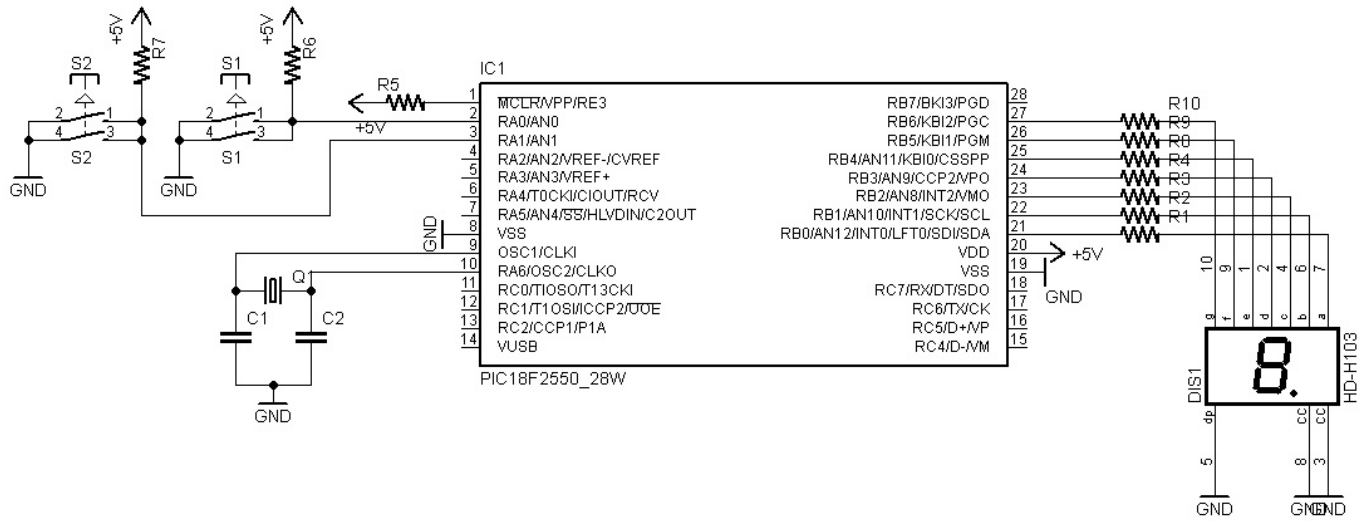
Nombre: indica el nombre que se le da a la función, se recomienda que esté relacionado con la tarea que llevara a cabo.

Lista de parámetros: se enlista el tipo de dato y el nombre de cada parámetro. En caso de no utilizar parámetros se deja el paréntesis vacío o se incluye la palabra void

Ejemplo 1: Control de display 7 segmentos

Objetivo: Utilizaremos dos pulsadores para incrementar, decrementar o resetear un conteo de 0 a 9 que mostraremos en un display de 7 segmentos de cátodo común. El reseteo ser a el caso en el que se presiona los dos pulsadores a la vez.

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
# include <p18f2550 .h>
/* ** Include para realizacion de demoras ** */
# include <delays .h>
/* ** Configuracion de los Fuses del microcontrolador ** */
# pragma config FOSC =XT_XT , FCMEN =OFF , IESO =OFF , CPUDIV = OSC1_PLL2
# pragma config PWRT =ON , BOR =OFF , BORV =0, WDT =OFF , WDTPS =32768
# pragma config MCLRE =ON , LPT1OSC =OFF , PBADEN =OFF , CCP2MX = OFF
# pragma config STVREN =OFF , LVP =OFF , XINST =OFF , DEBUG = OFF
# pragma config CP0 =OFF , CP1 =OFF , CP2 =OFF , CPB =OFF , CPD = OFF
# pragma config WRT0 =OFF , WRT1 =OFF , WRT2 = OFF
# pragma config WRTB =OFF , WRTC =OFF , WRTD = OFF
# pragma config EBTR0 =OFF , EBTR1 =OFF , EBTR2 =OFF , EBTRB = OFF
unsigned char i; // Para controlar vizualizacion del Display .-
const rom unsigned char Display7Seg [10]={0 x3F , 0x06 , 0x5B , 0x4F , 0x66 ,
                                          0x6D , 0x7D , 0x07 , 0xFF , 0 x6F };
void main ( void ){
   ADCON1 =0 x0F ;// Todos entrada / salida digitales .-
   TRISA =0 xFF ; // Todos como entrada .-
   TRISB =0 X00 ; // Todos como salida .-
   LATB =0 x3F ; // Comienza en cero .-
   i =0;
   while (1){
      // Si se presionan los 2 a la vez se resetea .-
      if( PORTAbits . RA0 ==0 & PORTAbits . RA1 ==0){
         i =0;
         // Cargamos en puerto valor de la tabla indicado por i.-
         LATB = Display7Seg [0];
         Delay10KTCYx (30);
      } else if( PORTAbits . RA0 ==0){ // Se incrementa cuenta .-
         ++i;
         // Volvemos a 0. Directamente se puede hacer if (++i ==10)
         if(i ==10){ i =0;}
         // Cargamos en puerto valor de la tabla indicado por i.-
         LATB = Display7Seg [i];
         Delay10KTCYx (30);
      } else if( PORTAbits . RA1 ==0){ // Se decrementa cuenta .-
         --i;
         if(i ==255){ i =9;} // Volvemos a 9.
         // Cargamos en puerto valor de la tabla indicado por i.-
         LATB = Display7Seg [i];
         Delay10KTCYx (30);
      }
   }
}
```



Hardware para ejemplo Nº 1.

Dos formas de incluir una función en nuestro código:

Realizando la declaración en el encabezado y después la definición en cualquier sector del programa.

```
// Declaración de la función
void Funcion ( void );
char OtraFuncion (char , int )
.
.
.
void main ( void ){
    .
    .
    .
    // Llamo a la función .
    Funcion ();
    m= OtraFuncion (5 ,5430);
}
// Definicion de la funcion ..
// ..( Puede estar en cualquier lugar del
programa )
void Funcion ( void ){
    // Sentencias
}
char OtraFuncion ( char k,int p){
    // Sentencias
}
```

Otra forma es no realizar la declaración de la función y realizar directamente la definición, pero esta tiene que estar si o si antes de su invocación.

```
    .
    .
    .
// Definición de la función
void Funcion ( void ){
    // Sentencias
}
char OtraFuncion ( char k,int p){
    // Sentencias
}
void main ( void ){
    .
    .
    .
    // Llamo a la función .
    Funcion ();
    m= OtraFuncion (5 ,5430);
}
```

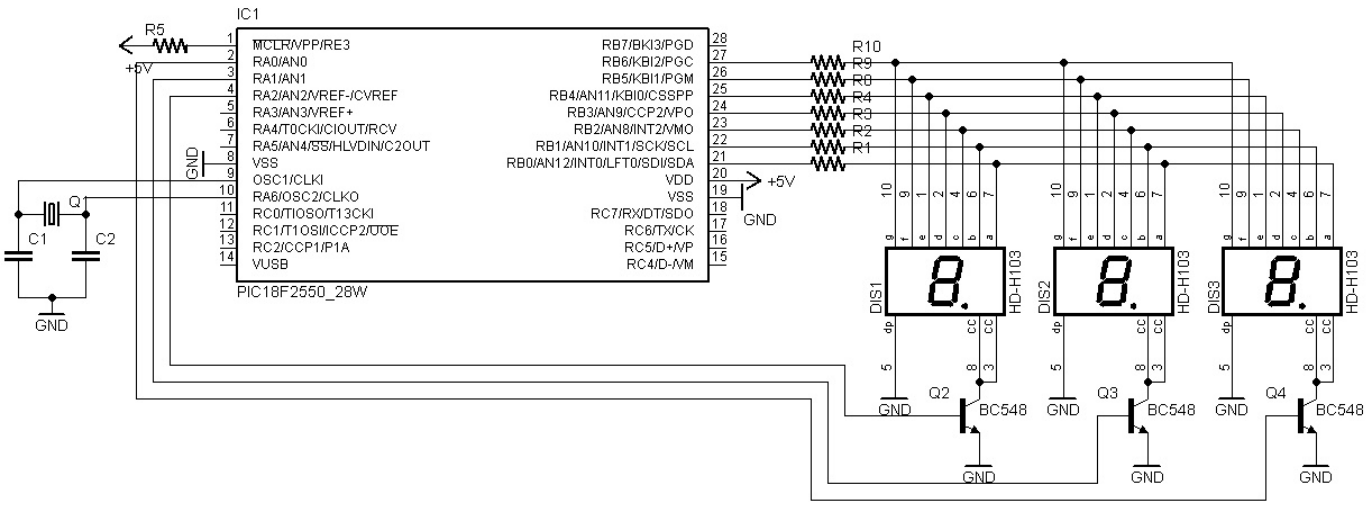
Ejemplo 2: Control de varios display, multiplexión de la señal

Objetivo: Controlar 3 display de 7 segmentos visualizando el conteo automático de 0 a 999.

Código:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
# include <p18f2550 .h>
/* ** Include para realizacion de demoras ** */
# include <delays .h>
/* ** Configuracion de los Fuses del microcontrolador ** */
# pragma config FOSC =XT_XT , FCMEN =OFF , IESO =OFF , CPUDIV = OSC1_PLL2
# pragma config PWRT =ON , BOR =OFF , BORV =0, WDT =OFF , WDTPS =32768
# pragma config MCLRE =ON , LPT1OSC =OFF , PBADEN =OFF , CCP2MX = OFF
# pragma config STVREN =OFF , LVP =OFF , XINST =OFF , DEBUG = OFF
# pragma config CP0 =OFF , CP1 =OFF , CP2 =OFF , CPB =OFF , CPD = OFF
# pragma config WRT0 =OFF , WRT1 =OFF , WRT2 = OFF
# pragma config WRTB =OFF , WRTC =OFF , WRTD = OFF
# pragma config EBTR0 =OFF , EBTR1 =OFF , EBTR2 =OFF , EBTRB = OFF
// Para controlar vizualizacion del Display .-
unsigned char i, Unidad , Decena , Centena ;
const rom unsigned char Display7Seg [10]={0 x3F , 0x06 , 0x5B , 0x4F , 0x66 ,
                                          0x6D , 0x7D , 0x07 , 0xFF , 0 x6F };
/* ** Declaracion de funcion a utilizar */
void Visualizacion ( void );
void main ( void ){
   ADCON1 =0 x0F ;// Todos entrada / salida digitales .-
   TRISA =0 xF0 ;
   TRISB =0 x00 ; // Todos como salida .-
   LATA =0 x00 ; // Comienza en 0
   Unidad =0;
   Decena =0;
   Centena =0;
   while (1){
      // Llamamos funcion que actualiza displays .-
      Visualizacion ();
      // Actualizamos cuenta .-
      ++ Unidad ;
      if( Unidad ==10){
         Unidad =0;
         ++ Decena ;
         if( Decena ==10){
            Decena =0;
            ++ Centena ;
         }
      }
   }
}
```

```
void Visualizacion ( void ){
   for (i =1;i <=20;++ i){
      // Cargamos en puerto valor de la tabla indicado por Unidad .-
      LATB = Display7Seg [ Unidad ];
      LATAbits . LATA0 =1; // Enciendo Display Unidad .-
      Delay1KTCYx (5); // Demora de 5 ms (XT =4 MHz )
      LATAbits . LATA0 =0;
      LATB = Display7Seg [ Decena ];
      LATAbits . LATA1 =1;
      Delay1KTCYx (5);
      LATAbits . LATA1 =0;
      LATB = Display7Seg [ Centena ];
      LATAbits . LATA2 =1;
      Delay1KTCYx (5);
      LATAbits . LATA2 =0; // Apago Display Centena .-
   }
}
```



Hardware para ejemplo Nº 2.

Preprocesador y Directivas del preprocesador

El preprocesador es el primer programa que se llama en la etapa de compilación de un programa. El preprocesador tiene su propio lenguaje y sus directivas inician con un #.

Las ventajas que tiene usar el preprocesador son:

*los programas son más fáciles de desarrollar,
*son más fáciles de leer,
*son más fáciles de modificar
*se pueden generalizar para varias arquitecturas o compiladores.

Directivas

#include

Esta directiva ya la hemos utilizado, se emplea para incluir archivos y suele darse al principio de los programas, porque en general se desea que su efecto alcance a todo el archivo fuente. Por esta razon los archivos preparados para ser incluidos se denominan headers o archivos de cabecera. En ellos se declaran las funciones que se implementan y definiciones para su implementacion.

#define

La directiva define tiene dos tipos de uso, como si fuera un objecto o como si fuera una

funcion. Las que se asemejan a funciones toman parametros mientras que las que se asemejan a objetos no.

Su formato es el siguiente:

```
# define <identificador > <lista de tokens a
reemplazar >
# define <identificador >(< lista de
parametros>)<lista de tokens a reemplazar >
```

Ejemplos:

```
# define e 2.718258
# define LED LATBbits . LATB5
# define Suma_10 (x) {x +=10;}
# define Suma_Divide_10 (x) {(x +=10);( x
/=10);}
```

Nota: Se debe tener cuidado en la implementacion de esta directiva. Las que se asemejan a funciones, pueden tomar parametros pero no se comportan como una funcion, ya que el preprocesador reemplaza un texto por otro, lo cual conlleva al mayor uso de la memoria de programa.

#ifdef, #ifndef, #else, #elif y #endif

Estas directivas son utilizadas para realizar compilaciones condicionadas, por ejemplo para hacer una libreria generalizada para varias arquitecturas, para ser utilizada por varios compiladores o simplemente para seleccionar el nivel de uso de cierto proyecto.

Ejemplos:

```
#if defined ( __18CXX )
   # include <p18cxxx .h>
# elif defined ( __dsPIC30F__ )
   # include <p30fxxxx .h>
# elif defined ( __dsPIC33F__ )
   # include <p33Fxxxx .h>
# elif defined ( __PIC24H__ )
   # include <p24Hxxxx .h>
# elif defined ( __PIC24F__ )
   # include <p24Fxxxx .h>
# endif
```

```
// Para no utilizar pin RW sacar comentario a
la siguiente linea .-
//# define __LCD_DONT_WAIT
# define LCD_PIN_E LATCbits . LATC4
# define LCD_PIN_RS LATCbits . LATC2
# ifndef __LCD_DONT_WAIT
   # define LCD_PIN_RW LATCbits . LATC3
# endif

// CCS
#if defined ( __PCH__ )
   char Buffer [512];
# endif
// C18
# ifdef __18CXX
   # pragma udata =0 x100
   unsigned char Buffer [512];
   # pragma udata
# endif
// C30 y C32
#if defined ( __C30__ ) || defined (
__PIC32MX__ )
   unsigned char __attribute__ (( aligned (4)))
Buffer [512];
# endif
```

Control de LCD

Para realizar el control de un LCD necesitamos usar la librería xlcd.h ubicada en C:/MCC18/h.

Esta librería es para un LCD con controlador Hitachi HD44780 o compatible, utilizando 8 o 4 bits de ancho de bus para envío/recepción de datos. El usuario debe proveer 3 delay para el correcto funcionamiento, DelayPORXLCD() de 15ms, DelayXLCD() de 5ms y DelayFor18TCY() de 18 Tcy.

En este caso no vamos a modificar la librería, eso lo vamos a dejar para más adelante, pues lo vamos a controlar con el puerto B, configuración por defecto, pero en el caso de que se modifique sugiero siempre respaldarla con una copia de seguridad.

Ejemplo Nº 3: Control de varios display, utilizando directivas de preprocesador

Objetivo: Se realiza el mismo ejemplo anterior pero utilizando directivas de preprocesador, definiendo el hardware a utilizar. Realizándolo de esta manera es mucho más sencillo realizar cualquier modificación en el código del hardware utilizado.

Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
# include <p18f2550 .h>
/* ** Include para realizacion de demoras ** */
# include <delays .h>
/* ** Configuracion de los Fuses del microcontrolador ** */
# pragma config FOSC =XT_XT , FCMEN =OFF , IESO =OFF , CPUDIV = OSC1_PLL2
# pragma config PWRT =ON , BOR =OFF , BORV =0, WDT =OFF , WDTPS =32768
# pragma config MCLRE =ON , LPT1OSC =OFF , PBADEN =OFF , CCP2MX = OFF
# pragma config STVREN =OFF , LVP =OFF , XINST =OFF , DEBUG = OFF
# pragma config CP0 =OFF , CP1 =OFF , CP2 =OFF , CPB =OFF , CPD = OFF
# pragma config WRT0 =OFF , WRT1 =OFF , WRT2 = OFF
# pragma config WRTB =OFF , WRTC =OFF , WRTD = OFF
# pragma config EBTR0 =OFF , EBTR1 =OFF , EBTR2 =OFF , EBTRB = OFF
/* ** Definiciones para preprocesador ** */
// Pin para control display visualizador de unidades .
# define DISPLAY_PIN_U LATAbits . LATA0
# define DISPLAY_PIN_D LATAbits . LATA1
# define DISPLAY_PIN_C LATAbits . LATA2
# define DISPLAY_TRIS_U TRISAbits . TRISA0 //
# define DISPLAY_TRIS_D TRISAbits . TRISA1
# define DISPLAY_TRIS_C TRISAbits . TRISA2
// Puerto para enviar data a displays .
# define DISPLAY_DATA LATB
# define DISPLAY_TRIS_DATA TRISB
/* ************************************ */
// Para controlar vizualizacion del Display .-
unsigned char i, Unidad , Decena , Centena ;
const rom unsigned char Display7Seg [10]={0 x3F , 0x06 , 0x5B , 0x4F , 0x66 ,
                                          0x6D , 0x7D , 0x07 , 0xFF , 0 x6F };
/* ** Declaracion de funcion a utilizar */
void Visualizacion ( void );
void main ( void ){
   ADCON1 =0 x0F ;// Todos entrada / salida digitales .-
   DISPLAY_TRIS_DATA =0 x00 :
   DISPLAY_TRIS_U =0;
   DISPLAY_TRIS_D =0;
   DISPLAY_TRIS_C =0;
   DISPLAY_PIN_U =0;
   DISPLAY_PIN_D =0;
   DISPLAY_PIN_C =0;
   Unidad =0;
   Decena =0;
   Centena =0;
```

```
    while (1){
        // Llamamos funcion que actualiza displays .-
        Visualizacion ();
        // Actualizamos cuenta .-
        ++ Unidad ;
        if( Unidad ==10){
            Unidad =0;
            ++ Decena ;
            if( Decena ==10){
                Decena =0;
                ++ Centena ;
            }
        }
    }
}
void Visualizacion ( void ){
    for (i =1;i <=20;++ i){
        // Cargamos en puerto valor de la tabla indicado por Unidad .-
        DISPLAY_DATA = Display7Seg [ Unidad ];
        DISPLAY_PIN_U =1; // Enciendo Display Unidad .-
        Delay1KTCYx (5); // Demora de 5 ms (XT =4 MHz )
        DISPLAY_PIN_U =0;
        DISPLAY_DATA = Display7Seg [ Decena ];
        DISPLAY_PIN_D =1;
        Delay1KTCYx (5);
        DISPLAY_PIN_D =0;
        DISPLAY_DATA = Display7Seg [ Centena ];
        DISPLAY_PIN_C =1;
        Delay1KTCYx (5);
        DISPLAY_PIN_C =0; // Apago Display Centena .-
    }
}
```

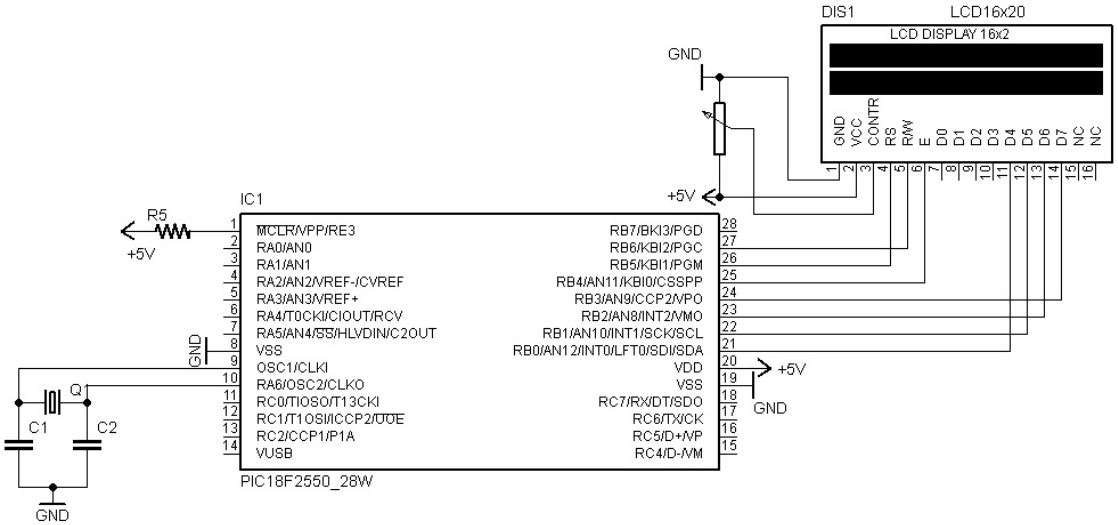
Ejemplo Nº 4: Control de LCD

Objetivo: Vamos a escribir un simple mensaje en un LCD. Se crearan 2 funciones adicionales para un mejor el control, la primeria seria el envío de comandos, con una previa espera de disponibilidad del LCD y la segunda es para controlar la posición del cursor en el LCD.

Código:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
# include <p18f2550 .h>
/* ** Includes ** */
# include <delays .h>
# include <xlcd .h>
/* ** Configuracion de los Fuses del microcontrolador ** */
# pragma config FOSC =XT_XT , FCMEN =OFF , IESO =OFF , CPUDIV = OSC1_PLL2
# pragma config PWRT =ON , BOR =OFF , BORV =0, WDT =OFF , WDTPS =32768
# pragma config MCLRE =ON , LPT1OSC =OFF , PBADEN =OFF , CCP2MX = OFF
# pragma config STVREN =OFF , LVP =OFF , XINST =OFF , DEBUG = OFF
# pragma config CP0 =OFF , CP1 =OFF , CP2 =OFF , CPB =OFF , CPD = OFF
```

```
# pragma config WRT0 =OFF , WRT1 =OFF , WRT2 = OFF
# pragma config WRTB =OFF , WRTC =OFF , WRTD = OFF
# pragma config EBTR0 =OFF , EBTR1 =OFF , EBTR2 =OFF , EBTRB = OFF
void DelayFor18TCY ( void ){
   Delay10TCYx (2);
}
void DelayPORXLCD ( void ){
   Delay1KTCYx (15);
}
void DelayXLCD ( void ){
   Delay1KTCYx (2);
}
// Envia comando al LCD
void comandXLCD ( unsigned char a){
   BusyXLCD ();
   WriteCmdXLCD (a);
}
// Ubica cursor en (x = Posicion en linea , y = N de linea )
void gotoxyXLCD ( unsigned char x, unsigned char y){
   unsigned char direccion ;
   if(y != 1)
      direccion = 0 x40 ;
   else
      direccion =0;
   direccion += x -1;
   comandXLCD (0 x80 | direccion );
}
void main ( void ){
   OpenXLCD ( FOUR_BIT & LINES_5X7 ); // Iniciamos LCD .-
   comandXLCD (0 x06 ); // Nos aseguramos incremento de direccion , display fijo
   comandXLCD (0 x0C ); // Encendemos LCD .-
   putrsXLCD (" Probando LCD ");
   gotoxyXLCD (1 ,2); // Pasamos al oriden del Linea 2.-
   putrsXLCD ("Por Suky ");
   while (1){ // Bucle infinito .
   }
}
```



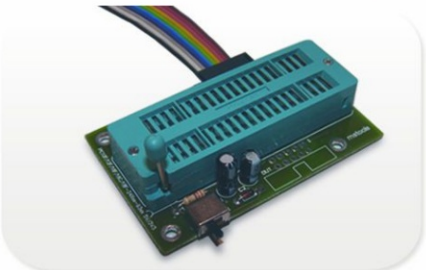
Hardware para ejemplo Nº 4.

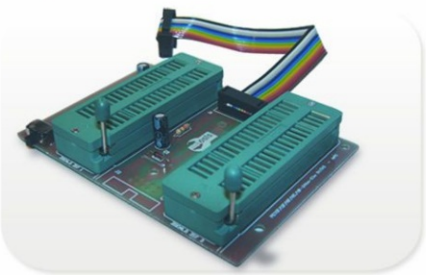
GTP-USB+

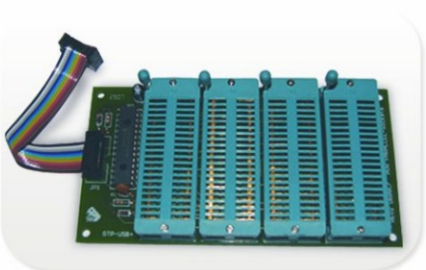
Grabador de Microcontroladores y Memorias por puerto USB 2.0
Hardware oficial del software Winpic800.

- **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.

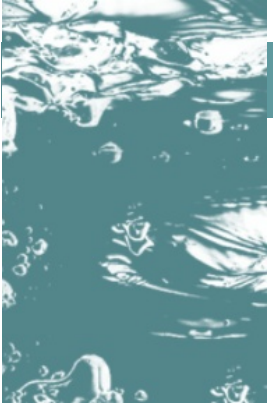
Módulos ZIF disponibles para todas las familias







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



Tutorial ASM
...desde 0

Todo programa consiste en una serie de instrucciones, que sirven para indicarle al dispositivo que es lo que debe hacer para lograr nuestro cometido. Los PIC de la gama media disponen de un repertorio de 35 instrucciones, que debemos conocer tan bien como la palma de nuestra mano y a las que dedicamos esta segunda parte del Tutorial.

// por: David (Leon Pic) //
david@meteorologiafacil.com.ar

INSTRUCCIONES: Breve introducción

Cada instrucción tiene un ancho de 14 Bits, es por eso que la memoria de programa tiene el mismo ancho. Justamente para poder alojar cada instrucción.

Las instrucciones, están divididas en tres grupos. Los cuales son:

- **Byte-Oriented** operation (Orientada a Byte)
- **Bit-Oriented** operation (Orientada a Bit)
- **Literal and Control** operation (Operación con Literal y Control)

Primer grupo: Byte-Oriented operation

Cada instrucción de este grupo está compuesta por:

- OPCODE (Código)
- DESTINATION (Destino)
- FILE REGISTER ADDRESS (Dirección del archivo de registro)

OPCODE o código, es el código de cada instrucción y que es única para cada instrucción. Está formada por los bits del 13 al 8.

DESTINATION o destino, indica en dónde se va a guardar el dato. Por ejemplo, si hacemos una suma, tenemos dos opciones dónde guardarlo, una puede ser el registro W y la otra opción puede ser otro registro cualquiera o una posición de la RAM. Está formada por el séptimo bit.

La constante que nos indica esto es la letra d. Si esta letra es 0, la operación se guardará en el registro W. En cambio si vale 1, la operación se guardará en el registro o posición de memoria que estemos trabajando al momento de usar una instrucción.

Hay instrucciones, como veremos más adelante, que no es necesario indicar dónde queremos guardar la operación, ya que se hace en forma automática; y hay otras instrucciones que si no se indica el destino, nos puede dar un error o una advertencia al compilar y el compilador elegirá él el destino pudiendo ocasionar que nuestro programa falle.

Y por último, tenemos FILE REGISTER ADDRESS que se carga con la dirección del registro a ser guardado. Está formada por los bits 6 al 0. La constante que nos indica esto, es la letra f

Segundo grupo: Bit-Oriented operation

Cada instrucción de este grupo está compuesta por:

- OPCODE (Código)
- BIT ADDRESS (Bit de dirección)
- FILE REGISTER ADDRESS (Dirección del archivo de registro)

OPCODE es igual al primer grupo. Está formado por los bits 13 al 10.

El BIT ADDRESS, se utiliza para direccionar la operación. Está formado por los bits 9 al 7. Como pueden observar, se sacrificó bit del opcode para dárselo al bit address. La constante que nos indica esto es la letra b

Y por último tenemos FILE REGISTER ADDRESS, que es igual al primer grupo. Está formado por los bits 6 al 0. Igual que en el primer grupo, la constante que nos indica esto es la letra f.

Tercer grupo: Literal and Control

Cada instrucción de este grupo, está compuesta por:

- OPCODE
- LITERAL

OPCODE es igual que en el primer grupo. Está compuesta por los bits 13 al 8. Excepto para las instrucciones CALL y GOTO que está compuesta por los bits 13 al 11 (prestar mucha atención a esto, cuando veamos estas dos instrucciones entenderán la importancia).

Y el LITERAL que puede ser un valor, por ejemplo para sumar, para restar, para cargar al registro W, en fin, un número decimal, binario o hexadecimal. O puede ser un valor de dirección a dónde apuntar para las instrucciones CALL y GOTO.

Está compuesta por los bits 7 al 0. Excepto para las instrucciones CALL y GOTO que está compuesta por los bit 10 al 0 (prestar mucha atención a esto, cuando veamos estas dos instrucciones entenderán la importancia).

En la página siguiente podemos ver una imagen con las 35 instrucciones agrupadas por los tres grupos. Se trata de la Table 13-2, con el Set de Instrucciones de la familia PIC16F87X

En la tabla vemos 5 columnas. De izquierda a derecha tenemos la primera columna llamada Mnemonic Operands que son las instrucciones con que se puede programar dicho pic. Presten atención que están divididos por los tres grupos que vimos en detalle.

En la segunda columna tenemos la descripción de cada instrucción, que dicho sea de paso, lo veremos a continuación.

En la tercera columna vemos como está formada en código binario cada instrucción con sus constantes descritas anteriormente. Vemos también la cantidad de ciclos de máquina que consume cada instrucción para ser completado. Hay instrucciones que consumen 1 o 2 ciclos para completar su trabajo y otras que consumen 1 y 2 ciclos y que figura así 1(2). Estas instrucciones son de salto incondicional cuando se cumple cierta lógica. Mientras no produzcan un salto, consumen un ciclo, pero en el momento de producir un salto, consumen dos ciclos de reloj.

En la cuarta columna vemos si afecta o no algún bit del registro Status. En caso de que afecte, en la tabla nos muestra cuales bits se ven afectados después de ser ejecutado esa instrucción.

Y en la quinta columna, hace referencia a la nota que está al fin de la tabla. Me imagino que ya tiene un diccionario Ingles-Español en la mano ¿verdad? Y sino es así, pues tendrán que esperar para más adelante.

TABLE 13-2: PIC16F87X INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

Note: Additional information on the mid-range instruction set is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

Las instrucciones

Les voy a arruinar el momento de alegría. Las instrucciones hay que estudiarlas de memoria. Si, leyeron bien, de **memoria**. Lo que tienen que saber sobre las instrucciones, es como se escriben, que hace cada instrucción y lo más importante que bit del **REGISTRO STATUS** afecta.

Vamos a ir viéndolo por orden alfabético y con sencillos ejemplos. Y otra cosita más, como es de esperarse, están en INGLÉS o son abreviaturas pero en INGLÉS.

Recordemos que: .123 o D'123' es en decimal; 0x7B o 7Bh o H'7B' es en Hexadecimal; B'01111011' es en binario.

ADDLW

Suma un valor designado por el programador al registro W

```
ADDLW   .128
```

Si W tenía cargado un valor = .5, después de la instrucción W tiene cargado el valor .133

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0
DC Se pone a 1 si hubo un acarreo del bit 3 al 4
C Se pone a 1 si hubo desbordamiento, o sea, cuando se supera H'FF'

ADDWF

Suma el valor del registro W con el valor de un registro cualquiera. El destino de esta suma, lo elige el programador.

```
ADDWF   TEMP,W
```

Si W tenía guardado .133 y la posición de la RAM llamada TEMP tenía el valor cargado con .2, W vale .135 y TEMP continúa valiendo .2

Ahora si hubiera puesto así:

```
ADDWF   TEMP,F
```

TEMP valdría .135 y W valdría .133

NOTA: Para indicar la dirección de dónde se guarda, también se puede poner 0 o 1 en vez de W o F. 0 corresponde guardarlo en el registro W y 1 en el registro TEMP (para este caso).

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0
DC Se pone a 1 si hubo un acarreo del bit 3 al 4
C Se pone a 1 si hubo desbordamiento, o sea, cuando se supera H'FF'

ANDWF

Realiza la operación AND entre W y un registro designado por el programador. El destino de esta operación lo elije el programador.

```
ANDWF   TEMP,F
```

Si antes de la instrucción W vale B'11100011' y TEMP vale B'00111010' Después de la instrucción TEMP vale B'00100010' y W vale B'11100011'

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0

BCF

Pone a 0 el bit de un registro. El bit debe ser indicado por el programador.

```
BCF     TEMP,2
```

Antes de la instrucción TEMP vale B'11111111'. Después de la instrucción TEMP vale B'11111011'
Para recordar, **B**it **C**lear es borrar **F**ile es archivo o registro

No afecta a ningún bit del registro Status.

BSF

Pone a 1 el bit de un registro. El bit debe ser indicado por el programador.

```
BSF     TEMP,0
```

Antes de la instrucción TEMP vale B'01110110'. Después de la instrucción TEMP vale B'01110111'

Para recordar, **B**it **S**et es poner a 1 **F**ile Archivo o registro

No afecta a ningún Bit del registro Status.

BTFSC

Salta un línea si el bit de un registro es cero. El bit debe ser indicado por el programador.

```
BTFSC   TEMP,5
BCF     PORTA,0
BSF     PORTB,0
```

Caso 1:

TEMP vale B'00011110'. El CP analizará solo el Bit 5 del registro TEMP, como es 0, salta la instrucción BCF PORTA,0 y ejecuta la siguiente línea que es BSF PORTB,0 y continua haciendo la instrucción.

Caso 2:

TEMP vale B'00111000'. El CP analizará solo el Bit 5 del registro TEMP, como es 1 no salta la instrucción y hará la instrucción BCF PORTA,0 y luego continua con la instrucción BSF PORTB,0

No afecta a ningún Bit del registro Status.

BTFSS

Salta una línea si el bit de un registro es 1. El bit debe ser indicado por el programador.

```
BTFSS   TEMP,3
ADDWF   PORTC
ANDWF   NODO
```

Caso 1:

TEMP vale B'01101100'. El CP analizará solo el Bit 3 del registro TEMP, como es 1, salta la instrucción ADDWF PORTC y ejecuta la siguiente línea que es ANDWF NODO y continúa haciendo la instrucción.

Caso 2:

TEMP vale B'11110000'. El CP analizará solo el Bit 3 del registro TEMP, como es 0 no salta la instrucción y hará la instrucción ADDWF PORTC y luego continúa con la instrucción ANDWF NODO.

No afecta a ningún Bit del registro Status.

Normalmente, continuando las instrucciones BTFSS y/o BTFSC va un GOTO o CALL pero no la he puesto porque aún no se explicaron estas instrucciones.

CALL

Se dirige a una dirección de la memoria de programa designado por el programador. En otras palabras, se utiliza para dirigirse a una rutina o tarea. Su principal ventaja es que una vez que finalizó la tarea, vuelve al punto siguiente desde dónde se llamo.

```
CALL    ESC_PORTB
```

No afecta a ningún bit del registro Status.

Hacemos una excepción con respecto a ver las instrucciones por orden alfabético y veremos ahora la instrucción GOTO.

GOTO

Se dirige a una dirección de la memoria de programa designada por el programador. En otras palabras, se utiliza para saltar instrucciones que no queremos que se ejecuten. A diferencia de la instrucción CALL, no hay forma de volver cuando se ejecuta la instrucción.

```
GOTO    INICIO
```

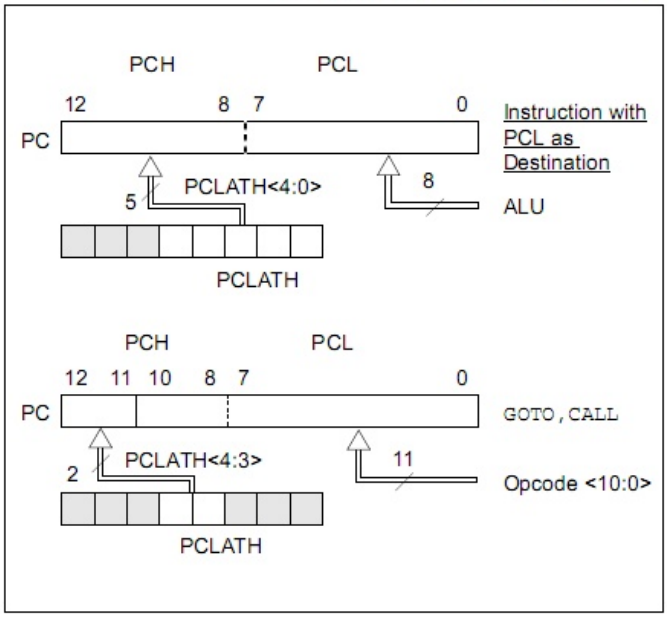
No afecta a ningún bit del registro Status.

Extendiendo la explicación

Vamos a hablar del PC para entender bien sobre estas instrucciones. La excelente explicación que continua, por desgracia no es mía (ojalá fuera mi explicación), son de las personas Maunix y BrunoF (del foro Todopic)

En la siguiente imagen, vemos el diagrama de bloques del PC o CP.

FIGURE 2-5: LOADING OF PC IN DIFFERENT SITUATIONS



El PC es de 13 bits en este caso (8kwords). 14 son los bits de cada "word" o instrucción que se graban en cada posición de la FLASH (memoria de programa).

El PC se reparte en: sus 8 bits de menor peso en el registro PCL, y los 5 restantes en el registro PCLATH.

Los pics al tener un set de instrucciones reducido no pueden en una sola instrucción meter todos los bits necesarios para direccionar toda su memoria.

El Program Counter son 2 registros, el PCHigh y PCLow . Cuando haces un CALL o un GOTO, solo se rellenan 11 bits (los 8 del PClow y 3 del PChigh) y los restantes 2 los rellena con el PCLATH (para completar los 13bits).

El STACK (pila) tiene toda la dirección, no solo parcial. Si haces un CALL desde la pagina 0 a la página 3 y luego un RETURN el código **SI volverá a la página 0**, pero el PCLATH sigue apuntando a la página 3, entonces si usas otro GOTO o CALL, debes tener en cuenta de modificar el PCLATH.

Entonces, dijimos que:

El PC = ProgramCounter o CP = Contador de Programa, tiene 13 bits; del 0 al 12.

Al ejecutar un CALL o un GOTO, se copian del 0 al 10, y los otros 2 bits se copian del registro PCLATH. El PCLATH solo estará allí para esa situación.

En un RETURN o RETFIE la micro-electronica del PIC, pega la dirección del PC que estaba guardada.

Lo vemos con un ejemplo:

1.

```
STACK = vacío
PC = 0x00A0
PCLATH = 0b000011000
Ejecutas un CALL 0x230
```

2. El STACK tiene en su posición 0 la dirección 0x00A0.

```
PC = 111000110000
```

3. Se ejecuta la subrutina y en ese punto el PC ya quedó en

```
PC = 111000110111
```

4. Viene un RETURN. La microelectrónica del PIC copiará el STACK tal cual en el Program Counter + 1

Valor STACK 0x00A0 + 1 --> PC = 0x00A1

5. El código sigue ejecutándose en la página 0 pero hay que tener en cuenta que el PCLATH apunta a la página 3 por ello si harás otro CALL o GOTO, deberás cambiar de nuevo el PCLATH si la subrutina no está en la página 3.

Para cerrar el tema

Vamos a entrar a todo detalle en el Program Counter (PC) para que se vayan todas las dudas ya que es muy importante. Vayamos al

tema del PC, Computed Goto (lo que algunos llaman "tabla"), CALL, RETURN y GOTO.

El Program Counter (PC) está conformado en esta familia de uC (y refiriéndonos a la familia 16F, las otras poseen más o menos bits implementados) por 13 bits repartidos entre dos registros: PCH y PCL.

El PCL es legible/escribible directamente a través del registro físico PCL (valga la redundancia). En cambio, el PCH no es directamente accesible. No puede ser leído, y sólo puede se grabado mediante un buffer que contiene el valor temporalmente (oh! aquí aparece nuestro famoso PCLATH). Entonces, recordar: **El PCLATH es sólo un buffer temporal que almacena los 5 bits de mayor peso del PC para ser escritos cuando se ejecute una instrucción que lo requiera**.

Ahora, hay dos situaciones posibles en las que el PC debe ser cargado de manera distinta: una es cuando queremos trabajar con tablas y otra cuando realizamos un CALL o un GOTO que no esté en el mismo banco.

1era situación: Tabla (Computed Goto)

La tabla es una situación de uso del PC en la que se afecta directamente al registro PCL. **Cuando se afecte directamente al PCL mediante una instrucción, es necesario que el usuario asegure que PCLATH tenga sus 5 bits pre-cargados adecuadamente**.

Hago un ejemplo:

Mal:

```
   org 0x000
   movlw 0x01
   call tabla

   org 0x300
tabla
   addwf  PCL,F
   retlw 0x03
   retlw 0x01
   retlw 0x0F
   .....
```

Bien:

```
   org 0x000
   movlw 0x03
   movwf  PCLATH
   movlw 0x01
   call tabla

   org 0x300
tabla
   addwf  PCL,F
   retlw 0x03
   retlw 0x01
   retlw 0x0F
   .....
```

Mejor:

```
   org 0x000
   pageselw  tabla
   movlw 0x01
   call tabla

   org 0x300
tabla
   addwf  PCL,F
   retlw 0x03
   retlw 0x01
   retlw 0x0F
   .....
```

Pageselw es una instrucción del MPASM que genera dos instrucciones: un movlw literal y un movwf PCLATH. El valor del literal es automáticamente seleccionado por el ensamblador según la etiqueta (o posición de memoria) que se le especifique.

En el caso anterior pageselw tabla generaría estas dos instrucciones:

```
   movlw 0x03
   movwf PCLATH
```

Si no aseguramos que los 5 bits del PCLATH estén correctamente seteados al momento de afectar al PCL mediante alguna instrucción (generalmente es la addwf, pero puede usarse subwf y muchas otras) entonces el programa saltará a una posición indeseada.

2da situación: CALL y GOTO

En esta familia de uC, cada instrucción es codificada en 14 bits. En el caso de las instrucciones CALL y GOTO, su estructura es la siguiente:

F2 F1 F0 K10 K9 K8 K7 K6 K5 K4 K3 K2 K1 K0

Donde las F indican cuál instrucción es la que debe ejecutarse (100 para la CALL 101 para la GOTO), y las K corresponden a la dirección a la cual queremos llamar (con un CALL) o saltar (con un GOTO).

Aquí se ve claramente un problema. Podemos ver que un CALL o un GOTO sólo almacenan 11 bits de la dirección a la cual debe ir. 11 bits es un máximo 2048 posiciones. ¿Qué pasa cuando un uC posee más de 2k de memoria Flash entonces? Por ejemplo, un 16F877A posee 8k de memoria Flash. ¿Cómo haría para llamar a una subrutina que está más allá de la posición 2047 de la flash?

La solución nuevamente se encuentra en el PCLATH (y es nuevamente el usuario el que tiene el deber de pre-cargar el valor adecuado).

Entonces, dijimos que el PC contiene 13 bits de longitud. 13 bits son hasta 8kwords (un word es en esta familia un conjunto de 14 bits que conforman una instrucción la cual se aloja en la memoria FLASH del uC). Un CALL o un GOTO sólo contienen los 11 bits de menor peso de la dirección a la cual ir, por lo que los 2 bits restantes deberán ser pre-cargados en los bits 4 y 3 del registro PCLATH por el usuario programador.

Cuando se ejecuta una instrucción CALL o GOTO, es imprescindible que el registro PCLATH esté correctamente precargado. La instrucción a la que el uC irá estará conformada por 13 bits y ellos serán:

PCLATH,4 PCLATH,3 K10 K9 K8 K7 K6 K5 K4 K3 K2 K1 K0

Cabe mencionar que el uC carga a PC<10:0> con el valor pasado por los 11 bits de K, y a PC<12:11> con el valor de los bits PCLATH<4:3>. El registro PCLATH no es modificado de ninguna manera. Sólo se leen esos dos bits.

Por ejemplo, en un uC de 8kWords hay 4 páginas. Una página cada 2048 words. Si se está en una página y se quiere ir a otra es necesario precargar antes dichos bits del PCLATH para poder hacerlo.

El usuario no debe preocuparse por precargar el PCLATH en dos situaciones:

- Si el uC no posee más de 2kWords de memoria Flash;
- O si en el código creado por el usuario, no se utiliza la memoria FLASH más allá de la posición 2047(0x7FF).

Si ocurre al menos uno de esos dos casos, es suficiente con asegurar que los bits PCLATH<4:3> se encuentren ambos en cero.

Vamos con un par de ejemplos:

Mal:

```
  org 0x000         ;Esto es página0
  call cruzo

  org 0x800         ;Esto ya es el página1
cruzo
  retlw 0xFF
  .....
```

Bien:

```
  org 0x000          ;Esto es página0
  movlw 0x08
  movwf PCLATH
  call cruzo

  org 0x800          ;Esto ya es el página1
cruzo
   retlw 0xFF
  .....
```

```
Mejor:
   org 0x000            ;Esto es página0
   pagesel cruzo        ;automaticamente
                         seleccionar banco
   call cruzo

   org 0x800            ;Esto ya es el página1
cruzo
   retlw 0xFF
   .....
```

Pagesel es una instrucción del MPASM que genera dos instrucciones: un bcf/bsf PCLATH,3 y un bcf/bsf PCLATH,4. El software ensamblador selecciona automáticamente la instrucción bcf o bsf según el banco en el cual se encuentra la etiqueta (o posición de memoria) que se le especifique. En el caso anterior pagesel cruzo generaría estas dos instrucciones:

```
bsf     PCLATH,3
bcf     PCLATH,4
```

Ya que la subrutina cruzo se encuentra en la página1.

Finalmente, cuando se ejecuta una instrucción CALL, se carga en el STACK el valor de la posición actual más 1(es decir, se guarda en el STACK el valor PC+1). Se guardan los 13 bits, por lo que durante las instrucciones RETURN, RETLW y RETFIE no es necesario precargar al PCLATH.

Para más información, ver el esquema sección 2.3 del datasheet de los PIC16F87XA que habla de cómo cargar al PC según cada situación.

CLRF

Borra el contenido de un registro seleccionado por el programador. La forma en que lo hace, pone en 0 los 8 bit del registro. Este registro, puede ser cualquiera de la posición de la RAM.

```
CLRF    PORTB
```

Antes de la instrucción PORTB vale B'11000111'. Después de la instrucción PORTB vale B'00000000'

Afecta a:
Z Se pone a 1

CLRW

Borra al registro W. La forma en que lo hace, pone en 0 los 8 bit del registro.

```
CLRW
```

Antes de la instrucción W vale B'00000111'. Después de la instrucción W vale B'00000000'.

Afecta a:
Z Se pone a 1

CLRWDT

Borra al WDT. La forma en que lo hace, pone en 0 al mismo.

```
CLRWDT
```

Antes de la instrucción WDT vale B'11111110'. Después de la instrucción vale B'0000000'.

Afecta a:
TO (neg) Se pone a 1
PD (neg) Se pone a 1

NOTA: El WDT o el temporizador perro guardián, sirve para destrabar al PIC. Cada vez que se desborda, o sea, cada vez que pasa de H'FF' a H'00', produce un reset y como es un reset, se dirige a la posición 0h de la memoria de programa.

La forma de trabajar con el, es ir poniendo en lugares estratégicos la instrucción ya explicada, de esta manera evitamos el desborde del contador. Si el CP se traba en algún bucle o algo similar, al no limpiar el contador, el WDT desbordará y llevará al CP a la pocición 0h de la memoria de programa.
Muchas veces se evita de usar esta herramienta por no tener que calcular por todo el programa dónde y cuando limpiar al WDT. Es recomendable su uso.

COMF

Realiza el complemento de un registro.

```
COMF    TEMP,F
```

Si TEMP tenía guardado B'00111101' luego de ejecutar la instrucción TEMP vale B'11000010'. Nótese, que aquí también podemos elegir el destino y esto nos deja guardarlo en el registro W si así lo requerimos.

Afecta a:
Z Se pone a 1 si la operación da 0

DECF

Decrementa en una unidad, o lo que es lo mismo, resta 1 el contenido de un registro

```
DECF    DECENA,W
```

Si antes de la instrucción DECENA vale .255, después de la instrucción W vale .254 y DECENA vale .255 Si por el contrario, hubiéramos elegido el destino F, después de la instrucción DECENA vale .254

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0

DECFSZ

Decrementa en uno, o lo que es lo mismo, resta en 1 el contenido de un registro y cuando este vale 0, el CP salta una instrucción

```
LOOP    DECFSZ  TEMP
        GOTO    LOOP
        BCF     PORTB,0
```

El CP descontará en 1 el registro TEMP y evalúa el valor, si no es cero, ejecuta línea siguiente que es GOTO LOOP, el cual se dirige de nuevo a la línea LOOP DECFSZ TEMP el cual volverá a descontar en 1 y evalúa el valor, si es cero salta la línea GOTO LOOP y ejecuta la instrucción BCF PORTB. Esta última línea, el programador pondrá la instrucción que necesite ejecutar.

Este pequeño programa que acabamos de ver, es un temporizador o un retardo que tardará en salir del bucle dependiendo de la frecuencia de reloj y el valor cargado en TEMP.

NOTA: Esta instrucción, también hay que elegirle el destino. En el caso que no se exprese, como en este caso, el MPLAB dará por sentado que el resultado se guardará en el registro F y no en W.

No afecta a ningún bit del registro STATUS.

INCF

Incrementa en 1, o suma 1, el contenido de un registro elegido por el programador.

```
INCF    INDF,F
```

Si antes de la instrucción INDF vale H'29', después de la instrucción INDF vale H'2A'. Nótese que también podemos elegir el destino. Si hubiéramos elegido W, después de la instrucción W vale H'2A' y INDF vale H'29'.

Afecta a:
Z se pone a 1 si el resultado es 0

INCFSZ

Incrementa en 1, o suma en 1, el contenido de un registro elegido por el programador y cuando este es 0, el CP salta una instrucción.

```
VOLVER  INCFSZ  CONTADOR
        GOTO    VOLVER
        INCF    PORTA
```

El CP incrementará en 1 el registro CONTADOR y evalúa el valor, si no es cero, ejecuta línea siguiente que es GOTO VOLVER, el cual se dirige de nuevo a la línea VOLVER INCFSZ CONTADOR el cual volverá a incrementar en 1 y evalúa el valor, si es cero salta la línea GOTO VOLVER y ejecuta la instrucción INCF PORTA. Esta última línea, el programador pondrá la instrucción que necesite ejecutar.

Este pequeño programa que acabamos de ver, es un temporizador o un retardo que tardará en salir del bucle dependiendo de la frecuencia de reloj y el valor cargado en

CONTADOR. Normalmente, se utiliza el retardo con DECFSZ pero este también es válido.

NOTA: Esta instrucción, también hay que elegirle el destino. En el caso que no se exprese, como en este caso, el MPLAB dará por sentado que el resultado se guardará en el registro F y no en W.

No afecta a ningún bit del registro STATUS.

IORLW

Realiza la operación OR entre W y un literal elegido por el programador. El resultado se guarda en W. La operación es W OR L.

Si antes de la instrucción W vale B'01110100' y el literal elegido es B'00011111', después de la instrucción W vale B'01111111'.

Afecta a:
Z se pone a 1 si la operación da 0

IORWF

Realiza la operación lógica OR entre el registro W y un registro elegido por el programador. La operación es W OR F.

```
IORWF    PORTC,F
```

Si antes de la instrucción W vale B'01111111' y PORC vale B'00001111' después de la instrucción PORTC vale B'01111111' y W vale B'01111111'. Nótese que podemos elegir el destino y la otra opción, como ya se dieron cuenta por las instrucciones pasadas, puede ser W.

Afecta a:
Z se pone a 1 si el resultado es 0

MOVLW

Carga al registro W con un literal elegido por el programador para luego hacer una operación matemática o moverlo a otro registro como veremos más adelante. Sin duda alguna, una de las instrucciones más usadas en la programación ASM.

```
MOVLW    .255
```

Si antes de la instrucción W vale .15, después de la instrucción W vale .255.

No afecta a ningún bit del registro STATUS.

MOVF

Mueve el contenido de un registro a otro registro elegido por el usuario.

```
MOVF     RETARDO,W
```

Si antes de la instrucción W vale H'2A' y RETARDO vale H'FF', después de la instrucción W vale H'FF'. Nótese que aquí podemos elegir el destino, y tenemos la posibilidad de elegir el destino al propio registro RETARDO. Al principio parece innecesario, pero se puede tomar como una verificación, ya que se ve afectado el registro STATUS bit Z.

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0

Anteriormente, habíamos dicho que esta instrucción se la puede tomar como verificación para saber si se guardó con el mismo valor que tenía, el bit Z se pone a 1 si el valor es igual al que tenía cargado.

MOVWF

Mueve el contenido del registro W a un registro cualquiera elegido por el programador. Sin duda alguna, esta instrucción, es otra muy usada en la programación ASM

```
MOVWF    ADCON0
```

Si antes de la instrucción W vale B'10000001' y ADCON0 vale 0x0, después de la instrucción ADCON0 vale 0x81.

No afecta a ningún bit del registro STATUS.

NOP

No realiza ninguna operación. Solo consume un ciclo de instrucción.

```
NOP
```

No afecta a ningún bit del registro STATUS.

RETFIE

Carga al CP con el valor de la parte alta de la pila para volver al lugar dónde se encontraba el CP antes de atender la interrupción. Al mismo tiempo, pone a 1 el bit GIE para activar de nuevo las interrupciones.

```
RETFIE
```

No afecta a ningún bit del registro STATUS.

RETLW

Carga al CP con el valor de la parte alta de la pila para volver al lugar dónde se encontraba el CP desde dónde se llamó a la subrutina y al retornar, lo hace con un literal cargado en W especificado por el programador. Esta instrucción, se utilizan en las tablas (para más detalle, ver la explicación del GOTO y CALL).

```
RETLW    'L'
```

En este ejemplo, el MPLAB, carga en W el código ASCII correspondiente a la letra L

Extendiendo el ejemplo:

```
   PAGESELW   TABLA          ;CONFIGURA AL PCLATH PARA VOLVER AL LUGAR CORRECTO
   MOVFW      contador       ;CARGA A W LA POCICIÓN A LEER EN LA TABLA POR EJEMPLO 3
   CALL       TABLA          ;LLAMA A LA RUTINA TABLA
   CALL       LCD_DATO       ;LLAMA A LA RUTINA PARA MOSTRA AL LCD
   NOP
;
;
TABLA  ADDWF    PCL,F         ;SUMA AL PCL CON EL CONTENIDO DE W POR EJEMPLO 3
       RETLW      '0'
       RETLW      '1'        ;RETORNA CON 1 ASCII
       RETLW      '2'        ;RETORNA CON 2 ASCII
       RETLW      '3'        ;RETORNA CON 3 ASCII
       RETLW      'T'        ;RETORNA CON T ASCII
```

Este es un ejemplo sencillo de como utilizar RETLW. Para interpretar este código empezamos desde PAGESELW, supongamos que el CP está en esta instrucción (que está explicado que hace) luego pasa a la instrucción MOVFW contador y suponemos que tiene cargado 3 en decimal, por lo que W pasará a tener 3 en decimal. El CP continua con CALL TABLA, el CP saltará por encima a todas las demás instrucciones y se dirige a la etiqueta TABLA y ejecuta la instrucción ADDWF PCL,F En el código hablamos que le suma 3 al PCL, por lo que saltará al RETLW '3' cargando a W con el código ASCII 3. Retorna justo debajo del CALL TABLA, o sea retorna a CALL LCD_DATO y ejecuta la rutina correspondiente, cuando termina, regresa al NOP (que puede ser cualquier instrucción que necesite el programador.

Si en cambio, contador hubiera tenido cargado 4 en decimal cuando llegue a la tabla y le sume al PCL este apuntará a RETLW 'T' cargando en W el código correspondiente ASCII.

No afecta a ningún bit del registro STATUS.

RETURN

Carga al CP con el valor de la parte alta de la pila para volver al lugar dónde se encontraba el CP cuando se llamó a la rutina o subrutina.

La diferencia con RETLW es que RETURN regresa sin cambiar a W. Este se utiliza para terminar una rutina y no se necesite ningún

dato. Por ejemplo en la rutina CALL LCD_DATO no nos sirve que vuelva con ningún valor ya que es una rutina para enviar datos a un LCD, así que esta rutina tendrá implementada RETURN

```
RETURN
```

No afecta a ningún bit del registro STATUS

RLF

Rota hacia la izquierda los bit de un registro seleccionado por el programador. El destino de la operación se puede elegir. Cada rotación equivale a multiplicar por 2 si el bit C del registro STATUS es 0.

```
RLF     PORTC,F
```

Si antes de la instrucción PORTC vale B'00001000', después de la instrucción vale B'00010000'. Si se hubiera elegido como destino W, PORTC después de la instrucción continua valiendo B'00001000' y W vale B'00010000'

Afecta a:
C se pone a 1 si hubo acarreo

RRF

Rota hacia la derecha los bits de un registro seleccionado por el programador. El destino de la operación se puede elegir. Cada rotación equivale a dividir por 2 si el bit C del registro STATUS es 0.

```
RRF     PORTB,F
```

Si antes de la instrucción PORTB vale B'10000000' después de la instrucción PORTB vale B'0100000'. Si se hubiera elegido como destino W, PORTB después de la instrucción continua valiendo B'10000000' y W vale B'01000000'

Afecta a:
C se pone a 1 si hubo acarreo

Extendiendo la explicación de las instrucciones RRF y RLF:

A la hora de utilizar estas dos instrucciones, hay que prestarle atención al bit C del registro STATUS. La razón de esto, es porque la rotación se hace a través del bit C.

Supongamos que tenemos lo siguiente:

```
BIT C = 0
TEMP = B'00010000'
```

Ejecutamos la instrucción RRF y TEMP vale B'00001000'. O si ejecutamos la instrucción RLF TEMP vale B'00100000'

Pero si ahora tenemos:

```
BIT C = 1
TEMP = B'00010000'
```

Ejecutamos la instrucción RRF y TEMP vale B'10001000'. O si ejecutamos la instrucción RLF TEMP vale B'00100001'

Vemos como rotan los bit dependiendo del valor del bit C. Pero anteriormente, habíamos dicho que estas dos instrucciones afectan al bit C. La actualización del bit C, lo hace después de la rotación. Lo vemos con un ejemplo:

```
MOVLW B'10001001'
MOVWF temp
BCF     STATUS,C ;PONEMOS A 0 AL BIT C
RLF     temp,F   ;ROTAMOS A LA IZQUIERDA
```

Al ejecutar este programa, nos dará lo siguientes resultados:

```
TEMP = B'00010010'
BIT C = 1
```

Y para ver la diferencia vemos lo siguiente:

```
MOVLW B'00000001'
MOVWF temp
BCF     STATUS,C ;PONEMOS A 0 AL BIT C
RLF     temp,F   ;ROTAMOS A LA IZQUIERDA
```

Al ejecutar este programa, nos dará lo siguientes resultados:

```
TEMP = B'00000010'
BIT C = 0
```

Algo que me había olvidado de mencionar pero que MIGSANTIAGO del foro de TODOPIC estuvo atento es que estas dos instrucciones, nos sirve para enviar datos en forma serial utilizando el bit C que lo veremos más adelante.

Recordemos que, para utilizar estas instrucciones para multiplicar o dividir, debemos asegurarnos de que el bit C, esté en 0.

SLEEP

Pone al microcontrolador en bajo consumo.

```
SLEEP
```

Afecta a:
TD se pone a 1
PD se pone a 1

SUBLW

Resta el contenido de W con un literal de hasta 8 bit (.255). El resultado se guarda en W.

```
SUBLW    .20
```

Si antes de la instrucción W vale .23 después de la instrucción W vale .3 Para saber si el resultado es negativo o positivo, hay que chequear el bit C del registro Status. Si hay acarreo, el resultado es negativo, y por el contrario, si no hay acarreo es positivo.

Afecta a:
Z se pone a 1 si el resultado es 0
DC se pone a 0 si hay acarreo del bit del 4 al 5 bit del registro (recordemos que en la resta, es distinto a la suma, por eso, se pone a 0 si hubo acarreo).
C se pone a 0 si hubo acarreo (recordemos que en la resta, es distinto a la suma, por eso, se pone a 0 si hubo acarreo).

SUBWF

Resta el contenido de un registro seleccionado por el programador con el contenido del registro W. La fórmula es F - W = d. d es la dirección elegida por el programador en dónde se guardará el resultado que puede ser el registro W o el registro elegido por el programador.

```
SUBWF    MINUENDO,W
```

Si antes de la instrucción W vale .55 y MINUENDO vale .56, después de la instrucción, MINUENDO vale .56 y W vale .1

Afecta a:
Z se pone a 1 si el resultado es 0
DC se pone a 0 si hubo un acarreo del 4 bit al 5 bit (recordemos que en la resta, es distinto a la suma, por eso, se pone a 0 si hubo acarreo).
C se pone a 0 si hubo acarreo del 7 bit. (recordemos que en la resta, es distinto a la suma, por eso, se pone a 0 si hubo acarreo).

SWAPF

Cambia los Nibbles de un mismo registro elegido por el programador. Los 4 bit de menor peso, pasan a ser lo 4 bits de mayor peso, y los 4 bits de mayor peso, pasan a ser los 4 bits de menor peso. El destino puede ser seleccionado.

Cabe pensar que puede ser una instrucción de muy poco uso, pero todo lo contrario si se utilizan con las interrupciones. Microchips recomienda su utilización a la hora de salvar el contexto y restaurarlo en una interrupción ya que no modifica el registro STATUS. Cuando trabajemos con las interrupciones, se verá que es muy recomendable salvar el registro STATUS y W en la RAM para luego restaurarlos. Si utilizamos la instrucción MOVF, es afectado el bit Z, perdiendo su estado original en el momento de la interrupción. Esto se soluciona, utilizando la instrucción SWAPF.

No se preocupe si no lo entiende por ahora. Lo entenderá cuando veamos ejemplos de interrupciones.

```
SWAPF   STATUS,W
```

Si antes de la instrucción W vale H'55' y el registro STATUS vale B'00100100', después de la instrucción el registro STATUS vale H'24' y el registro W vale B'01000010'

No afecta a ningún bit del registro STATUS.

XORLW

Realiza la operación lógica XOR entre un literal o valor y el registro W. El resultado queda guardado en el registro W.

```
XORLW   B'11000101'
```

Si antes de la instrucción W vale B'11111000', después de la instrucción W vale B'00111101'.

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0

XORWF

Realiza la operación XOR entre un registro elegido por el programador y el registro W. La operación es F XOR W = d. El resultado se puede elegir dónde será guardado.

```
XORWF   PORTB,F
```

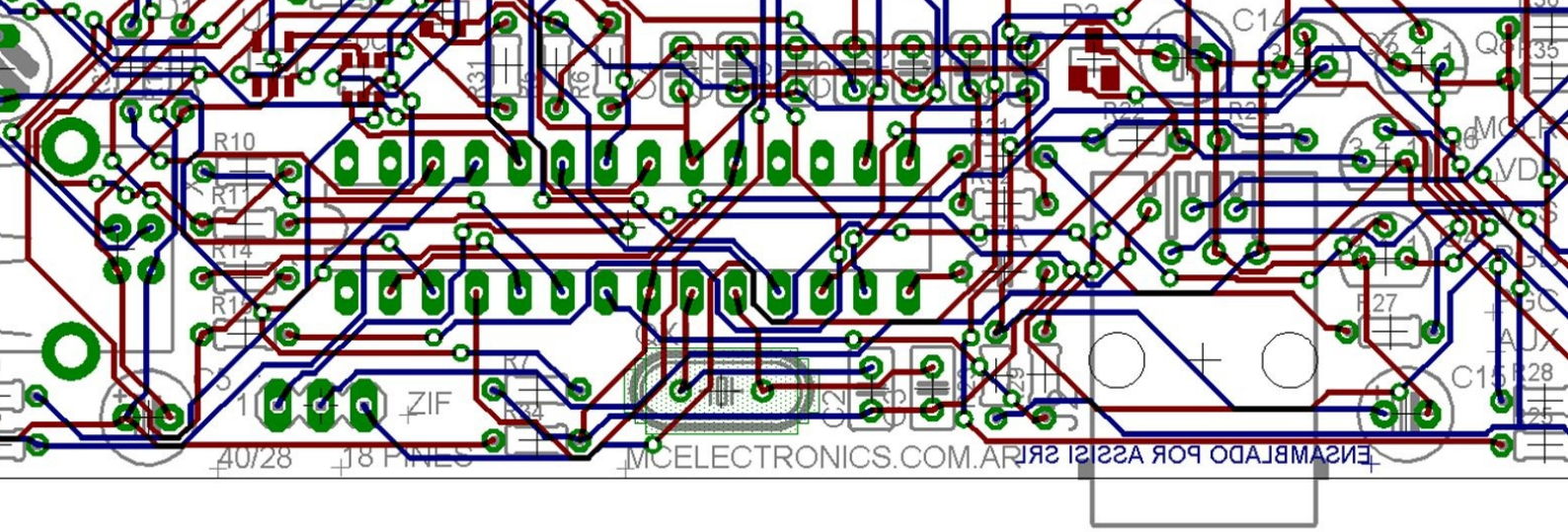
Si antes de la instrucción PORTB vale B11111110' y W vale B'00000001', después de la instrucción W vale .1 y PORTB vale B'11111111'.

Afecta a:
Z Se pone a 1 si el resultado de la operación es 0

Repasando...

ADDLW k - Add Literal and W
(W) + k -> (W)
ADDWF f,d - Add W and f
(W) + (f) -> (d)
ANDLW k - AND Literal with W
(W) AND (k) -> (W)
ANDWF f,d - AND W with f
(W) AND (f) -> (d)
BCF f,b - Bit Clear f
0 -> (f<b>)
BSF f,b - Bit Set f
1 -> (f<b>)
BTFSS f,b - Bit Test f, Skip if Set
Pasa por alto si (f<b>) = 1
BTFSC f,b - Bit Test f, Skip if Clear
Pasa por alto si (f<b>) = 0
CALL k - Call Subroutine
Llama a Subrutina (k)
CLRF f - Clear f
00h -> (f)
CLRW - Clear W
00h -> (W)
CLRWDT - Clear Watchdog Timer
00h -> WDT
COMF f,d - Complement f
(f(neg)) -> (d)
DECF f,d - Decrement f
(f) - 1 -> (d)
DECFSZ f,d - Decrement f, Skip if 0
(f) - 1 -> (d). Pasa por alto si resultado = 0
GOTO k - Unconditional Branch
Salto a etiqueta (k)
INCF f,d - Increment f
(f) + 1 -> (d)
INCFSZ f,d - Increment f, Skip if 0
(f) + 1 -> (d). Pasa por alto si resultado = 0
IORLW k - Inclusive OR Literal with W
(W) OR k -> (W)
IORWF f,d - Inclusive OR W with f
(W) OR (f) -> (d)
MOVF f,d - Move f
(f) -> (d)
MOVLW k - Move Literal to W
k -> (W)
MOVWF f - Move W to f
(W) -> (f)
NOP - No Operation
RETFIE - Return from Interrupt
TOS -> PC, 1 -> GIE
RETLW k - Return with Literal in W
k -> (W); TOS -> PC
RLF f,d - Rotate Left f through Carry
Contenido registro f rota 1 bit a la izquierda
RETURN - Return from Subroutine
Regresa de Subrutina
RRF f,d - Rotate Right f through Carry
Contenido registro f rota 1 bit a la derecha
SLEEP - SLEEP Mode
SUBLW k - Subtract W from Literal
k - (W) -> (W)
SUBWF f,d - Subtract W from f
(f) - (W) -> (d)
SWAPF f,d - Swap Nibbles in f
(f<3:0>) -> (d<7:4>), (f<7:4>) -> (d<3:0>)
XORLW k - Exclusive OR Literal with W
(W) XOR k -> (W)
XORWF f,d - Exclusive OR W with f
(W) XOR (f) -> (d)

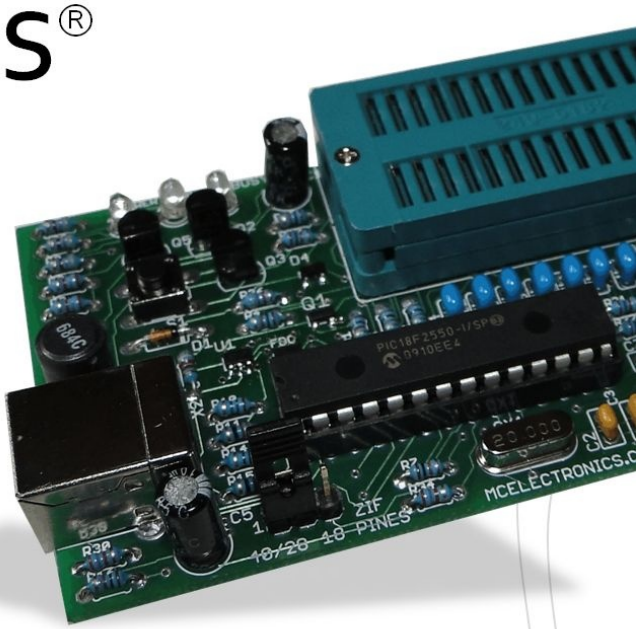
k: Campo Literal, constante o etiqueta. **f:** Dirección Registro. **b:** Dirección de Bit en un registro de 8 bits.
W: Registro de Trabajo. **d:** Selección de Destino. 0 almacena resultado en W; 1 almacena resultado en f. Por defecto d=1.



mc electronics®

Programador y Debugger Express

USB para PIC y dsPIC



El MCE PDX USB se puede utilizar como Programadador y Debugger Express. Además es un Analizador lógico de 3 canales. Incluye zócalo ZIF, conector RJ11 y EasyJack de 6 pines compatible con PICKit2 de Microchip.

$ 149,00 +IVA

USD 38,90

Incluye envío sin cargo a todo el país !

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

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

PIN 796948766332890345678046587411

Generar señal VGA con un PIC

Casi todos hemos intentando alguna vez utilizar un microcontrolador para generar una señal de vídeo compatible con un monitor VGA, utilizados por la mayoría de los ordenadores. Se trata de una buena idea, que puede dotar a nuestros proyectos de una pantalla “de verdad”, barata, grande, brillante y en colores. Sin embargo, no se trata en absoluto de una tarea sencilla: los breves tiempos implicados en la generación de cada píxel hace que sea un objetivo muy difícil de lograr. A lo largo de este artículo intentaremos proporcionarte información útil para que puedas encarar con éxito ese desafío.

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

Cuando un producto de cualquier tipo se fabrica en enormes cantidades, su costo disminuye notablemente. Ese es, sin dudas, el caso de los monitores utilizados en los ordenadores personales. Los monitores VGA, sobre todo aquellos más viejos que utilizan como elemento de imagen un tubo de rayos catódicos, pueden conseguirse por muy poco dinero, e incluso, si buscamos un poco, gratis. Ante este panorama surge inevitablemente la idea de “usarlos para algo”, y como nuestra pasión es la electrónica, lo más natural del mundo es intentar convertirlos en parte de nuestros proyectos. Si lográsemos generar las señales adecuadas, podríamos utilizar estas pantallas como “display” en cualquiera de nuestros proyectos. Desafortunadamente, esto no es algo sencillo. Basta con buscar un poco por la red para darse cuenta de que solo hay un puñado de proyectos destinados a generar imágenes de vídeo compatibles con monitores VGA, y la mayoría de ellos simplemente generan barras de color, o imágenes en blanco y negro de muy baja resolución. Sin embargo, es posible crear imágenes y texto en colores, con pocos componentes, un microcontrolador y el programa adecuado.

A lo largo de este articulo vamos a intentar proporcionar las bases necesarias para que puedas desarrollar un dispositivo pequeño y barato, capaz de recibir ordenes a través de un puerto SPI y desplegar texto -en colores- sobre un monitor VGA. Uno de los mejores trabajos que se pueden consultar sobre como generar estas señales -y el resto de los parámetros de la imagen- con un microcontrolador PIC18 es el de Victor Timofeev (*http://www.pic24.ru/*), en el que nos hemos basado para escribir esto y cuya lectura recomendamos fuertemente. El sitio de Victor está en ruso (algunas partes en inglés), pero es posible lograr una traducción bastante decente utilizando Google Translator.

Figura 1: Podemos utilizar un monitor VGA como parte de nuestros proyectos.

La imagen VGA

Lo primero que necesitamos conocer -y al detalle- es la forma en que uno de estos monitores genera su imagen. No debemos olvidar que no intentamos utilizar una placa VGA como la que poseen los ordenadores para generar la imagen, sino que nuestro proyecto debe encargarse de proporcionar al monitor las señales adecuadas para que este muestre lo que queremos. Esto significa que conceptos como “segmento de memoria de vídeo” o “modo 13” -propios de los adaptadores VGA mencionados- no tienen ninguna utilidad para nosotros. Necesitamos enviar al monitor, en el momento exacto, la información necesaria para que este pueda “dibujar” cada píxel que conforma la imagen.

Un monitor VGA común puede ser controlado utilizando solo 5 lineas: dos de sincronismo (horizontal y vertical) y tres señales analógicas (rojo, verde y azul). Las señales de sincronismo, como su nombre lo indica, sirven para que el monitor “sepa” cuando se encuentra en el comienzo de un cuadro o de una linea. Las señales analógicas correspondientes a cada color permiten variar la intensidad del los mismos simplemente variando su voltaje. Al menos en teoría, combinando correctamente el valor de tensión aplicado a las tres lineas analógicas se puede obtener cualquier color que deseemos. En la práctica, la velocidad de nuestro microcontrolador y las características de nuestro conversor digital-analógico determinarán la cantidad de colores que podemos mostrar. La imagen se dibuja linea a linea, de arriba hacia abajo. Cada linea comienza a la izquierda y termina a la derecha de la pantalla. Cuando una linea se ha completado, debemos enviar un pulso de sincronismo horizontal para que la electrónica del monitor la haga avanzar a la posición siguiente. Cuando todas las lineas que conforman una imagen han sido dibujadas, enviamos un pulso de sincronismo vertical para indicarle al monitor que la imagen (el “cuadro”) está completo y que la próxima linea a dibujar debe estar, nuevamente, en la parte superior de la pantalla. Todo esto se repite decenas de veces por segundo.

La “resolución” de nuestra imagen se mide en píxeles. Decir que tenemos una resolución de “640x480” significa que la imagen se compone de 640 píxeles a lo ancho -o 640 píxeles en cada línea- y 480 lineas por cuadro. Nuestro programa debe ser capaz de enviar al monitor la tensión adecuada en cada una de las lineas analógicas para que cada uno de los píxeles que la componentes tenga el color deseado. Veamos un ejemplo concreto: si queremos mostrar 60 cuadros por segundo, en el modo “VGA Standar” de 525 lineas (de las cuales “vemos” solo 480), disponemos de solo

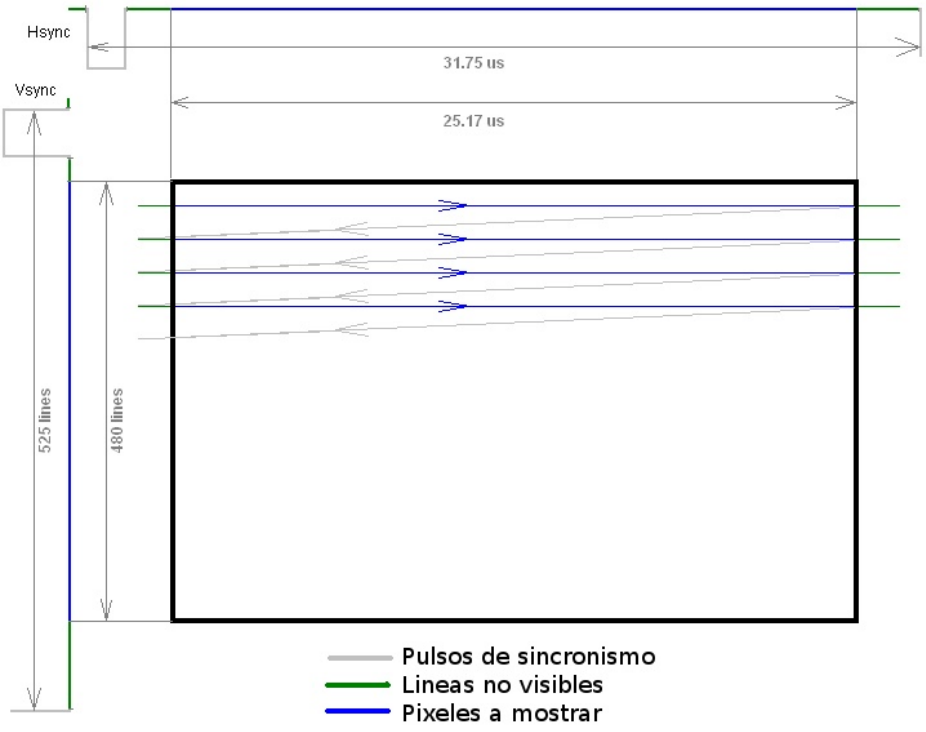


Figura 2: Diagramas de tiempos.

Tiempo por linea = 1 segundo / 60 cuadros / 525 lineas = 31,75 us.

31,75 millonésimas de segundo para dibujar cada linea. Por razones derivadas del diseño, el trazo de la linea solo es

visible durante 25.17 us. (ver figura 2). En ese lapso de tiempo tenemos que obtener los datos correspondientes a cada píxel y colocar el valor correspondiente a cada linea analógica (R, G, B), una vez para cada píxel. Si pensamos dibujar 640 píxeles, solo tenemos

Tiempo por píxel = 25,17 us / 640 = 0.039328125us (39,33 ns)

El primer problema que enfrentamos es la generación de los pulsos de sincronismo. Necesitamos un pulso -bajo- de una duración de 3,9 us cada 31,75us., que será nuestro pulso de sincronismo horizontal. Y durante la generación de las últimas dos lineas de la imagen -lineas 524 y 525- necesitamos un pulso -también negativo- que sirva como sincronismo vertical. Victor recomienda utilizar una interrupción para generar el pulso de sincronismo horizontal. Para generar los valores adecuados de tensión en las lineas analógicas correspondientes a los colores rojo, verde y azul, puede utilizarse un esquema como el propuesto en pic24.ru:

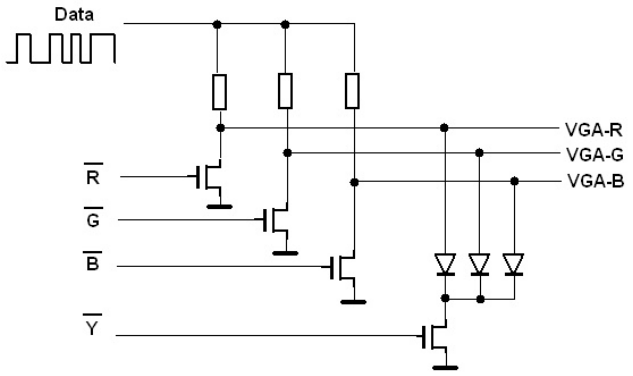


Figura 3: 4 pines del PIC bastan para generar el color de cada píxel.

Tres pines con salidas tipo “open drain” pueden “encender” o “apagar” las lineas correspondientes al rojo, verde y azul. Son 8 combinaciones que proporcionan 8 colores diferentes. El cuarto pin en cuestión (Y) proporciona un rudimentario mecanismo de control de brillo. Cuando esa salida está en nivel bajo, el transistor se cierra y los voltajes presentes en los nodos etiquetados como VGA-R, VGA-G y VGA-B se modifiquen a través del divisor formado por los resistores y diodos. Victor sugiere utilizar diodos 1N5819 -funcionan perfectamente, como puedes ver en las fotos- pero en las pruebas que hemos realizado pudimos comprobar que incluso un común y modesto 1N4148 sirve. Esto nos permite duplicar la cantidad de colores, ya que disponemos de 8 colores “vivos” y 8 colores “lavados”. En la practica -como se ve en la figura 4- son solo 15, ya que dos de ellos corresponden al negro.

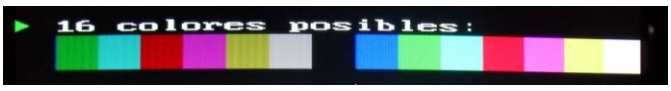


Figura 4: Tenemos 15 colores diferentes, sin necesidad de conversores DA externos.

El pin “data”, en la parte superior de la figura 3, se encarga de encender o apagar cada píxel. Resumiendo, para generar un píxel necesitamos ajustar el valor de los pines R, G, B e Y, y poner el pin “data” en el estado correcto. Si vamos a mostrar solo texto, no necesitamos cambiar el color de cada uno de los píxeles que componen la linea, sino que basta con hacer el cambio cada vez que el trazo del haz de electrones pasa de un carácter a otro. Esto significa que disponemos de un poco más de tiempo para controlar la señal “data”, que es la que en definitiva va a determinar nuestra resolución horizontal. Victor, en el trabajo mencionado, explica todo esto en detalle y proporciona un programa completo (código fuente incluido) que, junto al circuito de la figura 5, permite escribir 30 filas de 30 columnas de texto, con 16 colores posibles para cada carácter, sobre fondo negro, en un monitor VGA estándar.

Los caracteres a mostrar se envían al dispositivo mediante el bus SPI. La velocidad máxima de transmisión de datos es de 250Kbps, y los comandos que acepta se muestran en la tabla 1.

Todo esto está implementado en una librería que puede descargarse junto al resto del código fuente, junto a una explicación más detallada de cada comando, desde *http://www.pic24.ru/lib/exe/fetch.php/osa/articles/terminal.rar*

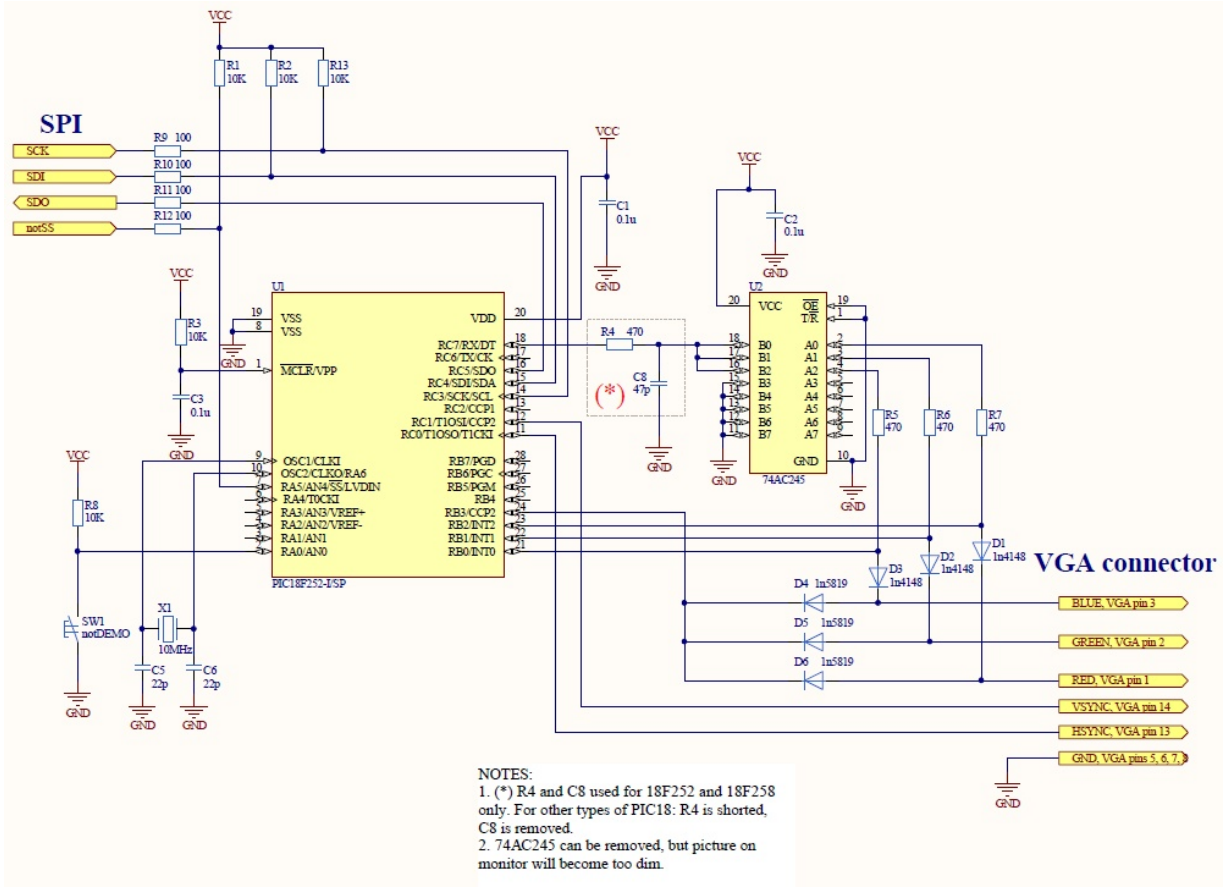
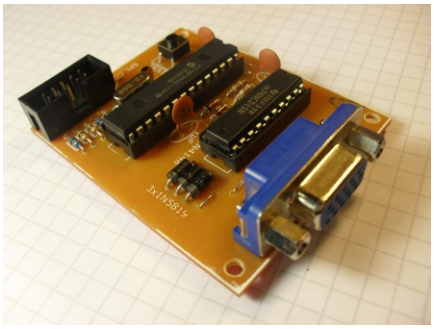
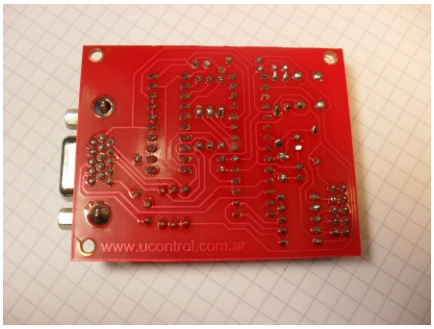
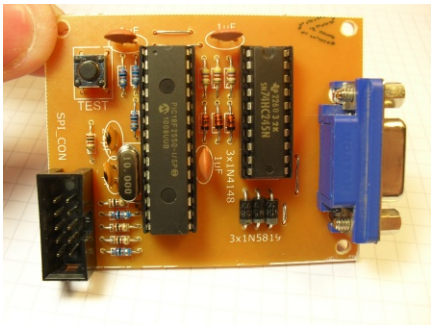


Figura 5: Circuito propuesto por Victor Timofeev.

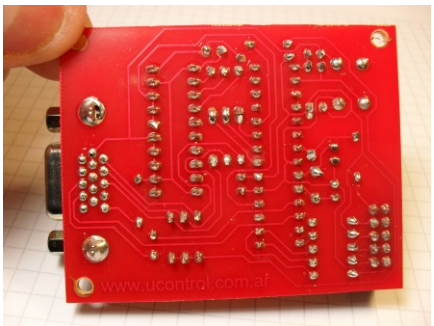






Cod	Nombre	Opción	Descripción
00	SPI_CMD_NOP		No hace nada.
01	SPI_CMD_SET_X	X	Fija la posición horizontal del cursor (0..X_SIZE-1)
02	SPI_CMD_SET_X	Y	Fija la posición vertical del cursor (0..Y_SIZE-1)
03	SPI_CMD_SET_CURSOR	S	Fija el tamaño del cursor (0..15)
04	SPI_CMD_SET_COLOR	C	Fija el color actual (0..15)
05	SPI_CMD_SET_SYMBOL_COLOR	C	Fija el color de la posición actual (0..15)
06	SPI_CMD_CLRSCR		Limpia pantalla, pone color 15 y cursor en 0,0
07	SPI_CMD_CHECK	state	Pide el estado del comando SPI_CMD_CLRSCR
08	SPI_CMD_GET_SYMBOL	char	Devuelve el carácter en la posición del cursor
09	SPI_CMD_GET_X X	X	Devuelve el valor de la columna en que está el cursor
0A	SPI_CMD_GET_Y Y	Y	Devuelve el valor de la fila en que está el cursor
0B	SPI_CMD_GET_CURSOR	S	Devuelve el tamaño del cursor
0C	SPI_CMD_GET_COLOR	C	Devuelve el color del cursor
0D	SPI_CMD_GET_CURSOR	C	Devuelve color de la celda actual
0E	SPI_CMD_GET_X_SIZE	X_SIZE	Devuelve el tamaño (horizontal) de la pantalla
0F	SPI_CMD_GET_Y_SIZE	Y_SIZE	Devuelve el tamaño (vertical) de la pantalla

Tabla 1: Comandos aceptados.



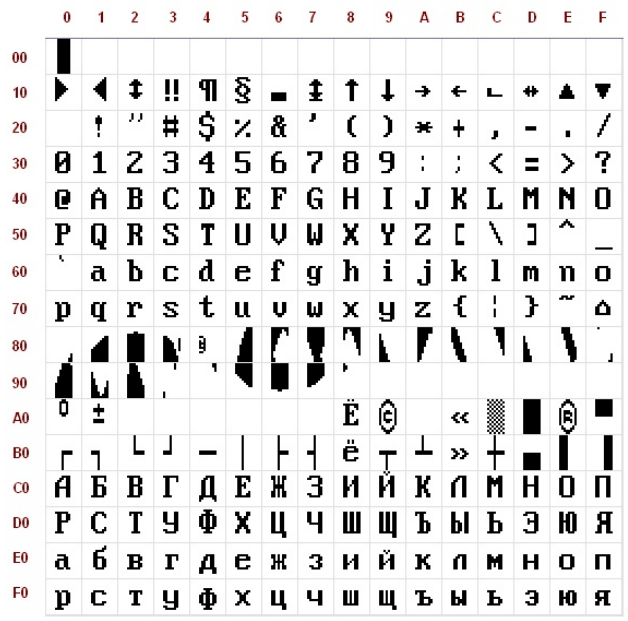


Figura 6: Este es el juego de caracteres incluidos. Por supuesto, puede modificarse.

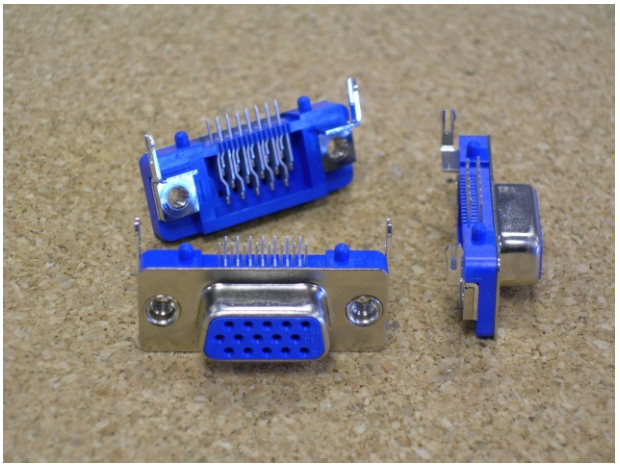
La figura 6 muestra los 256 caracteres disponibles. La definición de estos caracteres se encuentra en el archivo font.asm y puede ser modificado para incluir caracteres no contemplados. Las fotografías que ilustran esta nota permiten apreciar la calidad y definición de la imagen obtenida con esta interfaz. Con poco trabajo puede construirse un dispositivo pequeño y flexible que permitirá a nuestros proyectos mostrar texto en cualquier monitor. ¿Te animas a construirla?

El conector VGA

Un conector "HD-15" -a veces también llamado "RGBHV" o "HD-15"- como el que poseen la mayoría de los monitores VGA posee 15 pines dispuestos en tres hileras de 5 cada una. Es posible que aún haya en funcionamiento algún viejo monitor dotado de una versión anterior del conector actual, con solo 9 pines (llamado "DE-9"), pero el 99% de los monitores VGA que encuentres por ahí tendrán el conector azul de 15 pines que ves en la fotografía.

Las letras "HD" hacen referencia a "High Density" ("Alta densidad"), para diferenciarlos de los conectores que tienen el mismo factor de forma, pero sólo 2 filas de pines. Este conector permite transportar las cinco señales básicas que se necesitan para obtener una imagen vertical: los componentes analógicos rojo, verde y azul, y las señales de sincronismo vertical y horizontal. El cuadro siguiente muestra la función de cada uno de los pines de este conector:

Pin 1	RED	Red video
Pin 2	GREEN	Green video
Pin 3	BLUE	Blue video
Pin 4	N/C	Not connected
Pin 5	GND	Ground (HSync)
Pin 6	RED_RTN	Red return
Pin 7	GREEN_RTN	Green return
Pin 8	BLUE_RTN	Blue return
Pin 9	+5 V	+5 V (DDC)
Pin 10	GND	Ground (VSync, DDC)
Pin 11	N/C	Not connected
Pin 12	SDA	I²C data
Pin 13	HSync	Horizontal sync
Pin 14	VSync	Vertical sync
Pin 15	SCL	I²C clock




µCONTROL
Electrónica en General Pics en Particular