Aprendiendo a programar Microcontroladores PIC en Lenguaje C con CCS

www.edudevices.com.ar

Por Andrés Raúl Bruno Saravia

Entrega Nº 8.

A partir de aquí comenzaremos a usar lo aprendido en los capítulos anteriores. La única forma de asimilar e incorporar nuevos conocimientos, es poner en práctica lo que se aprende. Iremos desde lo simple a lo complejo, desde el encendido de un LED hasta el control por PC de nuestra aplicación o circuito electrónico.

Para trabajar podemos usar un protoboard, sin embargo no soy muy amigo de este tipo de herramientas ya que es fácil encontrarse con falsos contactos y crear verdaderas marañas de cables. Es preferible construir un circuito elemental que tenga todos los periféricos que veremos o en su defecto adquirir ya la placa armada. Si bien el adquirir algo armado nos quita el sabor del "hágalo usted mismo", nos permite ganar tiempo. Otra herramienta que se puede usar es el PROTEUS, sin embargo la virtualización de un circuito no nos permite afrontar las problemáticas reales, como son los rebotes, ruido electromagnético, etc.

Yo presentaré el esquema de una placa ideal y aconsejaré la compra de un modelo, sin embargo queda en el lector el tomar estas ideas u optar por las opciones anteriores que también son válidas, lo importante es aprender.

Características de los Microcontroladores PIC Línea Media Mejorada PIC16F1XXX

Microchip es una compañía en constante evolución y renovación, y por ende año tras año presenta nuevos microcontroladores al mercado y nuevas mejoras en sus arquitecturas típicas. Hasta el año 2010, los núcleos de los procesadores de Microchip podían dividirse en 6 familias:

- **PIC Linea Base (PIC10F,PIC12F5X y PIC16F5X)**
- **PIC Línea Media (PIC16F y PIC12F)**
- **PIC18F**
- **dsPIC30F/33F**
- **PIC24F/24H**
- **PIC32**

De todas estas familias, las más antiguas son los PIC Línea Base y PIC Línea Media, concebidos para ser programados en Lenguaje Assembler, con lo cual las limitaciones de su arquitectura, también limitan la cantidad de código que se puede escribir en Lenguaje C dentro del microcontrolador. **Es por ello que a partir de la creación de la familia PIC18F y desde allí en adelante, todos los microcontroladores se concibieron para trabajar en Lenguaje C.**

Sin embargo el mercado de los microcontroladores PIC Línea Media es muy amplio y para estos usuario Microchip mejoró el núcleo de estos MCU, desarrollando una nueva generación , a la cual ha bautizado como PIC Línea Media Mejorada, los cuales se identifican por su código inicial: PIC16F1xxx o PIC12F1xxx.
Esta renovación de los viejos núcleos PIC16F introduce todas las mejoras que han adquirido los PIC18F con el Tiempo.

Dichas características son las siguientes:

- **Memoria de Programa hasta 32K instrucciones**
- **Memoria de Datos hasta 2K**
- **Mejora de Periféricos (USART, CCP, PWM , Puertos I/O)**
- **Inserción de nuevos Periféricos**
- **Modos de muy bajo consumo**
- **Mejoras en el Oscilador Interno con un PLL x 4**
- **Ampliación del Set de Instrucciones**
- **Ampliación del STACK a 16 niveles**
- **Salvado automático de contexto**
- **Control del STACK por medio del usuario**
- **Mejora del registro FSR y creación de 2 (FSR0 y FSR1)**

Todas estas nuevas características hacen de estos PIC los sucesores indiscutibles de la vieja familia PIC16F con la posibilidad de migración a PIC18F menos abrupta.
Para iniciar en nuestro curso, trabajaremos con el PIC16F1939, el cual reemplaza tácitamente a los viejos PIC16F877A y con menor costo.

El nuevo PIC16F1939

Este microcontrolador, forma parte de la familia **PIC16F193X**, la cual esta constituida por los siguientes dispositivos:

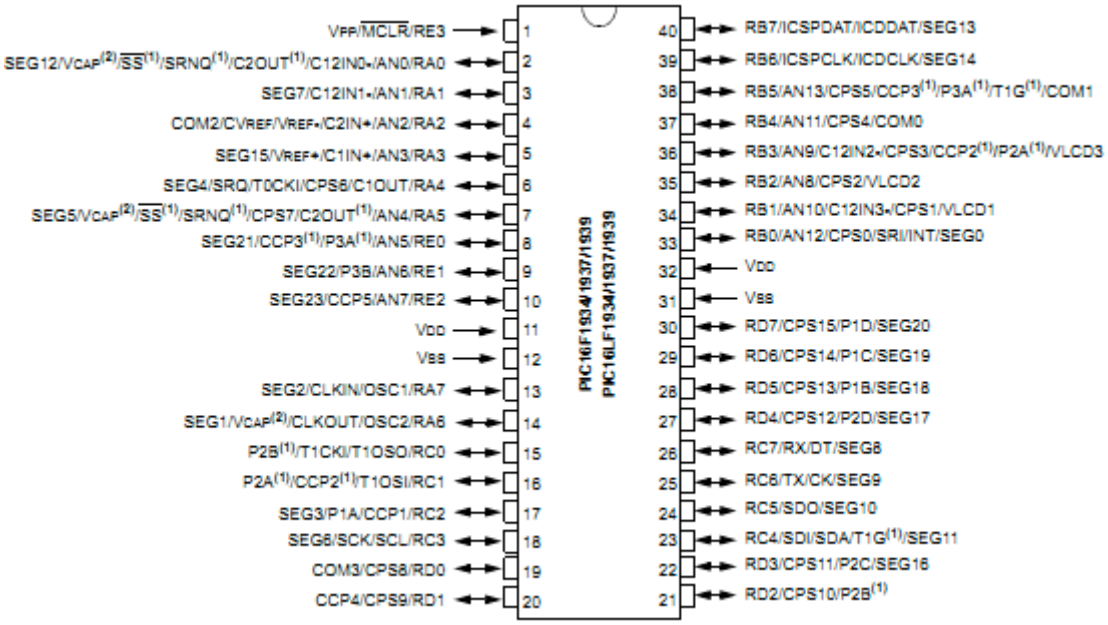
- PIC16F1933 (equivalente a un PIC16F873A)
- PIC16F1934(equivalente a un PIC16F874A)
- PIC16F1936(equivalente a un PIC16F876A)
- PIC16F1937(equivalente a un PIC16F877A)
- PIC16F1938(equivalente a un PIC16F876A pero con mas memoria)
- PIC16F1939(equivalente a un PIC16F877A pero con mas memoria)

El **PIC16F1939** es el más caro de la familia, sin embargo es más económico que su antecesor el PIC16F877A. Con esta política Microchip pretende seducir a sus consumidores para que abandonen sus aplicaciones emplazadas con las viejas unidades PIC16F y migren a los PIC16F1XXX .

Cuando migramos un desarrollo emplazado con PIC16F877A a un PIC16F1939 no solo ganamos en el costo sino que al mismo tiempo tenemos el doble de memoria de programa y mucha más memoria de datos, lo cual nos permitirá ampliar nuestra aplicación. En la siguiente figura vemos las características más destacables de estas unidades.

Device	Program Memory Flash (words)	Data EEPROM (bytes)	SRAM (bytes)	I/Os	10-bit A/D (ch)	Comparators	Timers 8/16-bit	EUSART	MI^{2}C/ SPI	ECCP Full Bridge	ECCP Half Bridge	CCP	LCD
PIC16F1933 PIC16LF1933	4096	256	256	25	11	2	4/1	Yes	Yes	1	2	2	16(1)/4
PIC16F1934 PIC16LF1934	4096	256	256	36	14	2	4/1	Yes	Yes	2	1	2	24/4
PIC16F1936 PIC16LF1936	8192	256	512	25	11	2	4/1	Yes	Yes	1	2	2	16(1)/4
PIC16F1937 PIC16LF1937	8192	256	512	36	14	2	4/1	Yes	Yes	2	1	2	24/4
PIC16F1938 PIC16LF1938	16384	256	1024	25	14	2	4/1	Yes	Yes	1	2	2	16(1)/4
PIC16F1939 PIC16LF1939	16384	256	1024	36	14	2	4/1	Yes	Yes	2	1	2	24/4

El PIN-OUT del PIC es el siguiente:

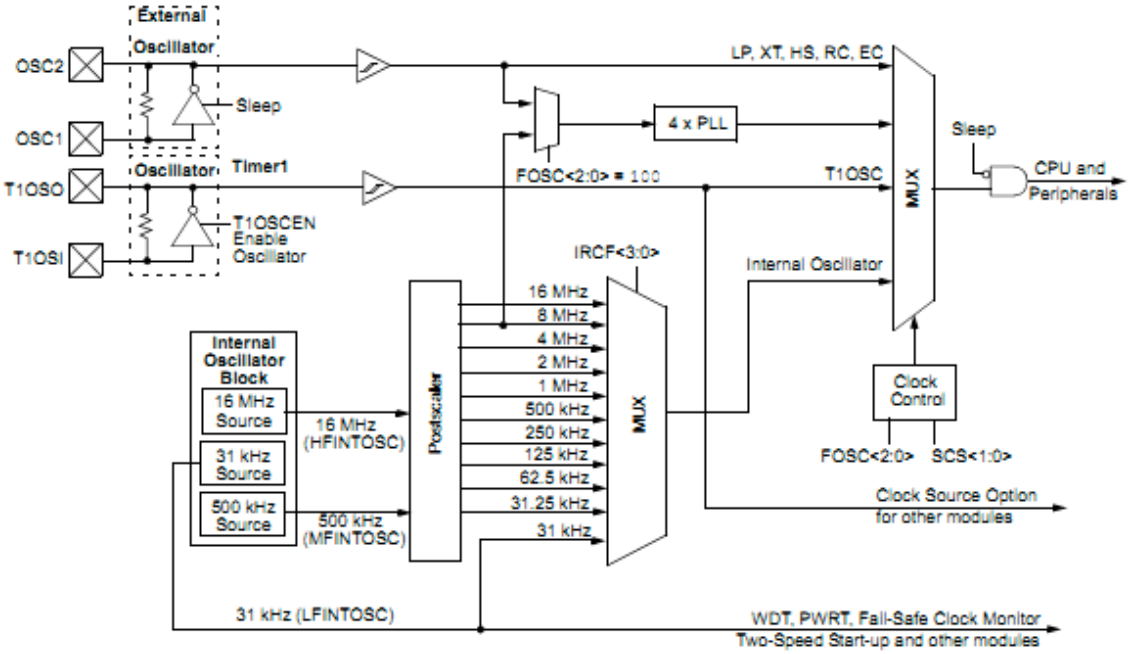


El ABC del Hardware elemental del PIC

Comencemos por decir que el Microcontrolador por ser una máquina secuencial, necesita para funcionar de un generador de pulsos de onda cuadrada, a los cuales se denominan pulsos de clock. Dichos pulsos pueden ser generados de 3 formas diferentes:

- Usando un Oscilador de Onda Cuadrada Externo (EC)
- Usando un resonador, Cristal de cuarzo o red RC colocada en OSC1 y OSC2
- Usando el Oscilador interno del Microcontrolador.

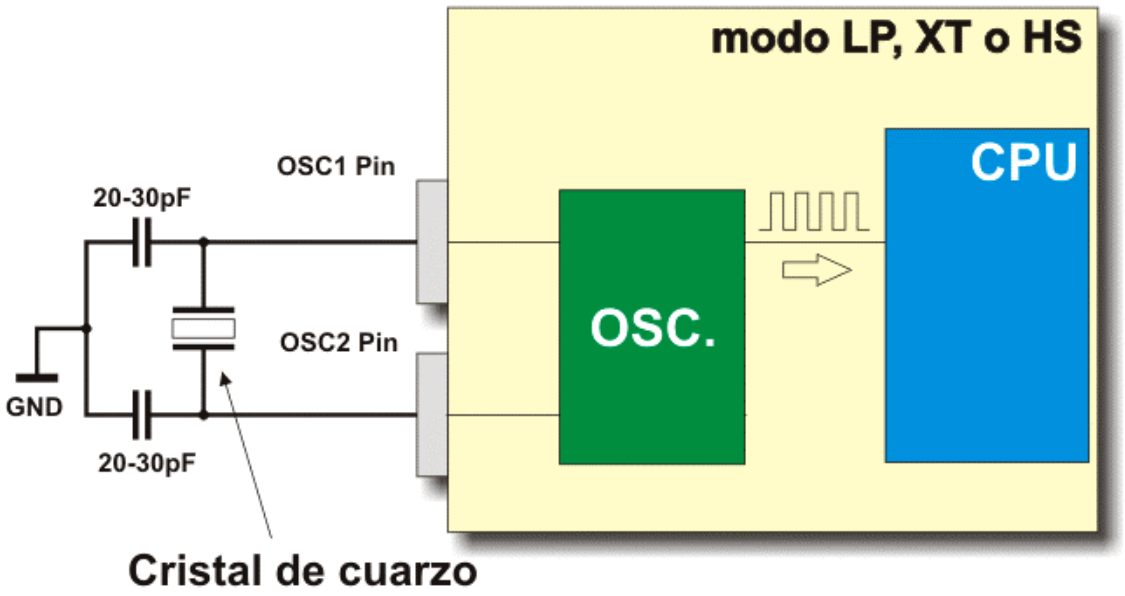
En los nuevos PIC16F1939, el módulo generador de clock presenta la siguiente arquitectura interna:



Las fuentes generadoras del clock para alimentar la CPU y los Periféricos pueden ser distintas:
Si usamos un cristal o un resonador externo, este lo podemos conectar entre los terminales OSC1 y OSC2. En caso de usar cristales, estos deben ser configurados en los bits de configuración como HS, XT o LP, según el rango de frecuencias donde trabajen. Para esto Microchip nos da una tablita de referencia:

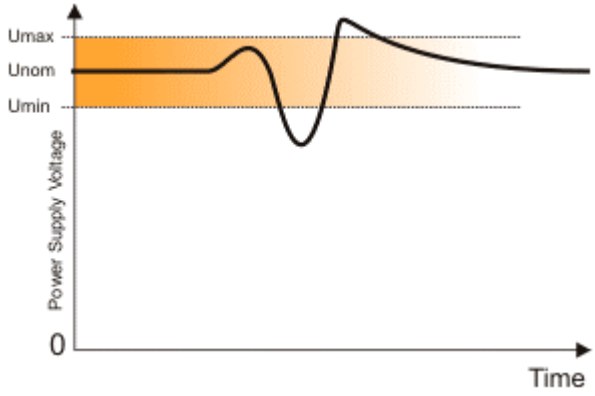
Osc Type	Crystal Freq.
LP	32 kHz
	200 kHz
XT	200 kHz
	1 MHz
	4 MHz
HS	4 MHz
	8 MHz
	20 MHz

En estos casos el hardware se realizará no solo conectando el respectivo cristal entre los terminales, sino también 2 capacitores del tipo cerámico, preferentemente NP0 respecto a masa, para estabilizar la frecuencia, y eliminar armónicos indeseados.



Los nuevos PIC incorporan un juego de 3 osciladores internos, denominados HFINTOSC de 16MHz, MFINTOSC de 500KHz y LFINTOSC de 31KHz. De estos por medio de un divisor de frecuencia programable, podemos obtener distintas frecuencias para excitar a la CPU y los periféricos. E incluso es posible derivar la salida del divisor de 8 MHz , para pasarla por el PLL y de esta forma tener 32 MHz de frecuencia de clock. En esta modalidad la CPU puede alcanzar su velocidad máxima de procesamiento para estos micros; 8 MIPS (8 Millones de Instrucciones Por Segundo).

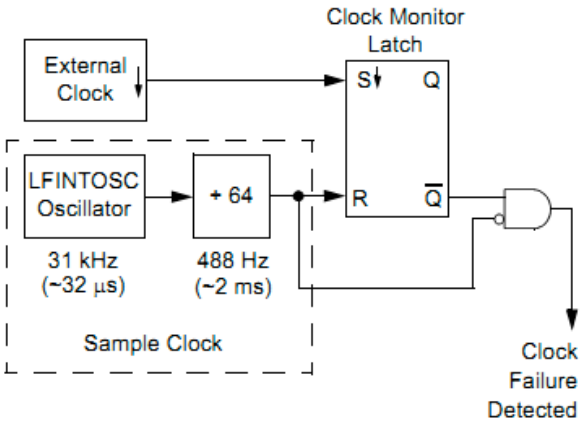
Esta opción del PLL no solo la activamos para el oscilador interno sino también para los cristales externos, sin embargo debe cuidarse de no superar los 32Mhz internos al CPU. Otra de las opciones internas que tiene este microcontrolador es la posibilidad de activar el BROWOUT RESET, el cual reseteará el microcontrolador en caso de detectar una variación de la tensión de alimentación del microcontrolador.



Además en los nuevos PIC es posible conmutar el oscilador en el momento de arranque, para ello existe un fusible de configuración mediante llamado IESO el cual habilita la conmutación o no.
La conmutación de la fuente del oscilador se usa por lo general cuando se quiere ahorrar energía, por ejemplo, se arranca en el modo de cristal externo de alta velocidad, y luego se pasa al oscilador interno de baja frecuencia.

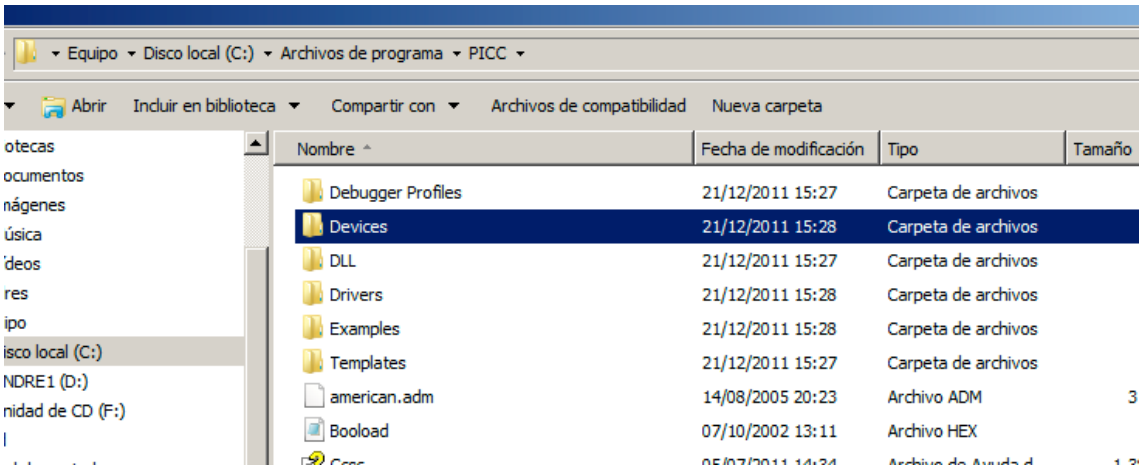
En este caso el sistema utiliza un bit denominado SCS (System Clock Select), el cual se encuentra dentro de un registro interno que controla al oscilador y que se denomina OSCCON. Este bit permite realizar la conmutación por software desde el oscilador externo al interno y viceversa.

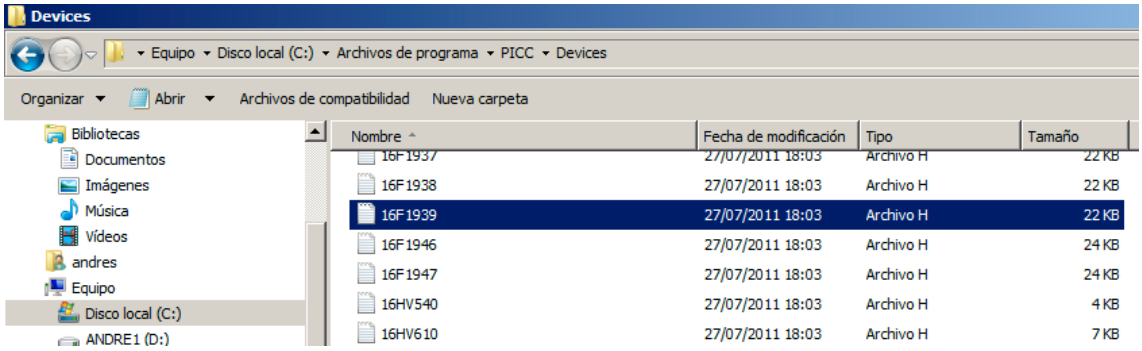
Por lo general, excepto que se este diseñando una aplicación donde se debe ahorrar al máximo los recursos de energía, la opción del IESO no se utiliza.
Otro recurso novedoso es el detector de la falla del oscilador primario o externo FSCM (Fail-Safe Clock Monitor). Este modulo, cuando se activa, monitorea el oscilador primario (LP, XT,HS, EC, RC y el Oscilador del Timer1). Para ello compara este oscilador con la señal de clock de 31Khz del LFINTOSC. Si el oscilador externo llegase a “perder pulsos de clock”, esto será detectado por el FSCM, lo cual activará una bandera indicadora denominada OSFIF (Oscilador Fail Interrupt Flag), la cual si se encuentra debidamente habilitada por medio del bit OSFIE, generará una interrupción, mediante la cual se puede conmutar hacia el oscilador interno usando el bit SCS mencionado anteriormente y pasar a trabajar con el oscilador interno HFINTOSC seteado en algún valor de frecuencia por medio del divisor de frecuencia programable.



Todas estas características pueden ser habilitadas o no, mediante el correcto seteo de los fusibles de configuración usando la directiva **#FUSES.**

Dentro de la carpeta PICC del compilador encontraremos una carpeta denominada DEVICES, dentro de la cual encontraremos los archivos de cabecera que contienen todas las etiquetas para manejar los microcontroladores PIC de forma sencilla y configurar todas las opciones vistas. Existe un archivo de cabecera por cada micro:

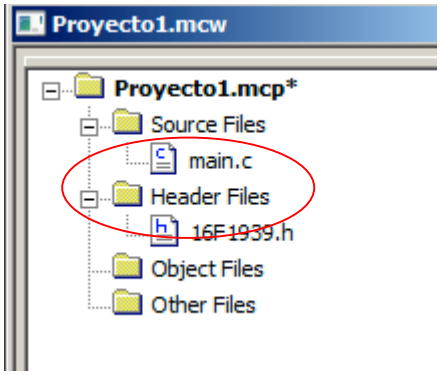




Dentro de este archivo podemos ver el conjunto de etiquetas que nos permiten setear estas opciones parra configurar los fusibles de configuración desde el código, usando la directiva **#FUSES:**

```
1  //////// Standard Header file for the PIC16F1939 device ////////////////
2  #device PIC16F1939
3  #nolist
4  //////// Program memory: 16384x14  Data RAM: 260  Stack: 16
5  //////// I/O: 36   Analog Pins: 14
6  //////// Data EEPROM: 256
7  //////// C Scratch area: 77   ID Location: 8000
8  //////// Fuses: LP,XT,HS,RC,INTRC_IO,ECL,ECM,ECH,NOWDT,WDT_SW,WDT_NOSL,WDT
9  //////// Fuses: PUT,NOPUT,NOMCLR,MCLR,PROTECT,NOPROTECT,CPD,NOCPD,NOBROWNOUT
10 //////// Fuses: BROWNOUT_SW,BROWNOUT_NOSL,BROWNOUT,CLKOUT,NOCLKOUT,NOIESO
11 //////// Fuses: IESO,NOFCMEN,FCMEN,WRT,WRT_EECON2000,WRT_EECON200,NOWRT
12 //////// Fuses: VCAP_A0,VCAP_A5,VCAP_A6,NOVCAP,PLL_SW,PLL,NOSTVREN,STVREN
13 //////// Fuses: BORV25,BORV19,DEBUG,NODEBUG,NOLVP,LVP
14 ////////
```

Estas etiquetas pueden cambian entre microcontroladores porque no todos tienen las mismas opciones de configuración, de hecho algunas de las que mencionamos son exclusivas de los PIC16F1XXX y PIC18F, mientras que en los PIC16F convencionales no se encontrarán. Por ello es buena práctica que a nuestros proyectos le adjuntemos el archivo de cabecera de nuestro microcontrolador, a modo de consulta:



Para asimilar estos conceptos vamos a aplicarlos en un proyecto inicial, el cual consiste e leer 3 pulsadores y accionar 3 leds.
En este caso configuraremos las opciones de los fusibles de configuración de la siguiente manera:

- Oscilador: Usaremos el Interno a una frecuencia de 4 MHz
- FSCM: No lo usaremos pues trabajamos con oscilador interno
- IESO: tampoco lo usaremos por la misma razón que el anterior
- WatchDog Timer: Lo desactivado ya que es una aplicación no comercial
- Power Up Timer: Lo activaremos para asegurar un arranque estable
- STVREN: no lo usaremos pues al ser sencilla la aplicación no es posible que se desbode el STACK
- DEBUG: no lo usaremos pues no vamos a depurar código
- BROUNOUT: no lo usaremos pues no es una aplicación comercial
- VCAP: no lo usaremos pues no trabajamos con teclados MTOUCH
- PROTECT: no lo usaremos pues no necesitamos proteger el código
- CPD: tampoco necesitamos proteger los datos en EEPROM pues no la usaremos
- WRT: tampoco usaremos la protección contra escritura de la memoria FLASH de programa pues esta opción se usa por lo general en los modos de auto programación cuando el microcontrolador viene equipado con un pequeño firmware llamado **BOOTLOADER.**

En estas condiciones (las cuales usamos usualmente cuando realizamos proyectos NO COMERCIALES, y que usaremos para todos los proyectos del libro), los fusibles de configuración se setearán mediante la directiva #FUSES de la siguiente forma:

```
#include<16F1939.h> //archivo de cabecera del PIC usado
#FUSES INTRC_IO   //Oscilador Interno y  tanto RA6 como RA7 son I/O
#FUSES NOCLKOUT   //No sale el clock interno hacia afuera
#FUSES NOWDT      //Watch Dog Timer apagado
#FUSES PUT        //Power Up Timer activado
#FUSES NOBROWNOUT //Reset ante variaciones de VCC desactivado
#FUSES NOIESO     //Switch del oscilador en el encendido apagado
#FUSES NOFCMEN    //detector de falla del oscilador principal apagado
#FUSES NOSTVREN   //reset por desborde del stack desactivado
#FUSES NOLVP      //Programación en baja tensión desactivada
#FUSES NOPROTECT  //Protección contra lectura de la FLASH desactivada
#FUSES NOCPD      //Protección de la EEPROM desactivada
#FUSES NOVCAP     //Regulador de voltaje para el MTOUCH apagado
```

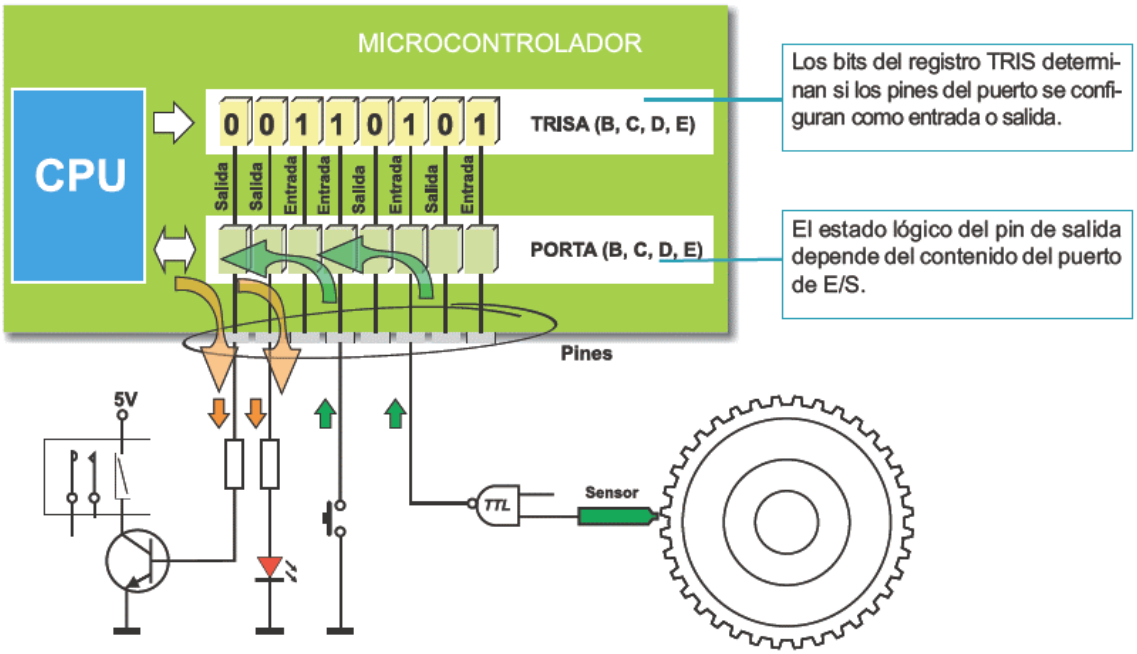
Para hacer más comprensible el código desde el punto de vista del programador, crearemos una serie de etiquetas para el manejo de los puertos de entrada/salida :

```
#define LED1 PIN_B0     //PORTB RB0
#define LED2 PIN_B1     //PORTB RB1
#define LED3 PIN_B2     //PORTB RB2
#define SW1  PIN_A0     //PORTA RA0
#define SW2  PIN_A1     //PORTA RA1
#define SW3  PIN_A2     //PORTA RA2
```

El compilador CCS, es un compilador orientado a los programadores, más que a los electrónicos, y por tal motivo tiende a ser más amigable la arquitectura y manejo del mismo al usuario. Por tal motivo a la hora de nombrar los puertos del microcontrolador, dentro del archivo de cabecera **16F1939.h** ha llamado a los mismos no por el nombre que figuran en el data manual, sino por el seudónimo PIN_XY donde X es la letra del puerto al que pertenecen por ejemplo A,B,C,D,etc, y la letra Y representa el número del bit.

Los Puertos de **entrada/salida (PORTS I/O)** son usados para controlar el entorno del microcontrolador. El PIC16F1939 cuenta con un total de 36 puertos. Estos si bien se controlan de forma independiente, se agrupan en arreglos de 8 bits. De esta forma nuestro PIC tiene 5 grupos denominados PORTA (8bits), PORTB (8bits), PORTC (8bits) , PORTD (8bits) y PORTE (4 bits). Cada puerto de forma independiente se denomina RXY, donde X es la letra del grupo al cual pertenece e Y es el número de orden, el cual puede ser desde 0 (menor peso) a 7 (el de mayor peso).

Todos los puertos son bi-direccionales, es decir que pueden ser programados para trabajar como puerto de entrada o salida. La dirección de los puertos esta controlada por unos registros denominados TRISX, donde X es la letra que corresponde al PORT que controlan:



Según el estado lógico que se cargue en un bit del registro TRIS, el PORT correspondiente trabajará como entrada o salida (1= Entrada, 0=Salida).
Los puertos tiene la capacidad de entregar como máximo una corriente de 25ma. Y reconocen el estado lógico como la lógica TTL, es decir que cualquier tensión por encima de los 2,5V es reconocida como 1 lógico, y por debajo de 0,8V como un cero. Por otra parte entregan una tensión 1 lógico como los CMOS, la cual llega a VCC.

En los microcontroladores PIC la mayoría de los puertos están multiplexados con otras funciones, como ser puertos de comunicaciones, entradas para el conversor analógico, entradas a comparadores de voltaje, salidas PWM, etc.

Cuando un PIC arranca su operación, la inicia con sus capacidades analógicas activas, esto significa que aquellas entradas que se conectan al conversor analógico digital o a los comparadores, o como en el caso del PIC16F1939, las entradas MTOUCH.

Es por esta razón que el usuario debe configurar dichas entradas como analógicas o digitales antes de usarlas. Para ello existe saciado a cada PORT, un registro denominado ANSELA para el PORTA, ANSELB para el PORTB, ANSELE para el PORTE y ANSELD para el PORTD. Para que un puerto determinado funcione como digital, al ANSEL correspondiente hay que ponerlo en estado lógico 1.

Para simplificar el manejo de los puertos CCS tiene una serie de funciones embebidas, es decir que están incorporadas en el mismo compilador, y que no se necesita agregar ningún archivo de cabecera para usarlas.
Estas son:

```
output_high(PIN_XY); //pone en uno un pin determinado
output_low(PIN_XY); //pone en cero un pin determinado
output_bit(PIN_XY,estado); //pone en cero o uno un pin determinado
output_X(valor);  //saca un valor de 8 bits por un puerto determinado
input(PIN_XY);    //lee el estado de un pin y nos devuelve el mismo
input_X();  //lee el estado de un puerto completo y devuelve el mismo
```

A continuación haremos uso de estas funciones, las cuales se encargan además de configurar el puerto como salida o entrada.
Además de estas funciones se incorpora una función mediante la cual se puede setear que pines funcionaran como analógicos o no, la misma se denomina:

```
setup_adc_ports(pines analógicos) //setea que puertos son analogicos
```

Para nuestro caso como no usaremos los puertos analógicos, le pasaremos a esta función la etiqueta NO_ANALOGS, como figura en el archivo de cabecera 16F1939.h:

```
552   #define sAN13          0x00002000    //| B5
553   #define NO_ANALOGS              0    // None
554   #define ALL_ANALOG     0x2F073F00    // A0 A1 A2
```

De esta forma nuestra configuración de puertos analógicos y digitales quedará de la siguiente forma:

```
setup_adc_ports(NO_ANALOGS); //configuramos los puertos como digitales
```

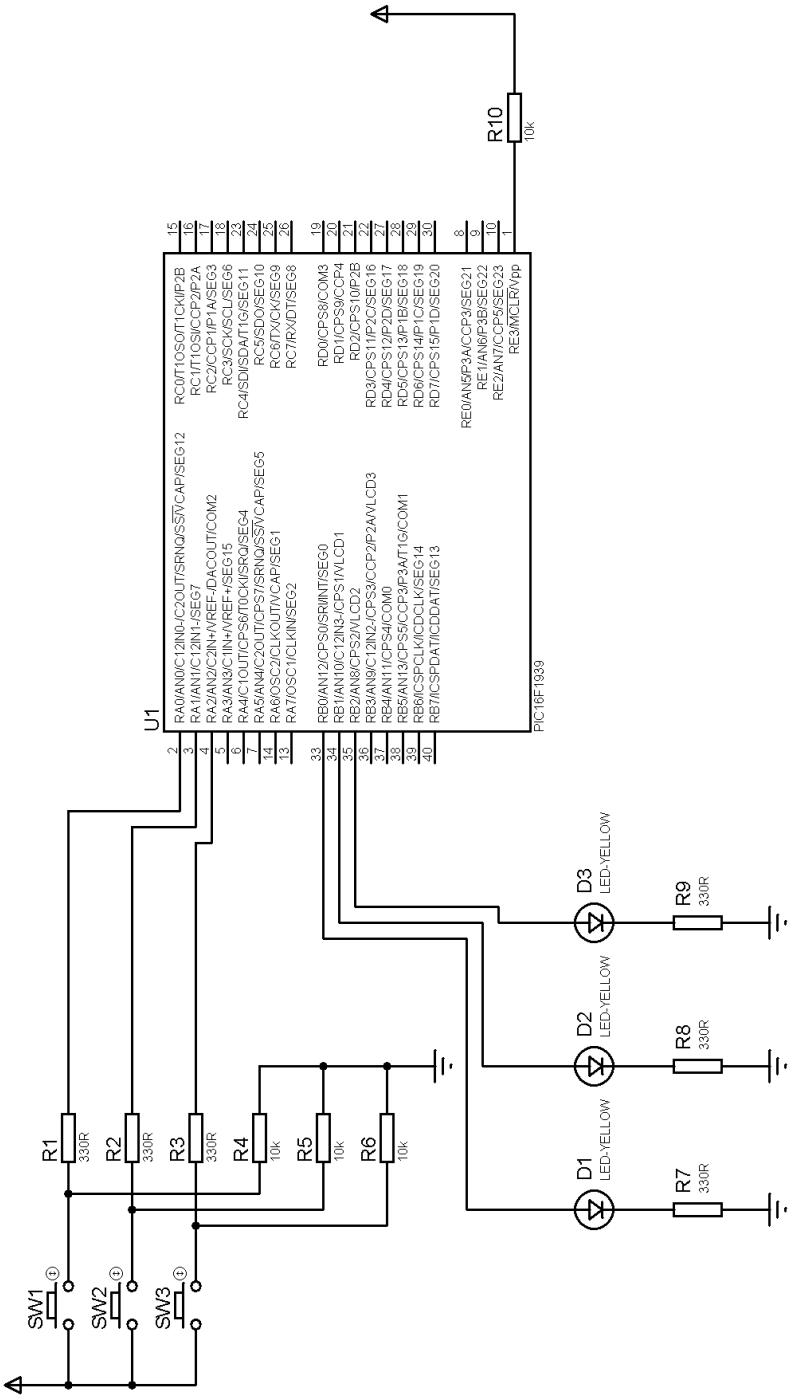
Lo que nos queda simplemente es el código, el cual estará incluido dentro de un bucle infinito, realizado mediante un while(TRUE), ya que esa será la tarea que eternamente deberá realizar el microcontrolador.
La tarea es sencilla: nuestra CPU solo debe "ver" que pulsador (SW) se accionó y conforme a ello encenderá un LED determinado. Para cubrir este objetivo usaremos las funciones para control I/O embebidas y las estructuras condicionales IF:

```
//bucle principal
while(TRUE){

      if(input(SW1))                //chequeamos si SW1=1
            output_high(LED1);      //si si, encendemos LED1
      else                          //sino
            output_low(LED1);       //apagamos LED1

      if(input(SW2))                //chequeamos si SW2=1
            output_high(LED2);      //si si, encendemos LED2
      else                          //sino
            output_low(LED2);       //apagamos LED2

      if(input(SW3))                //chequeamos si SW3=1
            output_high(LED3);      //si si, encendemos LED3
      else                          //sino
            output_low(LED3);       //apagamos LED3
      }
```

En el siguiente listado mostramos el programa completo:

```
#include <16F1939.h>
#fuses INTRC_IO,NOWDT,PUT,MCLR,NOPROTECT,NOCPD,NOBROWNOUT,NOCLKOUT
#fuses NOIESO,NOFCMEN,NOWRT,NOVCAP,NOSTVREN,NODEBUG,NOLVP
#use delay(internal,clock=4000000)
#define LED1 PIN_B0     //PORTB RB0
#define LED2 PIN_B1     //PORTB RB1
#define LED3 PIN_B2     //PORTB RB2
#define SW1  PIN_A0     //PORTA RA0
#define SW2  PIN_A1     //PORTA RA1
#define SW3  PIN_A2     //PORTA RA2

void main(void)
{
    setup_oscillator(OSC_4MHZ|OSC_INTRC|OSC_PLL_OFF);
    setup_adc_ports(NO_ANALOGS);
    while (TRUE)
    {
        if(input(SW1))  //chequeamos si SW1=1
            output_high(LED1); //si si, encendemos LED1
      else              //sino
            output_low(LED1); //apagamos LED1

      if(input(SW2))    //chequeamos si SW2=1
            output_high(LED2); //si si, encendemos LED2
      else              //sino
            output_low(LED2); //apagamos LED2

      if(input(SW3))    //chequeamos si SW3=1
            output_high(LED3);//si si, encendemos LED3
      else                //sino
            output_low(LED3); //apagamos LED3

    }
}
```

Observe que en el listado hemos seteado los fusibles de configuración en 2 lineas, aunque también podría haberse realizado todo en una.
El circuito que utilizaremos para hacer funcionar nuestro código, lo presentamos en la siguiente figura:

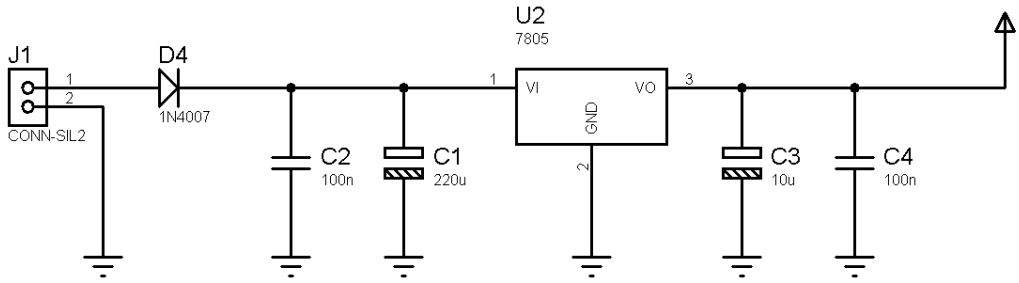


En el circuito es bastante sencillo:

Los resistores R4,R5 y R6 son resistores de PULL DOWN; R1,R2 y R3 limitan la corriente instantánea que se genera al accionar los pulsadores SW1,SW2 y SW3. Los LEDs estan polarizados a unos 10ma aprox. por medio de los resistores R7,R8 y R9. Finalmente el pin MCLR se encuentra conectado a VCC por medio del resistor de PULL UP R10.

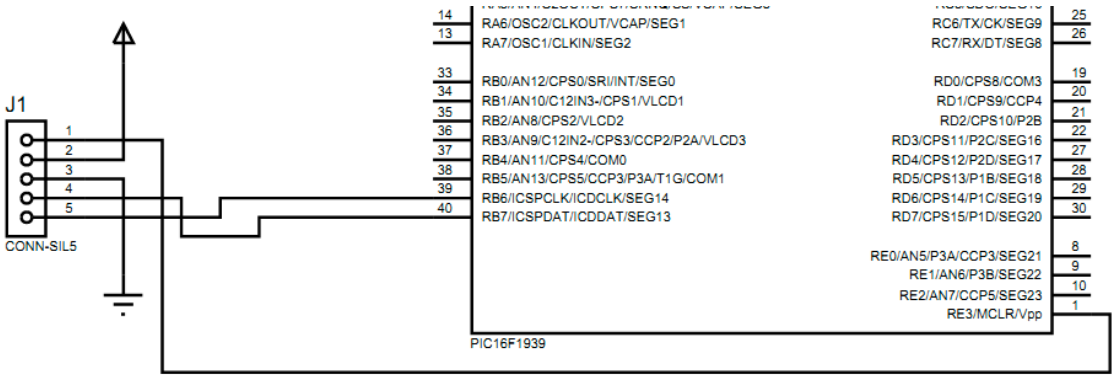
El circuito no necesita de ningún cristal externo ya que se usa el oscilador interno.

La aplicación puede alimentarse desde una batería de 9V o una fuente equivalente usando el siguiente circuito regulador:



El esquemático lo hemos realizado con el Proteus, y es por esta razón que debemos advertirles que los pines de alimentación NO FIGURAN EN EL CIRCUITO, ya que PROTEUS da por sentado que el técnico ya sabe como alimentarlo.

Dichos pines, para los encapsulados de 40 terminales son VCC:11 y 32, GND:12 y 31
Para poder programar nuestro código dentro del microcontrolador, prepararemos en el extremo del circuito impreso un conector de tira de pines (5 postes) para realizar la conexión ICSP según el siguiente esquema:



Como esta conexión será un estándar en nuestros circuitos ya que es una forma muy práctica de programar nuestro programa de aplicaciones sobre el PCB sin retirar el microcontrolador del mismo. Este mismo esquema de conexión ICSP es la que utiliza la herramienta del tipo **"PICKit2"** diseñada por la firma ***"edudevices"*** **(**www.edudevices.com.ar**)** para el mercado de Argentina. El lector podrá observar, luego con el uso, la velocidad que adquiere la depuración de un código utilizando este tipo de herramientas y las bondades del entorno CCS.

Continuará .....