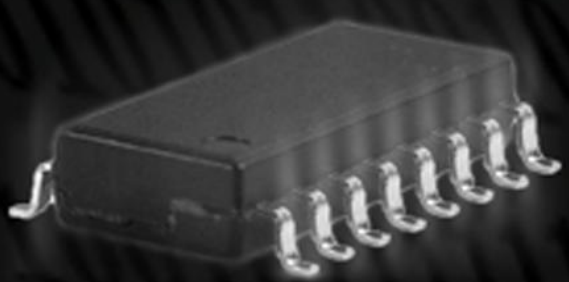
Tutorial

MPLAB C18



Nueva Edición

2010

Prefacio

Este es un tutorial básico de MPLAB C18 Microchip, donde se irá adquiriendo conceptos a medida que los utilizaremos en el desarrollo de los ejemplos, de esta manera lo que se presenta teóricamente lo asociamos inmediatamente con la práctica. Aunque claro está que el lenguaje es muy amplio y no se puede hacer ejemplos de cada concepto.

LICENCIA

Índice general

Capítulo 1

Introducción

1.1. Entorno de programación

MPLAB IDE

- Ensamblador, enlazador, gestión de proyectos, depurador y simulador. La interfaz gráfica del usuario MPLAB IDE sirve como un único entorno para escribir, compilar y depurar código para aplicaciones embebidas. Permite manejar la mayoría de los detalles del compilador, ensamblador y enlazador, quedando la tarea de escribir y depurar la aplicación como foco principal del programador (usuario)

- Gratuito, se puede descargar de www.microchip.com

1.2. Compilador

MPLAB C18

- MPLAB C18 es un compilador cruzado que se corre en un PC y produce código que puede ser ejecutado por la familia de microcontroladores de Microchip PIC18XXXX. Al igual que un ensamblador, el compilador traduce las declaraciones humanas en unos y ceros para ser ejecutados por el microcontrolador.

- Sigue la norma ANSI C, salvo en particularidades de los microcontroladores

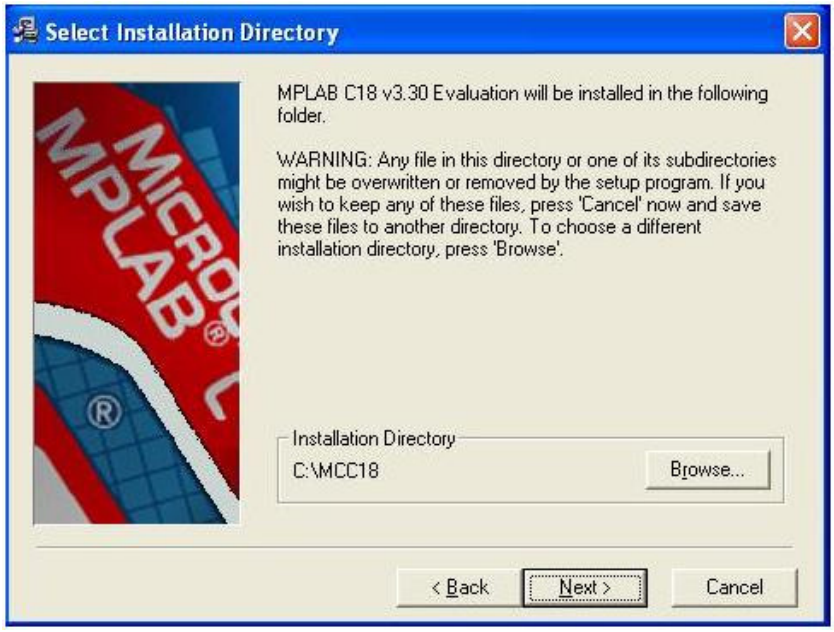
- Librerías para comunicaciones SPI, I2C, UART, USART, generación PWM, cadena de caracteres y funciones matemáticas de coma flotante

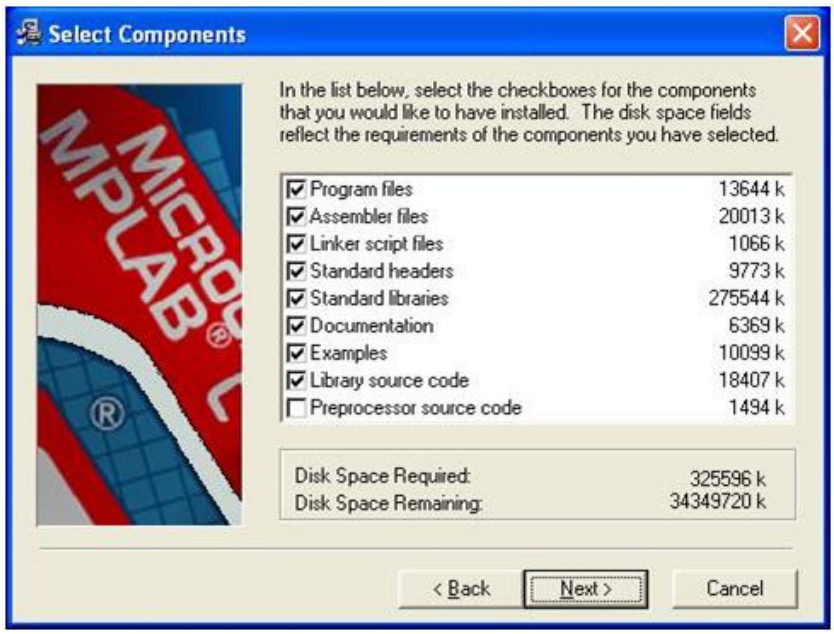
- Maneja números reales de 32 bits (float y double)

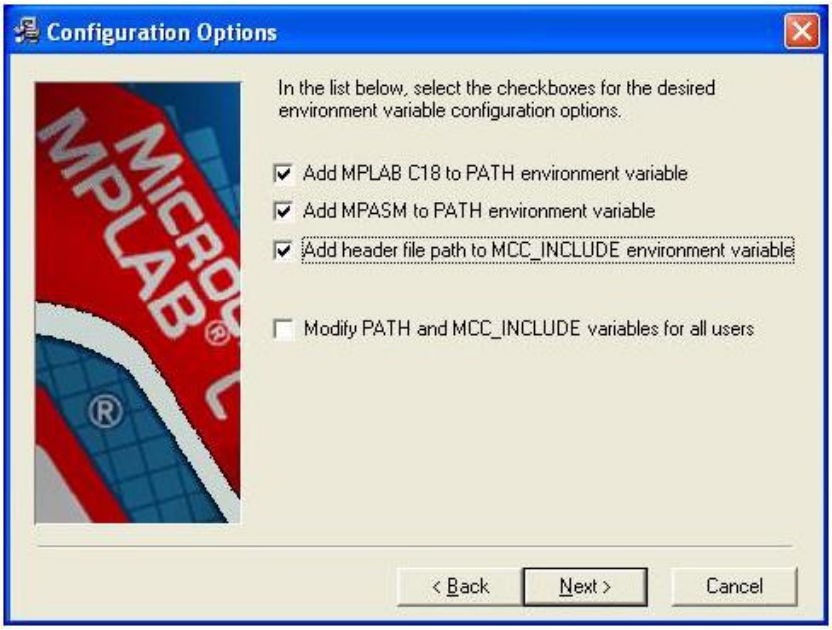
- Versión demo de 60 días, descargable de www.microchip.com

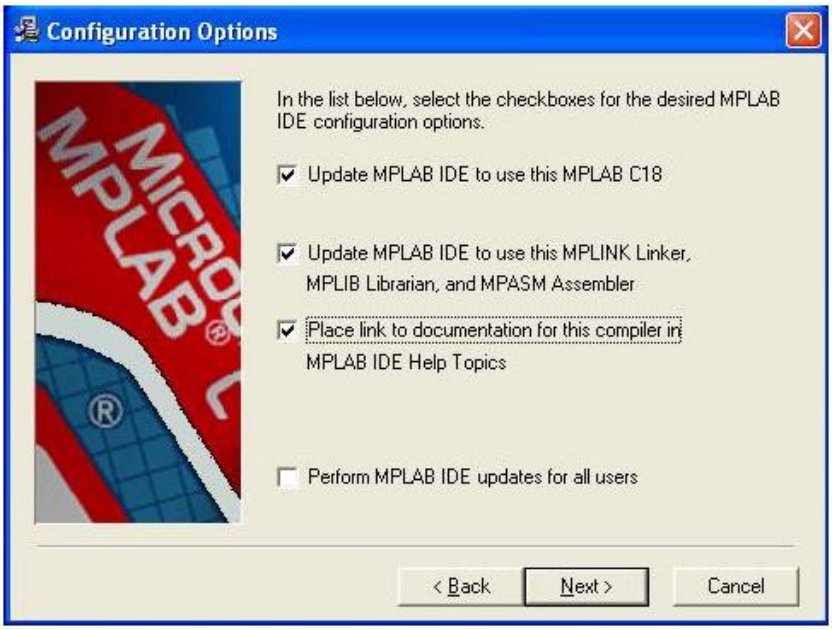
1.3. Instalación

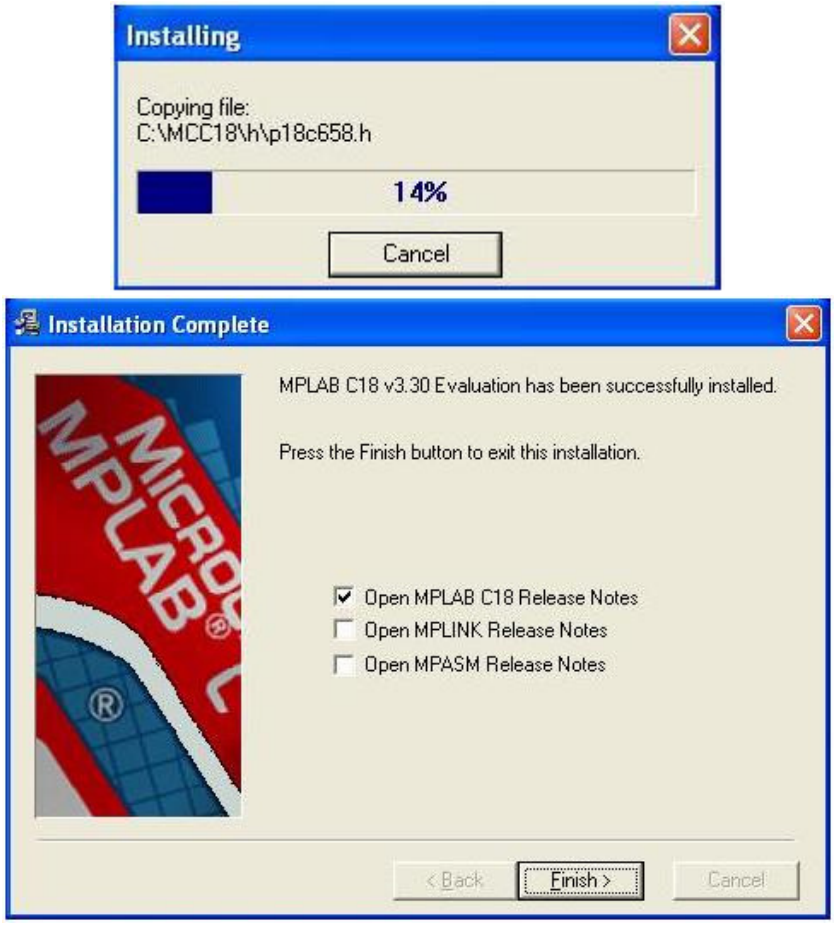
Se debe bajar directamente desde Microchip, hay disponible una versión gratuita para estudiantes que es un demo de 60 días. También para poder descargarlo es necesario registrarse. Una vez descargado ejecutar el instalador **MPLAB-C18-Evaluation-v3-30** versión actualmente disponible. Para la instalación seguimos los siguientes pasos:


Select Installation Directory
MPLAB C18 v3.30 Evaluation will be installed in the following folder.
WARNING: Any file in this directory or one of its subdirectories might be overwritten or removed by the setup program. If you wish to keep any of these files, press 'Cancel' now and save these files to another directory. To choose a different installation directory, press 'Browse'.
Installation Directory
C:\MCC18
Browse...
< Back
Next >
Cancel


Select Components
In the list below, select the checkboxes for the components that you would like to have installed. The disk space fields reflect the requirements of the components you have selected.
Program files 13644 k
Assembler files 20013 k
Linker script files 1066 k
Standard headers 9773 k
Standard libraries 275544 k
Documentation 6369 k
Examples 10099 k
Library source code 18407 k
Preprocessor source code 1494 k
Disk Space Required: 325596 k
Disk Space Remaining: 34349720 k
< Back
Next >
Cancel

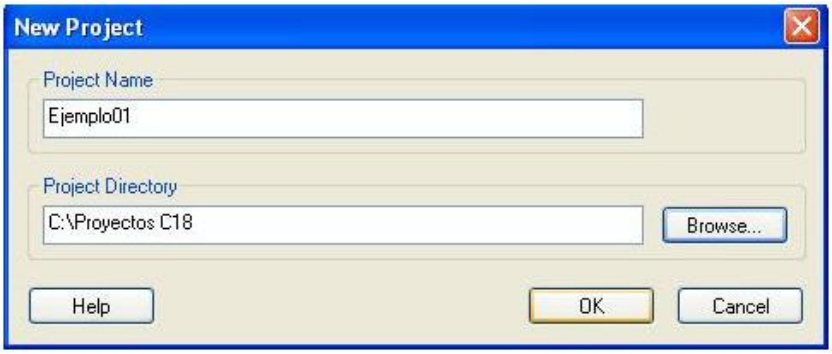

Configuration Options
In the list below, select the checkboxes for the desired environment variable configuration options.
Add MPLAB C18 to PATH environment variable
Add MPASM to PATH environment variable
Add header file path to MCC_INCLUDE environment variable
Modify PATH and MCC_INCLUDE variables for all users
< Back
Next >
Cancel


Configuration Options
In the list below, select the checkboxes for the desired MPLAB IDE configuration options.
Update MPLAB IDE to use this MPLAB C18
Update MPLAB IDE to use this MPLINK Linker, MPLIB Librarian, and MPASM Assembler
Place link to documentation for this compiler in MPLAB IDE Help Topics
Perform MPLAB IDE updates for all users
< Back
Next >
Cancel

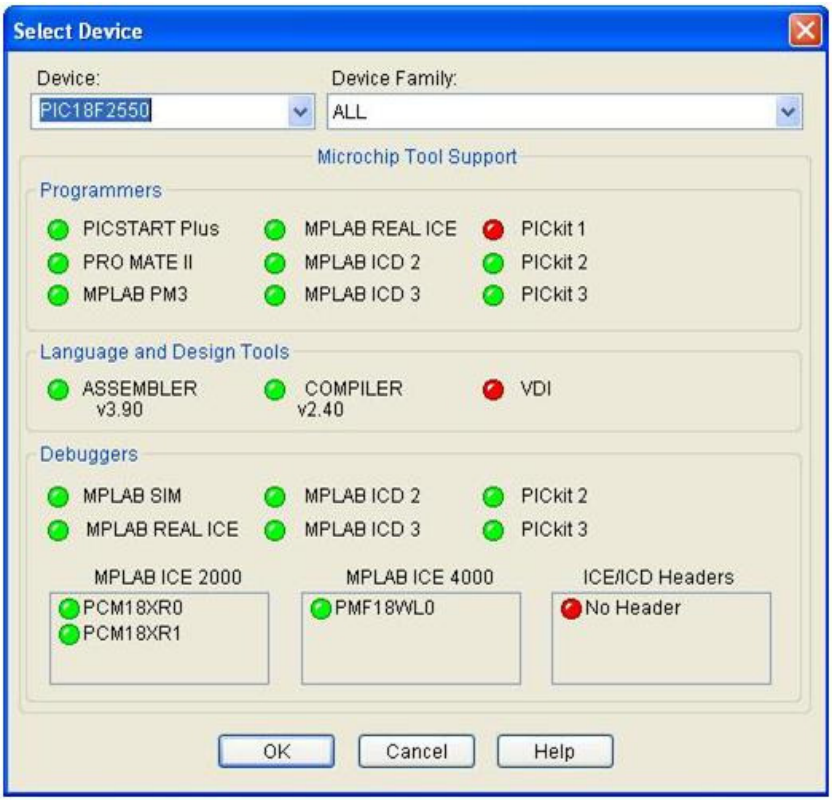


1.4. Creación de un nuevo proyecto

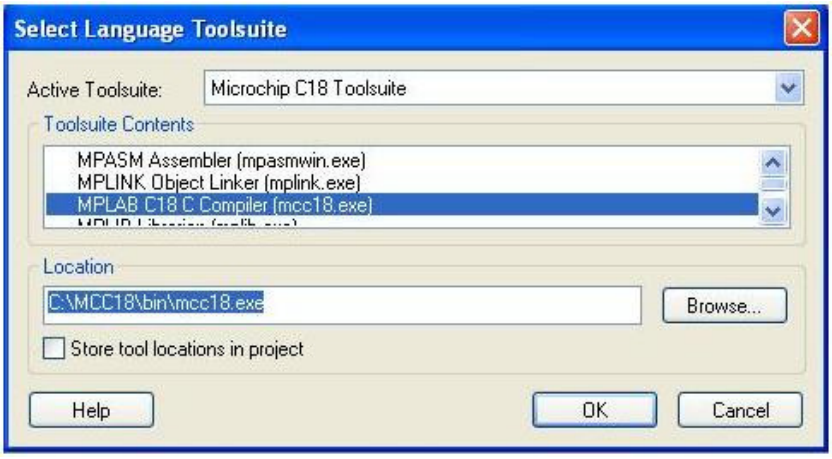
Project - New
Nos aparecerá una pantalla donde le indicamos el nombre de nuestro proyecto y la carpeta donde será guardado.



Pasamos a configurar el dispositivo con el cual trabajaremos: **Configure - Select Device**



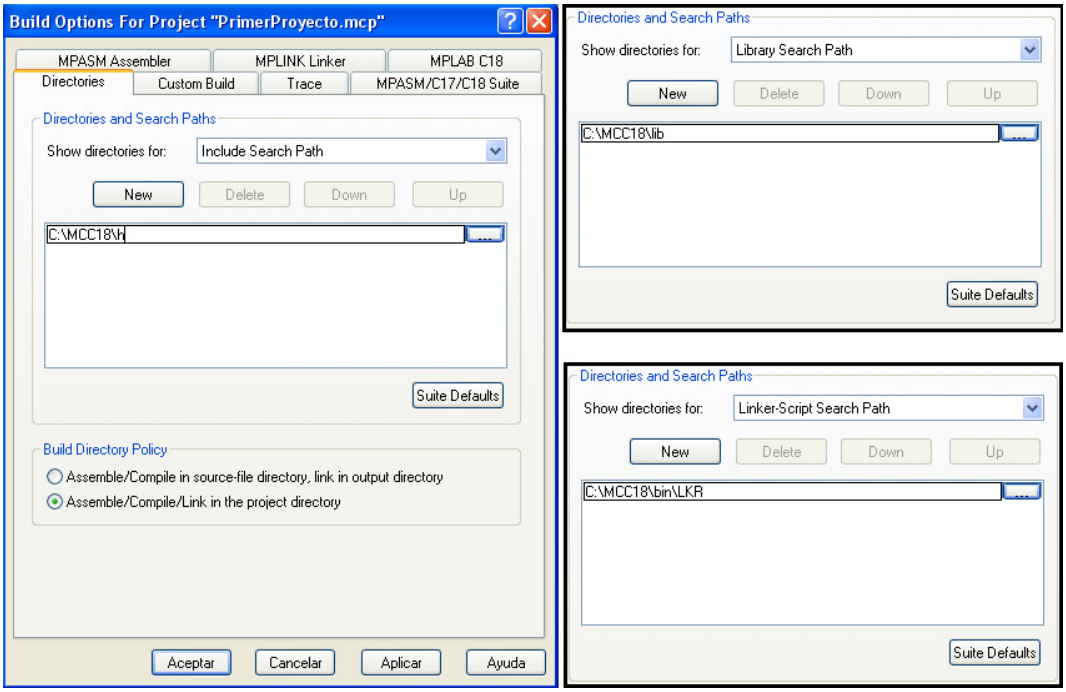
Seleccionamos el compilador:**Project - Select Lenguaje Toolsuite** y nos aseguramos que todas las direcciones son correctas.



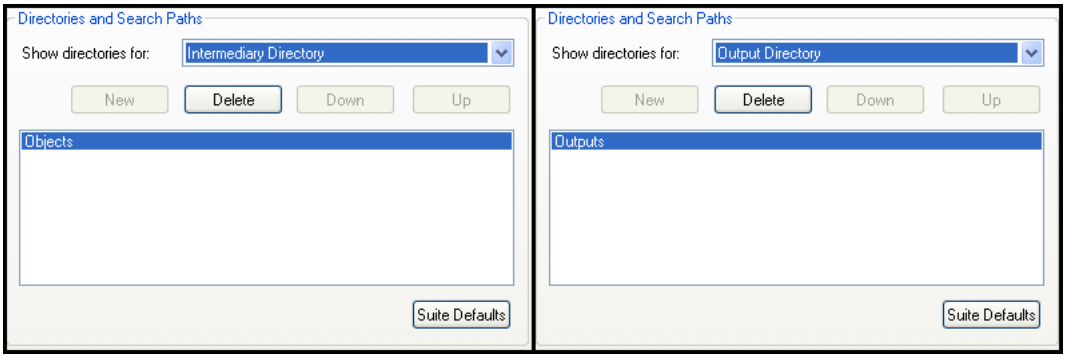
Configuramos los subdirectorios de trabajo: **Project - Build options - Project** Seleccionamos ubicación de ficheros de declaraciones, bibliotecas y script de enlazado.

Show directories for:

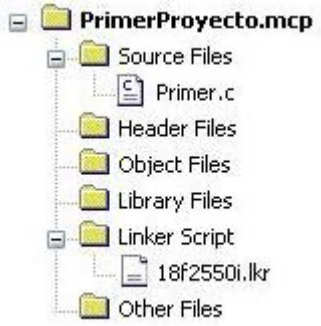
- Include Search Path
- Library Search Path
- Linker-Script Search Path



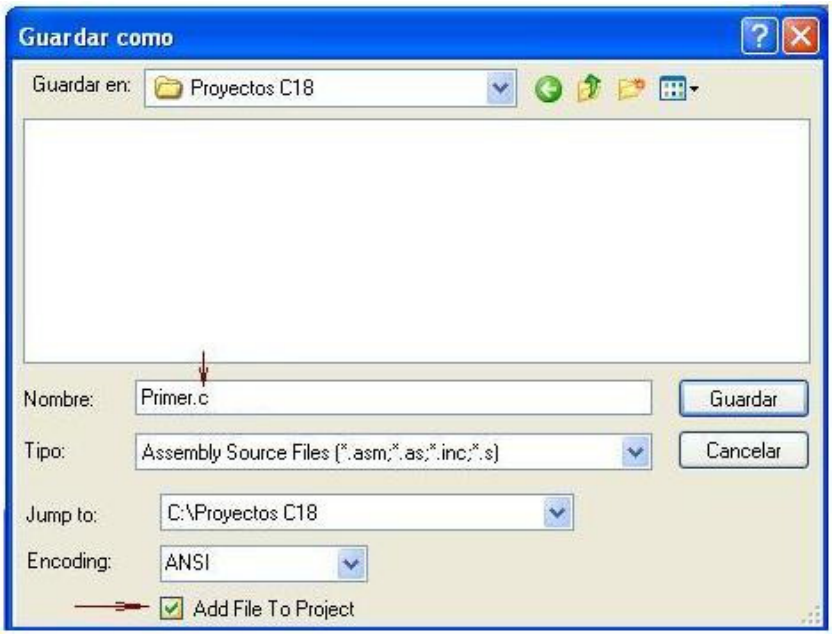
Tambii'e podemos crear carpetas como Output y Objects para organizar en ellos los archivos de salida e intermedios generados en el proceso de compilación.



Nota: Según la versión también se debe agregar al proyecto el archivo (PIC18F utilizado).lkr ubicado en MCC18/lkr, sino produce error de compilación.



Luego vamos a **New File** y lo guardamos en nuestra carpeta eligiendo extensión .c agregándolo a nuestro proyecto.



Con todo esto configurado ya podemos empezar a desarrollar nuestro código.

Capítulo 2

Comenzando con C18

La idea es ir adquiriendo conceptos a medida que los utilizaremos en el desarrollo de los ejemplos, de esta manera lo que se presenta teóricamente lo asociamos inmediatamente con la práctica. Aunque claro esta que el lenguaje es muy amplio y no se puede hacer ejemplos de cada concepto.

En el desarrollo de este tutorial se utilizará el microcontrolador PIC18F2550, y como en todo proyecto siempre se debe tener a mano el datasheet de los dispositivos utilizados, para la correcta interpretación y aplicación de las configuraciones realizadas.

2.1. Creando el código

Lo que se presenta aquí es la estructura general de un archivo fuente de C, en donde como primera medida se incluyen las librerías, colección de rutinas, que se van a utilizar en el proyecto. Tenemos las librerías estándar de Ansi C que incluye rutinas para manejo de cadenas de texto y operaciones con datos comunes como funciones matemáticas, librerías especificas del microcontrolador a utilizar (p18Fxxxx.h) la cual contiene estructuras de los registros del microcontrolador para control de sus bits, librerías para control de periféricos del microcontrolador (UART, I2C, SPI, ect) y las librerías propias creadas por el usuario dedicadas al control de un dispositivo externo o una tarea en común. La librería que siempre incluiremos en el archivo principal será la del PIC a usar:

```
#include <p18F2550.h>
```

Luego viene la configuración de los fuses del microcontrolador. O sea configuración de oscilador, watch–dog, Brown–out reset, power–on reset, protección del código, etc. Esto depende del microcontrolador que se utilice:
La sintaxis seria:

```
#pragma config Nombre_del_fuse=estado
```

Para esto es muy útil la ayuda que trae C18, recomiendo mirarla:
C:/MCC18/doc/hlpPIC18ConfigSet.

Ahora viene el código de nuestro programa:

```
main{

}
```

Ejemplo completo:

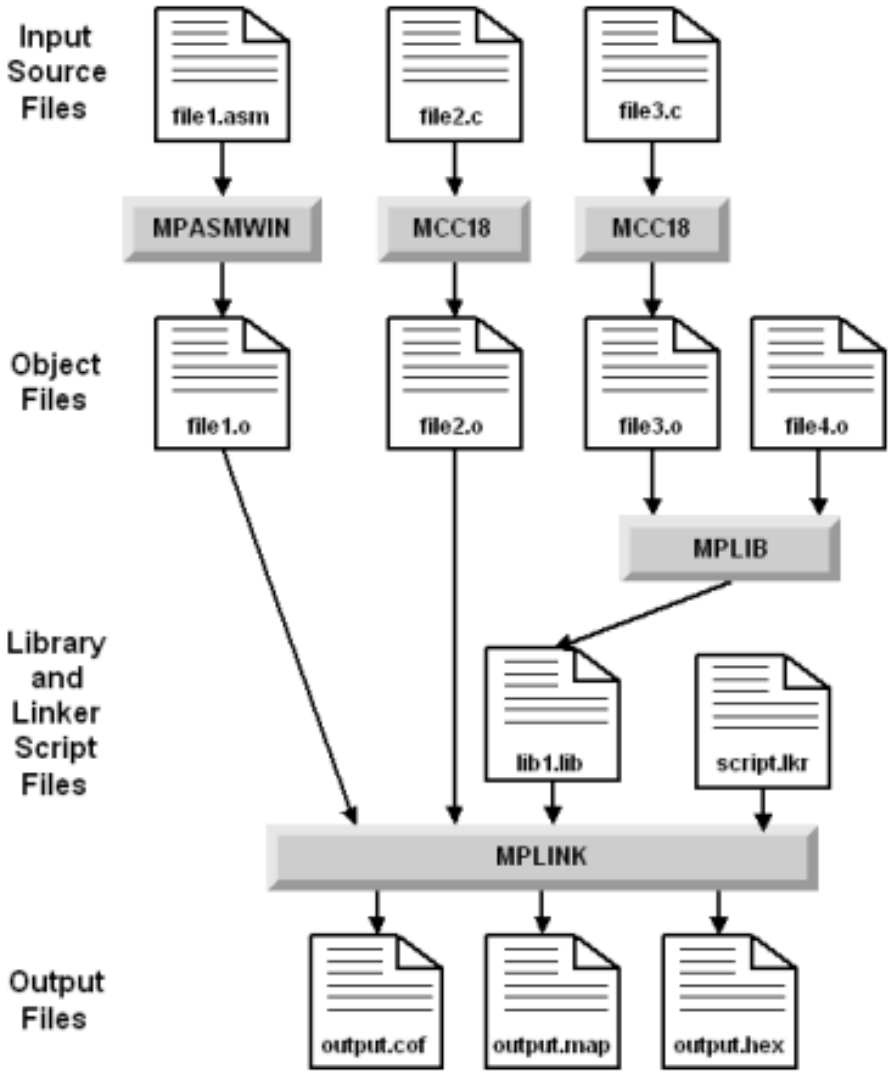
```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador  ** */
#pragma config FOSC=XT_XT,FCMEN=OFF,IESO=OFF,CPUDIV=OSC1_PLL2
#pragma config PWRT=ON,BOR=OFF,BORV=0,WDT=OFF,WDTPS=32768
#pragma config 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 main(void){

    // Sentencias.-
}
```

2.2. El proceso de compilación

El texto fuente es compilado en bloques de código de programa y datos que luego son *enlazados* (linked) con otros bloques de código y datos, y colocados en las regiones de memoria del microcontrolador PIC18XXXX seleccionado. Este proceso se llama *generación* (build) y se suele llevar a cabo muchas veces en el desarrollo de programas en el proceso de probado y depurado. También tenemos la posibilidad de utilizar *make* que solo compila los archivos fuentes que han sido modificados desde la última vez agilizando el proceso.

2.2.1. Flujo de la generación del hex

En la siguiente imagen se tiene un ejemplo de los pasos que lleva un determinado proyecto, donde tenemos 2 archivos fuentes en c (*.c), 1 archivo en assembler (*.asm) y un archivo precompilado (*.o).



Los archivos fuentes *.c son compilados por MPLAB C y el archivo *.asm es ensamblado por MPASM generando los archivos intermedios llamados archivos objetos. Estos archivos junto al *.lkr del microcontrolador son tomados por el enlazador para generar el *.hex que será utilizado para la grabación en la memoria de programa del microcontrolador.

Cabe la posibilidad de agrupar archivos objetos para crear bibliotecas (*.lib) como nos lo entrega Microchip. Estás las encontramos en MCC18/lib en donde existen librerías pre-compiladas para control de periféricos de cada microcontrolador y la librería estándar de C.

El archivo *lkr contiene información de la estructura y capacidades del microcontrolador con el cual se está trabajando, sirve como plantilla para el enlazador para organizar el código de programa y datos generados en el proceso de compilación.

Ahora ya tenemos una idea general de como es la estructura de un programa desarrollado en C y cual es el proceso que sigue en la generación del *.hex necesario para embeber a nuestro microcontrolador. Seguiremos con el estudio de las directivas utilizadas en C para el desarrollo de programas simples y más adelante encararemos el tema de las librerias, su modificación, creación, ect.

2.3. Operadores

Aquí definiremos todos los operadores utilizados por C18.-

2.3.1. Operadores aritméticos

Se utilizan para realizar cálculos matemáticos:

Operador	Descripción
+	suma
-	resta
*	multiplicación
/	división
++	incremento
- -	decremento

Disponemos de dos operandos poco comunes para decrementar o incrementar una variable. Un aspecto poco común es que podemos utilizarlos como prefijos (++k, - -i) o como postfijos (k++, i- -), con la diferencia que utilizando ++k primero se incrementa el valor y luego es utilizada la variable, y k++ utiliza la variable y luego la incrementa:
Ejemplos:

```
k=10;
a=++k; // a = 11, k = 11.
a=k++; // a = 10, k = 11.
```

2.3.2. Operadores de Comparación

Estos operadores se encargan de comparar dos condiciones de una expresión:

Operador	Descripción
==	igual
!=	distinto
<	menor que
>	mayor que
<=	menor o igual que
>=	mayor o igual que

2.3.3. Operadores lógicos

Son los encargados de producir resultados lógicos del tipo TRUE o FALSE

Operador	Descripción
&&	AND
||	OR
!	NOT

Algo a tener en cuenta en el uso de los operandos lógicos es su precedencia, ya que las expresiones vinculadas por && o || son evaluadas de izquierda a derecha, y la evaluación se detiene tan pronto como se conoce el resultado verdadero o falso. Por ejemplo al realizar:

```
if((k<10) && (++i==100)){

}
```

Lo primero que se evalúa es que k sea menor a 10, si no se cumple sin importar el resto salta a la siguiente línea. Si en el diseño del programa se necesitaba el incremento de la variable i el resultado será incorrecto, por lo que debe estudiarse su correcta diagramación.

2.3.4. Operadores bitwise

Son para modificar los bits de una variable:

Operador	Descripción
&	And
|	Or
$\wedge$	Xor
$\sim$	complemento
<<	rotar izquierda
>>	rotar derecha

El operador & es utilizado generalmente para llevar a cero ciertos bits. Por ejemplo:

```
k=0xA5;
a= 0xF0 & k; // a = 0xA0, k = 0xA5.-
```

El operador | es utilizado para setear ciertos bits. Por ejemplo:

```
k=0x03;
a=0x40 | k; // a = 0x43, k = 0x03.-
```

2.4. Estructuras de Control

2.4.1. Estructura if

Esta estructura se utiliza para ejecutar instrucciones en forma condicional, de acuerdo con la evaluación de la expresión. Seria si una condición es dada entonces acción.

```
if(condicion){
//Accion
}
```

Ejemplo:

```
if(PORTA==0x00){  // Es PORTA igual a cero(0)?
    LATB=0xFF;    // Si, entonces cargamos en puerto B el valor 0xFF
}
```

2.4.2. Estructura if-else

En este caso se agrega la instrucción else. Ahora se evalúa una condición original, si es verdadera, se ejecuta y sino no lo es, se ejecuta el bloque debajo de else.

```
if(condicion){
//Accion
}else{
        //Accion
}
```

Ejemplo:

```
if((PORTA&0x0F)!=0x00){  // Es RA0:RA3 distinto a cero(0)?  (RA4:RA7 no importan)
    LATB=0xFF;    // Si, entonces cargamos en puerto B el valor 0xFF
}else{
    LATB=0x00;    // No, entonces cargamos en puerto B el valor 0x00
}
```

2.4.3. Estructura while

Ejecuta un conjunto de instrucciones mientras una condición sea verdadera. La principal característica de esta estructura es que, antes de comenzar el bucle, verifica la condición, por lo que es posible que el bucle no llegue a ejecutarse.

```
while(condicion){
// Sentencias
}
```

Ejemplo:

```
// mientras PORTB sea igual a 0xFF y PORTC sea igual a 0xAA
while(PORTB==0xFF && PORTC==0xAA){
    a++; // Incrementamos en 1 a la variable a.-
}
```

2.4.4. Estructura do-while

Es parecida a un while solo que la condición se evalúa al final, por lo que el bucle se ejecutara por lo menos una vez.

```
do {
// Sentencias
} while (condicion);
```

Ejemplo:

```
do {
    LATB=(PORTB<<1); // Rotamos a la izquierda valor cargado en PORTB.-
} while (PORTC==0x00 || PORTD!=0xFF);
// mientras PORTC sea igual a 0x00 o PORTD sea distinto a 0xFF
```

2.4.5. Estructura for

Esta estructura se usa para ejecutar un bloque de código cierto número de veces. Posee un valor de inicio, un valor final y un valor de incremento.

```
for(valor inicial; valor final; valor de incremento ){
//Sentencias
}
```

Ejemplo:

```
// k comienza en 15, re decrementa en 2 por cada ciclo mientras..
// ..k sea mayor a 3.-
for(k=15;k>3;k-=2){

   LATC=k;
}
```

2.4.6. Estructura switch

Esta estructura permite ejecutar un bloque de código de acuerdo con el valor de una variable o expresión:

```
switch(Variable){
   case 0x01:
      //Sentencias.-
   break;
   case 0x02:
      //Sentencias.-
   break;
   default:
      //Sentencias.-
   break;
}
```

Default: ejecuta esa sentencia si no es ninguna de las anteriores.
Ejemplo:

```
switch(PORTC){
   case 0x01:
      LATB=0xAA;
   break;
   case 0x02:
      LATB=0x55;
   break;
   default:
      LATB=0xFF;
   break;
}
```

2.5. Accediendo a los bits de un registro

En la librería *p18fxxxx.h* encontramos la definición de los bits de cada registro mediante una estructura/unión con el nombre **REGISTRO***bits*. Para acceder individualmente a cada uno de los bits utilizandomos el operando punto de la siguiente manera:

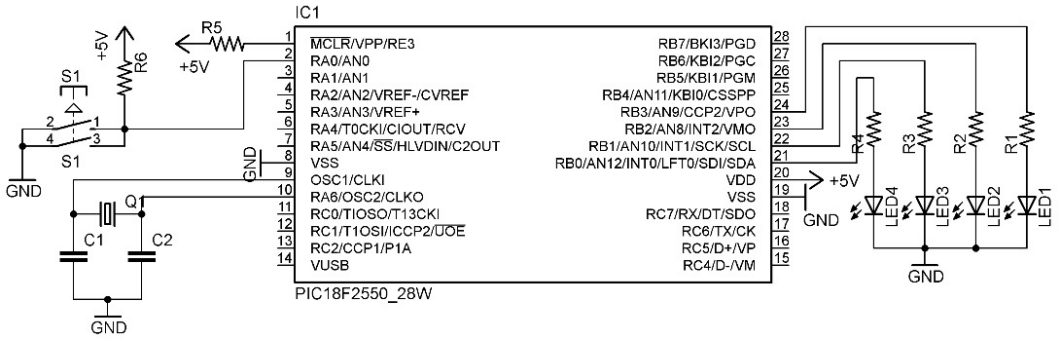
```
PORTBbits.RB0
SSP1CON2bits.ACKDT
LATBbits.LATB0
```

2.6. Primer ejemplo, control de leds con un pulsador

2.6.1. Objetivo

Encender 4 leds del puerto B mientras se mantenga accionado el pulsador.

2.6.2. Hardware



2.6.3. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.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 main(void){

    ADCON1=0x0F;          /* Todos entradas/salidas digitales */
    TRISA=0xFF;           /* Todos como entradas */
    TRISB=0xF0;           /* Nible bajo como salida */

    LATB=0x00;            /* Todos los leds apagados */
    while(1){             /* Bucle infinito */
        if(PORTAbits.RA0==1){   /* Se testea estado del pulsador */
            LATB=0x00;  /* Si esta en 1 logico apagamos leds */
        }else{
            LATB=0x0F;  /* Sino encedemos todos los leds */
        }
    }
}
```

2.7. Variables

Una variable es la asignación de un nombre a un espacio determinado en la memoria, el espacio dependerá del tipo de variable. C18 define los siguientes tipos:

Tipo	Tamaño	Mínimo	Máximo
char	8-bits	-128	127
unsigned char	8-bits	0	255
int	16-bits	-32768	32767
unsigned int	16-bits	0	65535
short	16-bits	-32768	32767
unsigned short	16-bits	0	65535
short long	24-bits	-8388608	8388607
unsigned short long	24-bits	0	1677215
long	32-bits	-2147483648	2147483647
unsigned long	32-bits	0	4294967295
float	32-bits	exp(-126)	exp(128)
double	32-bits	exp(-126)	exp(128)

Según dónde estén declaradas, las variables pueden ser globales (declaradas fuera de todo procedimiento o función) o locales (declaradas dentro de un procedimiento o función). Las primeras serán accesibles desde todo el código fuente y las segundas sólo en la función donde estén definidas.

2.7.1. Almacenamiento de los datos "endianness"

El ordenamiento de los datos en una variable multi-byte se realiza en little-endian. El byte menos significativo ocupa la posición más baja.

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

Da como resultado:

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

Ejemplo:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.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
// Variables globales.-
unsigned char k;
long p;
float Temp;

void main(void){
    // Variables locales:
    unsigned int r,s;

    // Sentencias.-
}
```

2.7.2. Modificadores de las variables

Mplab C18 utiliza los modificadores establecidos por ANSI:

Auto: las variables declaradas fuera de las funciones son globales y las declaradas en la función son locales. Si no se inicializan toman un valor indefinido.

Static: variables locales a una función, y sirven para retener el valor de la variable en llamadas sucesivas a dicha función.

Extern: La variable declarada pertenece a otro módulo, por lo que no es necesario reservar memoria para ella.

Const: El contenido de la variable es fijo.

Volatile: el contenido de la variable puede cambiar.

Register: La variable declarada debe guardarse en un registro del microcontrolador.

Overlay: Se aplica a variables locales, hace un almacenamiento estático y las inicializa en cada llamada.

Ram: La variable se sitúa en la memoria de datos.

Rom: la variable se sitúa en la memoria del programa. Por lo general se usa para cadena de caracteres contantes.

Especificación de banco de memoria de datos:

Far: La variable puede ir en cualquier banco.

Near: La variable tiene que estar en el banco de acceso. Cuando la variable se ubica en la memoria de programa se ubicará en los primero 64k, y cuando la variable se ubica en la RAM se ubica en las primeras 0x5F posiciones de la memoria, generando menos código y acceso más rápido.

En la siguiente imagen se puede ver el assembler generado para los distintos tipos de variables:

```
char k,w;
near rom char r;
far rom char s;
```

08A	k	0x00
00F8	r	0x00
00F9	s	0x00
08B	w	0x00

```
22:                     s=15;
  00A6    0EFB    MOVLW 0xfb
  00A8    6EF6    MOVWF 0xff6, ACCESS
  00AA    0E00    MOVLW 0
  00AC    6EF7    MOVWF 0xff7, ACCESS
  00AE    0E00    MOVLW 0
  00B0    6EF8    MOVWF 0xff8, ACCESS
  00B2    0E0F    MOVLW 0xf
  00B4    6EF5    MOVWF 0xff5, ACCESS
  00B6    000C    TBLWT*
23:                     r=10;
  00B8    0EFA    MOVLW 0xfa
  00BA    6EF6    MOVWF 0xff6, ACCESS
  00BC    0E00    MOVLW 0
  00BE    6EF7    MOVWF 0xff7, ACCESS
  00C0    0E0A    MOVLW 0xa
  00C2    6EF5    MOVWF 0xff5, ACCESS
  00C4    000C    TBLWT*
24:                     k=20;
  00C6    0100    MOVLB 0
  00C8    0E14    MOVLW 0x14
  00CA    6F8A    MOVWF 0x8a, BANKED
```

```
22:                     w=s;
  00A6    0EF9    MOVLW 0xf9
  00A8    6EF6    MOVWF 0xff6, ACCESS
  00AA    0E00    MOVLW 0
  00AC    6EF7    MOVWF 0xff7, ACCESS
  00AE    0E00    MOVLW 0
  00B0    6EF8    MOVWF 0xff8, ACCESS
  00B2    0008    TBLRD*
  00B4    CFF5    MOVFF 0xff5, 0x8b
  00B6    F08B    NOP
23:                     w=r;
  00B8    0EF8    MOVLW 0xf8
  00BA    6EF6    MOVWF 0xff6, ACCESS
  00BC    0E00    MOVLW 0
  00BE    6EF7    MOVWF 0xff7, ACCESS
  00C0    0008    TBLRD*
  00C2    CFF5    MOVFF 0xff5, 0x8b
  00C4    F08B    NOP
24:                     w=k;
  00C6    C08A    MOVFF 0x8a, 0x8b
  00C8    F08B    NOP
```

Para las variables guardadas en la memoria de programa el acceso no es tan inmediato, sino que se realiza mediante las operaciones Table Reads o Table Writes, los cuales mueven los datos entre el espacio de memoria RAM y de Programa. Cuando se trabaja una variable NEAR solo se necesita 16-bits para su direccionamiento, en cambio para una variable FAR (Que puede estar en cualquier banco) se necesitan 24-bits para su direcionamiento. Esto último se podrá observar más claro cuando se trate punteros.

2.7.3. 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 operandos 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 habrá 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 tendrá lógica aparente.

Reglas de promoción automática de expresiones

Estas reglas dicen que el compilador hará estrictamente las conversiones necesarias para llevar todos los operandos al tipo del mayor. El resultado de evaluar una operación aritmética será 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.-

2.8. Secciones

Las secciones son las diversas áreas en la memoria del PIC18XXX, incluyendo la memoria de programa, memoria de datos (RAM), memoria EEPROM y la memoria de la PILA, entre

otros. Si observamos el archivo *.lkr del microcontrolador utilizado, por ejemplo para el 18f2550 tenemos.

```
// File: 18f2550_g.lkr
// Generic linker script for the PIC18F2550 processor

#DEFINE _CODEEND _DEBUGCODESTART - 1
#DEFINE _CEND _CODEEND + _DEBUGCODELEN
#DEFINE _DATAEND _DEBUGDATASTART - 1
#DEFINE _DEND _DATAEND + _DEBUGDATALEN

LIBPATH .

#IFDEF _CRUNTIME
  #IFDEF _EXTENDEDMODE
    FILES c018i_e.o
    FILES clib_e.lib
    FILES p18f2550_e.lib

  #ELSE
    FILES c018i.o
    FILES clib.lib
    FILES p18f2550.lib
  #FI

#FI

#IFDEF _DEBUGCODESTART
  CODEPAGE   NAME=page       START=0x0               END=_CODEEND
  CODEPAGE   NAME=debug      START=_DEBUGCODESTART   END=_CEND        PROTECTED
#ELSE
  CODEPAGE   NAME=page       START=0x0               END=0x7FFF
#FI

CODEPAGE   NAME=idlocs     START=0x200000          END=0x200007       PROTECTED
CODEPAGE   NAME=config     START=0x300000          END=0x30000D       PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE          END=0x3FFFFF       PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000          END=0xF000FF       PROTECTED

#IFDEF _EXTENDEDMODE
  DATABANK   NAME=gpr0       START=0x0               END=0xFF
#ELSE
  ACCESSBANK NAME=accessram  START=0x0               END=0x5F
  DATABANK   NAME=gpr0       START=0x60              END=0xFF
#FI

DATABANK   NAME=gpr1       START=0x100             END=0x1FF
DATABANK   NAME=gpr2       START=0x200             END=0x2FF

#IFDEF _DEBUGDATASTART
  DATABANK   NAME=gpr3       START=0x300             END=_DATAEND
  DATABANK   NAME=dbgspr     START=_DEBUGDATASTART   END=_DEND           PROTECTED
#ELSE //no debug
  DATABANK   NAME=gpr3       START=0x300             END=0x3FF
#FI

DATABANK   NAME=usb4       START=0x400             END=0x4FF          PROTECTED
```

```
DATABANK   NAME=usb5       START=0x500            END=0x5FF          PROTECTED
DATABANK   NAME=usb6       START=0x600            END=0x6FF          PROTECTED
DATABANK   NAME=usb7       START=0x700            END=0x7FF          PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60            END=0xFFF          PROTECTED

#IFDEF _CRUNTIME
  SECTION    NAME=CONFIG     ROM=config
  #IFDEF _DEBUGDATASTART
    STACK SIZE=0x100 RAM=gpr2
  #ELSE
    STACK SIZE=0x100 RAM=gpr3
  #FI
#FI
```

Este define memoria de programa como ***page*** que se extiende desde *0x0000 a 0x7FFF*. Dentro de este espacio se pueden definir secciones ***code*** y ***romdata***. Cuando se encuentra la directiva *#pragma code*, el compilador genera instrucciones de código maquina ubicadas en este sector. En cambio *#pragma romdata* almacena datos en la memoria de programa.
La memoria de datos es definida por varios bancos de registros que contienen 256 bytes (*gpr = Registros de propósito general, usb = Registros para utilización de módulo USB (solo en ciertos microcontroladores)*). Dentro de estos bancos de datos podemos definir dos tipos de secciones, ***udata*** e ***idata***, donde ***udata*** indica datos sin inicialización e ***idata*** datos inicializados.
Note que algunas áreas están marcadas con **PROTECTED**, esto impide al enlazador ubicar instrucciones o datos en estos espacios a menos que se lo indique específicamente.
En nuestro código podemos definir sub-secciones utilizando la directiva #pragma:

```
# pragma udata    [attribute] [section-name [=address]]
# pragma idata    [attribute] [section-name [=address]]
# pragma romdata  [overlay]   [section-name [=address]]
# pragma code     [overlay]   [section-name [=address]]
```

attribute:

access: le indica al compilador que debe incluir la sección al espacio definido por el modo de acceso a datos *access bank* sin necesidad de modificar el banco actual de la ram.

overlay: permite que otras secciones se sitúen en las mismas posiciones físicas. Esto permite conservar memoria situando variables en las mismas posiciones, siempre que no se activen simultáneamente. Pero hay restricciones, las dos secciones deben pertenecer a ficheros distintos y deben tener el mismo nombre.

Ejemplos:

```
#pragma code my_code=0x100
void vFuncion(char k){

}
#pragma code
/*****************************************/
#pragma romdata const_table=0x3000
const rom char my_const_array[5] = {'a','b','c','d','e'};
```

```
#pragma romdata
/*****************************************/
#pragma udata access my_access
near unsigned char av1, av2;
#pragma udata
/*****************************************/
archivo1.c:
#pragma udata overlay my_overlay_data=0x200
int var1, var2;
#pragma udata

archivo2.c:
#pragma udata overlay my_overlay_data=0x200
long var;
#pragma udata
/*****************************************/
#pragma idata my_idata=0x500
char text[]="Hola Mundo";
#pragma idata
```

También podemos crear secciones modificado el archivo *.lkr del microcontrolador utilizado, subdividiendo un determinado banco:

```
DATABANK        NAME=gpr1       START=0x100     END=0x1EF
DATABANK        NAME=gpr1_      START=0x1F0     END=0x1FF  PROTECTED
#IFDEF _CRUNTIME
  SECTION    NAME=CONFIG        ROM=config
  SECTION    NAME=MISECCION     RAM=gpr1_
  ...
```

*gpr1_* es un banco de 16 bytes que posteriormente se le denomina como sección **MISECCION**. Como dijimos anteriormente con **PROTECTED** impedimos que el enlazador asigne automáticamente otras variables no indicadas por el programador (usuario) en la sección creada. Y para asignar variables a tal espacio hacemos:

```
#pragma udata MISECCION
    char k;
#pragma udata
```

En el caso de necesitar crear un arreglo de variables (véase más adelante) de un tamaño mayor a 256 bytes si o si debemos unir 2 o más de estos bancos para permitirle al enlazador (linker) ubicarlo en el espacio de memoria.

```
DATABANK        NAME=gpr1_      START=0x100     END=0x2FF
//DATABANK      NAME=gpr2       START=0x200     END=0x2FF

#IFDEF _CRUNTIME
  SECTION    NAME=CONFIG        ROM=config
  SECTION    NAME=MISECCION     RAM=gpr1_
  ...
```

Se unen los bancos *gpr1* y *gpr2* para formar *gpr1_* que contiene 512 bytes.

```
#pragma udata MISECCION
    char k[512];
```

```
#pragma udata
```

ATENCION: Se debe tener precaución al trabajar de esta manera, ya que, una variable multi-byte no puede pertenecer a 2 bancos consecutivos.

2.9. Demoras

Para utilizar demoras en nuestro código debemos incluir la librería delays.h. En ella tenemos 4 funciones con las cuales podemos hacer cualquier tipo de demoras, pero son tediosas dado que siempre necesitamos realizar cálculos:

```
Delay10TCYx(i)     -> 10.Tcy.i
Delay100TCYx(i)    -> 100.Tcy.i
Delay1KTCYx(i)     -> 1000.Tcy.i
Delay10KTCYx(i)    -> 10000.Tcy.i
```

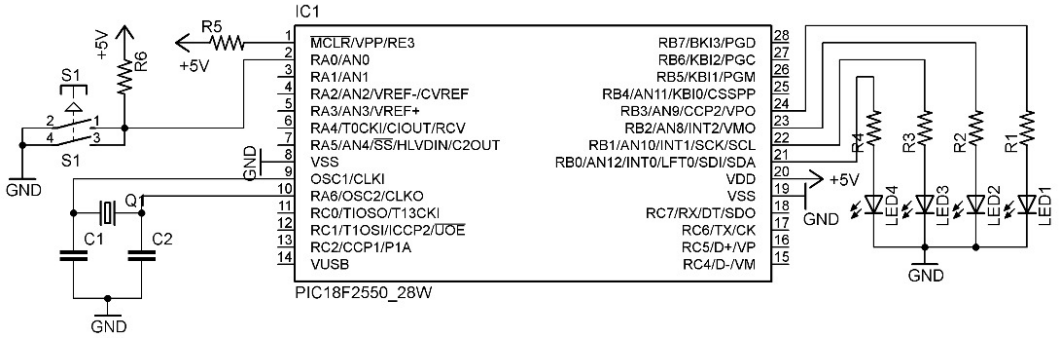
Donde Tcy es 4/Fosc, e i es un escalar que puede tomar valores entre 0 y 255.

2.10. Segundo ejemplo, Leds titilando

2.10.1. Objetivo

Hacer titilar 10 veces los leds del puerto B al accionar el pulsador.

2.10.2. Hardware



2.10.3. Codigo

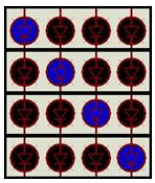
```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador  ** */
#pragma config FOSC=XT_XT,FCMEN=OFF,IESO=OFF,CPUDIV=OSC1_PLL2
#pragma config PWRT=ON,BOR=OFF,BORV=0,WDT=OFF,WDTPS=32768
#pragma config 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 k;        /* Variable utilizada para realizar efecto */

void main(void){

    ADCON1=0x0F;        /* Todos entradas/salidas digitales */
    TRISA=0xFF;         /* Todos como entradas */
    TRISB=0xF0;         /* Nible bajo como salida */

    LATB=0x00;          /* Todos los leds apagados */
    while(1){           /* Bucle infinito */
        if(PORTAbits.RA0==1){   /* Se testea estado del pulsador */
            LATB=0x00;  /* Si esta en 1 logico apagamos leds */
        }else{
            for(k=1;k<=10;k++){ /* Titila 10 veces */
                LATB=0x0F;                      /*  Enciende leds */
                Delay10KTCYx(30);       /* Demora 300ms */
                LATB=0x00;                      /* Apaga Leds */
                Delay10KTCYx(30);       /* Demora 300ms */
            }
        }
    }
}
```

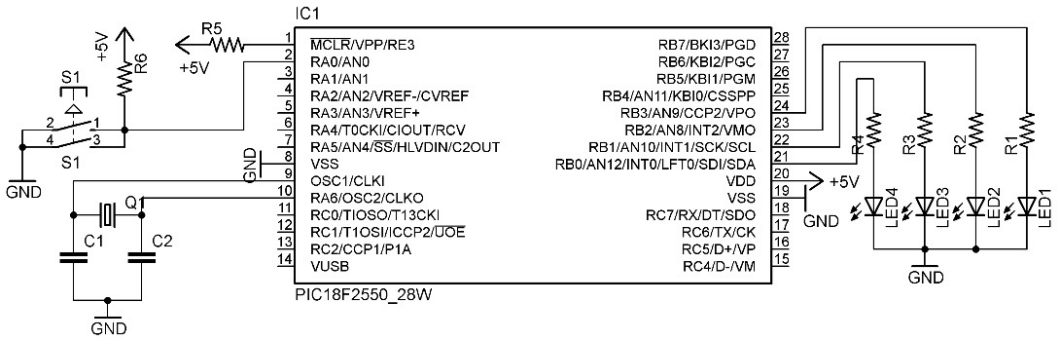
2.11. Tercer Ejemplo, Leds secuenciales

2.11.1. Objetivo

Al accionar el pulsador se realiza una secuencia de leds como se muestra en la figura:



2.11.2. Hardware



2.11.3. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.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 k;        /* Variable utilizada para realizar efecto */

void main(void){

    ADCON1=0x0F;        /* Todos entradas/salidas digitales */
    TRISA=0xFF;         /* Todos como entradas */
    TRISB=0xF0;         /* Nible bajo como salida */

    LATB=0x00;          /* Todos los leds apagados */
    while(1){           /* Bucle infinito */
        if(PORTAbits.RA0==1){   /* Se testea estado del pulsador */
            LATB=0x00;  /* Si esta en 1 logico apagamos leds */
        }else{
            LATB=0x01;                          /* Encedemos primer Led */
            for(k=1;k<=4;k++){  /* Rota 4 veces */
                Delay10KTCYx(30);       /* Demora 300ms */
                LATB=(PORTB<<1);        /* Rotamos Led encendido */
            }
        }
    }
}
```

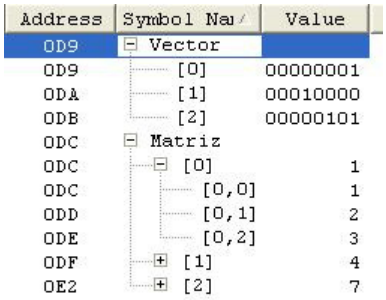
2.12. Arreglos de Variables

Nos permite trabajar con un conjunto de variables y acceder a cada una mediante un índice único que lo identifica. Todos los valores que contienen deben ser del mismo tipo.

```
unsigned char Vector[5];
unsigned char Matriz[3][3];
.
.
.

//Cargamos vector y matriz:
Vector[0]=156;
Matriz[1][1]=85;

//Leemos vector y matriz:
PORTB=Vector[4];
PORTB=Matriz[0][0];
```

En la declaracion se pueden pre cargar los valores de la siguiente forma:

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

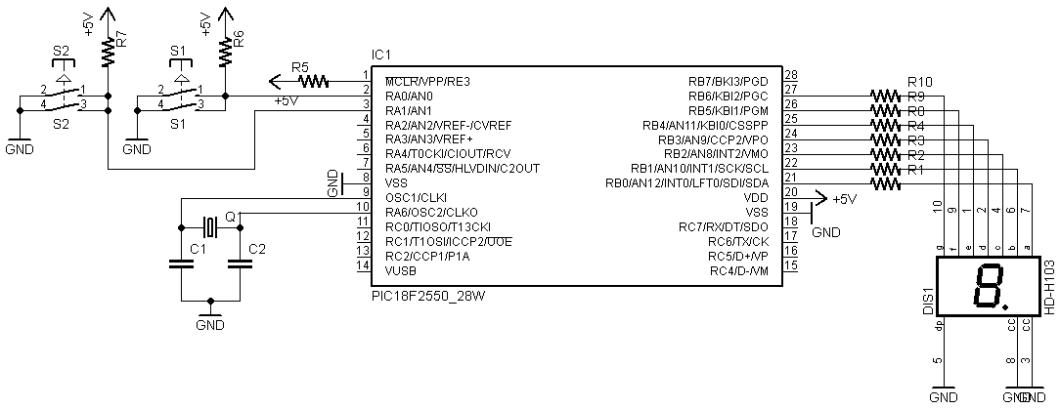


2.13. Cuarto Ejemplo, Control de display 7 segmentos

2.13.1. 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 será el caso en el que se presiona los dos pulsadores a la vez.

2.13.2. Hardware



2.13.3. 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

unsigned char i;        // Para controlar vizualizacion del Display.-
// Solo a modo de ejemplo ubicamos variables constantes en 0x400
#pragma romdata Display=0x400
const rom unsigned char Display7Seg[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66,
                                         0x6D, 0x7D, 0x07, 0xFF, 0x6F};
#pragma romdata

void main(void){
    ADCON1=0x0F;// Todos entrada/salida digitales.-
    TRISA=0xFF; // Todos como entrada.-
    TRISB=0X00; // Todos como salida.-

    LATB=0x3F; // 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);
        }
    }
}
```

Yendo a View/Wach y agregando la variable Display podemos visualizar que sea a asignado al espacio de memoria de programa ubicado entre 0x400 y 0x409.

Capítulo 3

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.

3.1. 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 este relacionado con la tarea que llevará a cabo.

Lista de parámetros : se enlista el tipo de dato y el nombre de cada parámetro. En caso de utilizar parámetros se deja el paréntesis vacio o se incluye la palabra void.

```
unsigned int Suma(unsigned char A, unsigned char B){
  unsigned int C;
  C=A+B;
  return(C);
}
void Envio_Data(unsigned char A){
        //Sentencia.-
```

```
}
void Muestras(void){
        //Sentencias.-
}
```

Dos formas de incluir una función en nuestro código:
Realizando la declaración en el encabezado y después la definición en cualquier sector del programa.

```
// Declaracion de la funcion
void Funcion(void);
char OtraFuncion(char,int)
.
.
.
void main(void){
        .
        .
.
// Llamo a la funcion.
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.

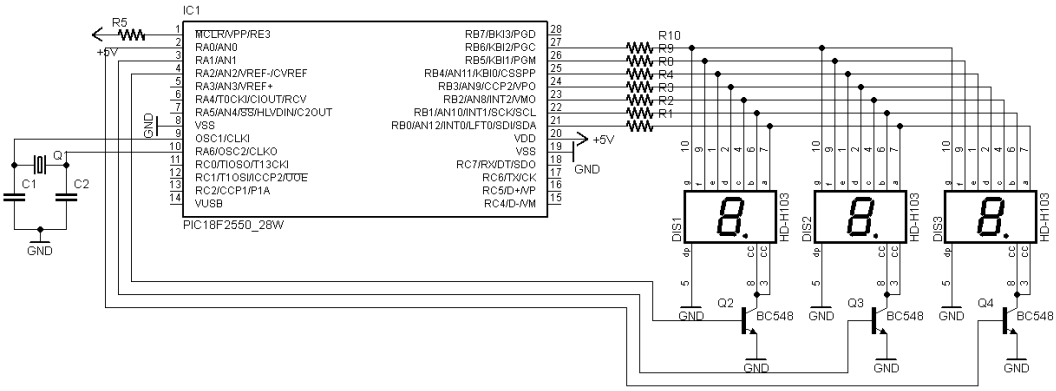
```
.
.
.
//Definicion de la funcion
void Funcion(void){
        // Sentencias
}
char OtraFuncion(char k,int p){
        // Sentencias
}


void main(void){
        .
        .
.
// Llamo a la funcion.
Funcion();
m=OtraFuncion(5,5430);
}
```

3.2. Ejemplo, Control de varios display, multiplexión de la señal

3.2.1. Objetivo

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

3.2.2. Hardware



3.2.3. 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]={0x3F, 0x06, 0x5B, 0x4F, 0x66,
                                         0x6D, 0x7D, 0x07, 0xFF, 0x6F};

/* ** Declaracion de funcion a utilizar */
void Visualizacion (void);

void main(void){
    ADCON1=0x0F;// Todos entrada/salida digitales.-
    TRISA=0xF0;
    TRISB=0x00; // Todos como salida.-
```

```
    LATA=0x00;  // 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;
            }
        }
    }
}

// Solo a modo ejemplo colocamos funcion en la direccion 0x1000..
// ..de la memoria de programa;
#pragma code Funcion=0x1000
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=4MHz)
        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.-
    }
}
#pragma code
```

Si abrimos la ventana Disassembly Listing (View/...) y buscamos la función, podemos notar en la primer columna que la misma se inicia en la dirección indicada.

3.3. 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.

3.3.1. 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 razón 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 implementación.

#define

La directiva *define* tiene dos tipos de uso, como si fuera un objecto o como si fuera una función. Las que se asemejan a funciones toman parámetros 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 implementación de esta directiva. Las que se asemejan a funciones, pueden tomar parámetros pero no se comportan como una función, 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 compliaciones condicionadas, por ejemplo para hacer una librería 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=0x100
    unsigned char Buffer[512];
    #pragma udata
#endif
// C30 y C32
#if defined (__C30__) || defined (__PIC32MX__)
    unsigned char __attribute__ ((aligned(4))) Buffer[512];
#endif
```

3.4. Ejemplo, Control de varios display, utilizando directivas de preprocesador

3.4.1. Objetivo

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

3.4.2. 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]={0x3F, 0x06, 0x5B, 0x4F, 0x66,
                                         0x6D, 0x7D, 0x07, 0xFF, 0x6F};

/* ** Declaracion de funcion a utilizar */
void Visualizacion (void);

void main(void){
    ADCON1=0x0F;// Todos entrada/salida digitales.-

    DISPLAY_TRIS_DATA=0x00:
    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=4MHz)
        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.-
    }
}
```

3.5. 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 envio/recepción de datos. El usuario debe proveer 3 delay para el correcto funcionamiento, DelayPORXLCD() de 15ms, DelayXLCD() de 5ms y DelayFor18TCY() de 18 Tcy.

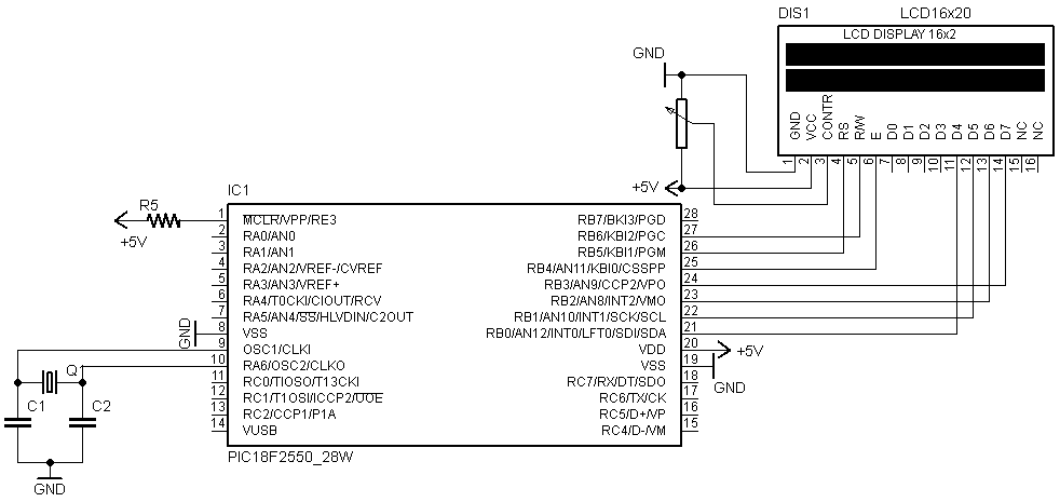
En este caso no vamos a modificar la librería (Véase más adelante), pues lo vamos a controlar con el puerto B, configuración por defecto, pero en el caso de que se modifique sugiero siempre respaldarla con una copia de seguridad.

3.5.1. Ejemplo, control de LCD

Objetivo

Vamos a escribir un simple mensaje en un LCD. Se crearán 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.

Hardware



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 = 0x40;
    else
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}

void main(void){

    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo
    comandXLCD(0x0C);                   // Encendemos LCD.-
    putrsXLCD("Probando LCD");
    gotoxyXLCD(1,2);                    //Pasamos al oriden del Linea 2.-
    putrsXLCD("Por Suky");
    while(1){                           // Bucle infinito.
    }
}
```

3.6. Conversión analógica digital

Para utilizar el módulo de conversión analógica-digital MPLAB C18 nos proporciona la librería adc.h la cual contiene funciones necesarias para configuración e implementación del mismo. Este módulo posee varias opciones de configuración que dependerán de la aplicación y para más información leer el datasheet del microcontrolador utilizado.

La librería al ser utilizable en todos los microcontroladores 18F agrupa las funciones dependiendo de sus caracteristicas, y para saber en cual entra el que estamos usando recurrimos al archivo *pconfig.h*, y por ejemplo para el PIC18F2550 le corresponde ADC_V5.

3.6.1. Funciones (para ADC_V5)

- *OpenADC(PARAM_SCLASS unsigned char,PARAM_SCLASS unsigned char,PARAM_SCLASS unsigned char);*

  Con ella se configure el reloj, el formato, tensión de referencia, puerto y canal de la conversión. Para saber que colocar en cada parámetro abrir AD Converter ubicado en *C:/MCC18/doc/periph-lib.*
  Ejemplos:

```
OpenADC(ADC_FOSC_32 & ADC_8_TAD & ADC_RIGHT_JUST,
        ADC_REF_VDD_VSS & ADC_INT_OFF, ADC_5ANA);
```

```
#define  USE_OR_MASKS
OpenADC(ADC_FOSC_RC | ADC_20_TAD| ADC_LEFT_JUST,
        ADC_REF_VREFPLUS_VREFMINUS |  ADC_INT_OFF, ADC_15ANA);
```

- *CloseADC();* Desactiva el conversor y la interrupción.
- *SetChanADC(Unsigned char);* Selecciona el canal que se va a utilizar.
- *ConvertADC();* Comienza la conversión.
- *SelChanConvADC(Unsigned char);* Selecciona canal y comienza conversión.
- *BusyADC();* Comprueba si la conversión a finalizado.
- *ReadADC();* devuelve la lectura del canal analógico seleccionado.

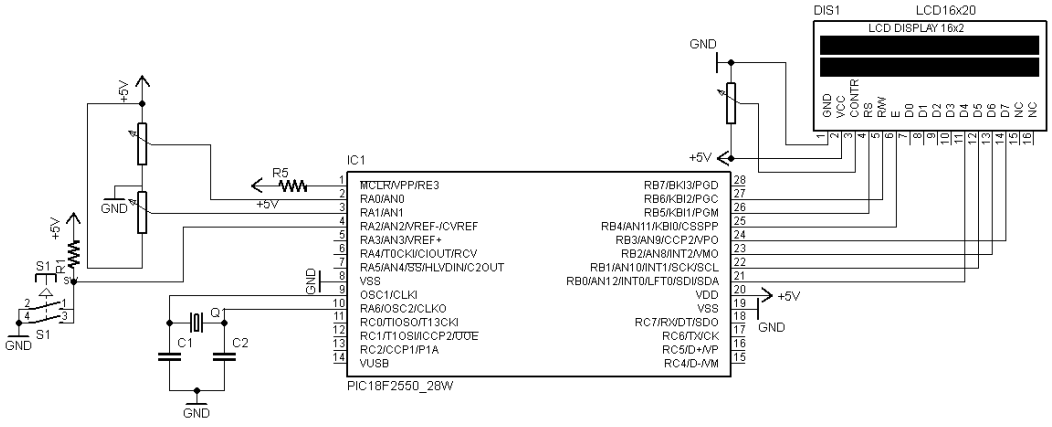
3.6.2. Ejemplo

Uso del módulo de conversión analógico/digital.

Objetivo

Tomar lectura de 2 valores analógicos y mostrarlo en un LCD. Al accionar un pulsador, leeremos los 2 canales analógicos y los mostraremos en el LCD durante 1 segundo.

Hardware



Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Includes ** */
#include <delays.h>
#include <xlcd.h>
#include <stdlib.h> //Libreria para conversiones a string
#include <adc.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 = 0x40;
    else
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}

void main(void){
    unsigned int Canal0, Canal1;
    char String[4];

    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo
    OpenADC(ADC_FOSC_RC & ADC_2_TAD & ADC_RIGHT_JUST,
            ADC_REF_VDD_VSS & ADC_INT_OFF, ADC_2ANA);
```

```
    comandXLCD(0x0C);                           // Encendemos LCD.-

    while(1){
        comandXLCD(0x01);       // Borra pantalla y vuelve al origen.-
        putrsXLCD("Presionar boton");
        while(PORTAbits.RA2==1){}// Espero a que se accione el pulsador.-
        SetChanADC(0);          // Selecciono canal a convertir.-
        //Delay10TCYx(1);// 10us para que se cargue el  capacitor sample&hold
                              //..(Solo cuando se selecciona ADC_0_TAD).-
        ConvertADC();                 // Comienza conversion.-
        while(BusyADC()==1){}     // Hasta que se finalice conversion.-
        Canal0= ReadADC();          // Realizo lectura.-

        SetChanADC(1);          // Selecciono canal a convertir.-
        //Delay10TCYx(1);// 10us para que se cargue el  capacitor sample&hold
                             // ..(Solo cuando se selecciona ADC_0_TAD).-
        ConvertADC();               // Comienza conversion.-
        while(BusyADC()==1){}   // Hasta que se finalice conversion.-
        Canal1= ReadADC();          // Realizo lectura.-

        comandXLCD(0x01);       // Borra pantalla y vuelve al origen.-
        putrsXLCD("Canal 0 = ");
        itoa(Canal0, String);           // Convertimos entero a string.-
        putsXLCD(String);
        gotoxyXLCD(1,2);
        putrsXLCD("Canal 1 = ");
        itoa(Canal1, String);           // Convertimos entero a string.-
        putsXLCD(String);
        Delay10KTCYx(100);
    }
}
```

Capítulo 4

Interrupciones

4.1. Introducción

Los dispositivos **PIC18** tienen múltiples fuentes de interrupción y la característica de prioridad de interrupción, que permite a cada fuente de interrupción asignarle un nivel de prioridad, bajo o alto. Cuando ocurre un evento de alta prioridad interrumpirá cualquier interrupción de baja prioridad que pueda estar en progreso. El vector de alta prioridad esta en 0x08 y el vector de baja prioridad en 0x18.

Cada fuente de interrupción tiene tres bits para controlar su operación. Las funciones de estos bits son:

- Bit bandera, que indica si un evento de interrupción ha ocurrido.
- Bit Enable, que admiten la ejecución de la interrupción, permitiendo la bifurcación del programa a la dirección del vector de interrupción.
- Bit de prioridad, para seleccionar prioridad baja o alta.

La característica de prioridad de interrupciones se activa seteando el bit **IPEN**. Si este no esta seteado, no existen prioridades y las interrupciones se comportan como en los dispositivos de gama media (**PIC16**) y todas las interrupciones se bifurcan al vector **0x08**.

4.2. Rutinas de atención a interrupciones

La directiva #***pragma interruplow nombre*** define rutina de servicio de interrupción (**ISR**) de baja prioridad y #p***ragma interrup nombre*** de alta prioridad.
Las ISR son funciones como cualquier otra, pero con las restricciones de que:

- No devuelven ni aceptan parámetros.
- No se puede invocar desde otros puntos del programa.
- Las variables globales que utilice se deben declarar como volatile.

El **Compilador C18** no sitúa automáticamente las **ISR** en las posiciones de los vectores de interrupción, por lo que el usuario debe ubicarlas.

4.2.1. Ejemplos

```
#pragma interruplow ISRBajaPrioridad
void ISRBajaPrioridad(void){
    // Tratamiento de interrupcion.-
}

// Creamos una nueva seccion de codigo a partir de la direccion 0x18.-
#pragma code PrioridadBaja = 0x18
Void VectorBajaPrioridad(void){
    // Instruccion insertada en la direccion 0x18.-
    _asm goto ISRBajaPrioridad _endasm
}
#pragma code // Cerramos seccion.-

#pragma interruplow ISRAltaPrioridad
void ISRAltaPrioridad(void){
    // Tratamiento de interrupcion.-
}

// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code PrioridadAlta = 0x08
Void VectorAltaPrioridad(void){
    // Instruccion insertada en la direccion 0x08.-
    _asm goto ISRAltaPrioridad _endasm
}
#pragma code // Cerramos seccion.-
```

Las rutinas de servicio de interrupción solo guardan el mínimo del contexto por defecto, y para guardar otras variables o zonas de memoria (declaradas en *.lkr) hay que indicarlos mediante la opción ***save=***
Por ejemplo para guardar el contenido de una variable global:

```
#pragma interrupt ISR_low save= VariableGlobal
```

También se pueden guardar registros internos o secciones completas de datos. Para salvar toda una sección de datos, por ejemplo sección *MISECTION* hacemos:

```
/* ** Archivo *.lkr  ** */
DATABANK   NAME=gpr1_      START=0x1F0                    END=0x1FF  PROTECTED
SECTION    NAME=MISECTION  RAM=gpr1_
```

```
// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion save=section("MISECTION")
void ISRRecepcion(void){
    if(PIR1bits.RCIF==1){
```

```
// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion
void ISRRecepcion(void){
MOVFF 0xfda, 0xfe4
NOP
MOVFF 0xfe2, 0xfda
NOP
MOVFF 0xfe9, 0xfe4
NOP
MOVFF 0xfea, 0xfe4
NOP
MOVFF 0xff6, 0xfe4
NOP
MOVFF 0xff7, 0xfe4
NOP
MOVFF 0xff5, 0xfe4
NOP
MOVFF 0xff3, 0xfe4
NOP
MOVFF 0xff4, 0xfe4
NOP
MOVFF 0xffa, 0xfe4
NOP
LFSR 0, 0
NOP
MOVLW 0x2
DECF 0xfe8, W, ACCESS
BNC 0x46
MOVFF 0xfee, 0xfe4
NOP
BRA 0x3c
LFSR 0, 0x2
NOP
MOVLW 0
DECF 0xfe8, W, ACCESS
BNC 0x56
MOVFF 0xfee, 0xfe4
NOP
BRA 0x4c
MOVF 0xfe6, F, ACCESS
    if(PIR1bits.RCIF==1){
BTFSS 0xf9e, 0x5, ACCESS
```

```
// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion save=section("MISECTION")
void ISRRecepcion(void){
MOVFF 0xfda, 0xfe4
NOP
MOVFF 0xfe2, 0xfda
NOP
MOVFF 0xfe9, 0xfe4
NOP
MOVFF 0xfea, 0xfe4
NOP
MOVFF 0xff6, 0xfe4
NOP
MOVFF 0xff7, 0xfe4
NOP
MOVFF 0xff5, 0xfe4
NOP
MOVFF 0xff3, 0xfe4
NOP
MOVFF 0xff4, 0xfe4
NOP
MOVFF 0xffa, 0xfe4
NOP
LFSR 0, 0
NOP
MOVLW 0x2
DECF 0xfe8, W, ACCESS
BNC 0x46
MOVFF 0xfee, 0xfe4
NOP
BRA 0x3c
LFSR 0, 0x2
NOP
MOVLW 0
DECF 0xfe8, W, ACCESS
BNC 0x56
MOVFF 0xfee, 0xfe4
NOP
BRA 0x4c
LFSR 0, 0x1f0
NOP
MOVLW 0x1
DECF 0xfe8, W, ACCESS
BNC 0x66
MOVFF 0xfee, 0xfe4
NOP
BRA 0x5c
MOVF 0xfe6, F, ACCESS
    if(PIR1bits.RCIF==1){
BTFSS 0xf9e, 0x5, ACCESS
```

4.2.2. Precauciones al trabajar con interrupciones

- Si una ISR llama a otra subrutina se debe guardar la sección de datos temporales utilizadas por las funciones *".tmpdata". save=section(".tmpdata")*
- Si una ISR llama a otra subrutina que devuelve un parámetro de 16 bits, el registro interno del microcontrolador *PROD* debe de guardarse. *save=PROD*
- Si utiliza la librería matemática o llama a funciones que devuelven valores de 24 o 32 bits, se debe guardar la sección de datos *"MATH_DATA". save=section("MATH_DATA")*

4.3. Módulo USART

Para la comunicación serial es necesario agregar la librería usart.h. Con esta librería se configura el modo de transmisión y recepción serie de nuestro microcontrolador.

4.3.1. Funciones

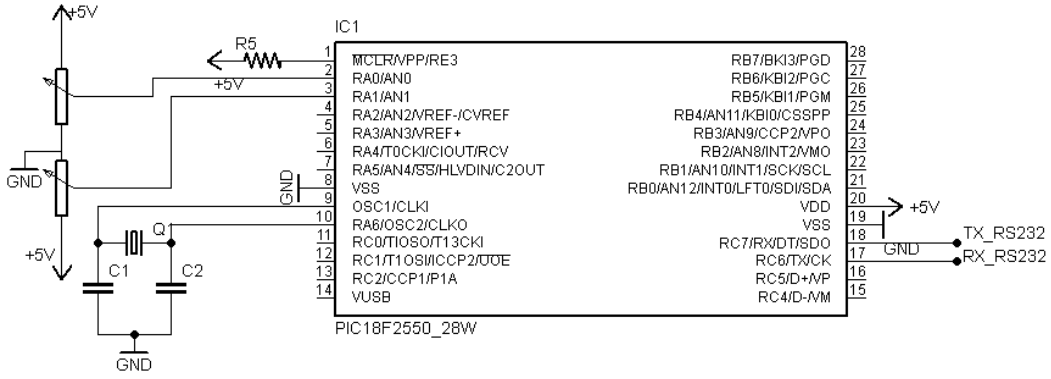
- *OpenUSART(unsigned char config, unsigned int spbrg);* Esta función corresponde a la configuración del módulo USART, asincrónica o sincrónica, 8 bits o 9 bits, velocidad de comunicación, etc. Para saber que colocar en cada parámetro abrir USART ubicado en *C:/MCC18/doc/periph-lib.*
- *CloseUSART();* Deshabilita módulo USART.
- *putcUSART(char data);* Envía un byte.
- *putsUSART(char *buffer);* Envia un string desde la memoria de datos.
- *putrsUSART(const rom char *data);* Envia un string desde la memoria de programa.
- *BusyUSART();* Determina si se ha transmitido el dato.
- *DataRdyUSART();* Indica si ha llegado un dato para ser leído.
- *getcUSART();* Lee un byte.
- *getsUSART(char *buffer, Unsigned char len);* Lee un string.

4.3.2. Ejemplo

Objetivo

La PC enviará comando de lectura de los canales analógicos. Si se recibe un 0x61 ('a'), enviar canal 0, si se recibe un 0x62 ('b') enviar canal 1 y si se recibe un 0x63 ('c') enviar los 2 canales analógicos.

Hardware



Nota: La adaptación de tensiones se puede hacer con el MAX232 o con transistores, como se desee.-

Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Includes ** */
#include <delays.h>
#include <stdlib.h> //Libreria para conversiones a string
#include <adc.h>
#include <usart.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 ISRRecepcion(void);
volatile char Data, Kbhit;

// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRRecepcion _endasm
}
#pragma code // Cerramos seccion.-

// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion
void ISRRecepcion(void){
    if(PIR1bits.RCIF==1){
        Data=getcUSART();       // Leemos Dato recibido
```

```
        Kbhit=1;                        // Indicamos que se ha recibido un dato.-
        PIR1bits.RCIF=0;        // Borramos bandera.-
    }
}


void main(void){
    unsigned int Canal0, Canal1;
    char String[4];

    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON &
              USART_ASYNCH_MODE & USART_EIGHT_BIT &
              USART_CONT_RX & USART_BRGH_HIGH,25); // 9600,8,n,1
    OpenADC(ADC_FOSC_RC & ADC_2_TAD & ADC_RIGHT_JUST,
            ADC_REF_VDD_VSS & ADC_INT_OFF, ADC_2ANA);
    Kbhit=0;
    RCONbits.IPEN=0;    // Deshabilitamos Prioridades
    INTCONbits.PEIE=1;  // Habilitamos interrupcion de perifericos.-
    INTCONbits.GIE=1;   // Habilitamos interrupcion Global.

    putrsUSART("Prueba Comunicacion Serial \r");
    while(1){
        while(Kbhit==0){}; // esperamos a recibir dato.-
        Kbhit=0;
        switch(Data){
            case 0x61:  // letra a
                SetChanADC(0);  // Selecciono canal a convertir.-
                //Delay10TCYx(1);//10us para que se cargue el  capacitor sample&hold
                ConvertADC();           // Comienza conversion.-
                while(BusyADC()==1){}   // Hasta que se finalice conversion.-
                Canal0= ReadADC();      // Realizo lectura.-
                putrsUSART("Canal 0 = ");
                itoa(Canal0, String);           // Convertimos entero a string.-
                putsUSART(String);
                putrsUSART("\r");
            break;
            case 0x62:  // letra b
                SetChanADC(1);  // Selecciono canal a convertir.-
                //Delay10TCYx(1);//10us para que se cargue el  capacitor sample&hold
                ConvertADC();           // Comienza conversion.-
            while(BusyADC()==1){}       // Hasta que se finalice conversion.-
                Canal1= ReadADC();      // Realizo lectura.-
                putrsUSART("Canal 1 = ");
                itoa(Canal1, String);           // Convertimos entero a string.-
                putsUSART(String);
                putrsUSART("\r");
            break;
            case 0x63:  // letra c
                SetChanADC(0);  // Selecciono canal a convertir.-
                //Delay10TCYx(1);//10us para que se cargue el  capacitor sample&hold
                ConvertADC();           // Comienza conversion.-
                while(BusyADC()==1){}   // Hasta que se finalice conversion.-
                Canal0= ReadADC();      // Realizo lectura.-

                SetChanADC(1);  // Selecciono canal a convertir.-
                //Delay10TCYx(1);//10us para que se cargue el  capacitor sample&hold
                ConvertADC();           // Comienza conversion.-
```

```
            while(BusyADC()==1){}       // Hasta que se finalice conversion.-
                Canal1= ReadADC();      // Realizo lectura.-

                putrsUSART("Canal 0 = ");
                itoa(Canal0, String);           // Convertimos entero a string.-
                putsUSART(String);
                putrsUSART("\r");
                putrsUSART("Canal 1 = ");
                itoa(Canal1, String);           // Convertimos entero a string.-
                putsUSART(String);
                putrsUSART("\r");
            break;
        }
    }
}
```

Utilizando Printf

Otra manera de enviar los datos es utilizando la función **printf**, la cual crea una cadena de texto con formato que puede ser enviada por ejemplo por el puerto serial. Más adelante veremos como seleccionar por que periférico enviar la cadena (Serial (por defecto), LCD, SPI, ect)

Para utilizar esta función debemos agregar la librería *stdio.h.*

```
printf (const rom char *fmt, ...);
```

Nos permite especificar el formato en que queremos mostrar datos por pantalla.

Parámetros: Cadena de formato (cómo se visualizan los datos) y Lista de valores (datos que se visualizan)
Formatos:
%d Entero con signo.
%u Entero sin signo.
%x Hexadecimal minúscula.
%X Hexadecimal mayúscula.

Ejemplo:

```
printf("Hola");
Data=10;
printf("El valor es : %u",Data); // Se muestra "El valor es : 10"
```

Ejemplo de formatos:

Especificación	Valor=0x12	Valor=0xFE
%03u	018	254
%u	18	254
%2u	18	*
%d	18	-2
%x	12	fe
%X	12	FE

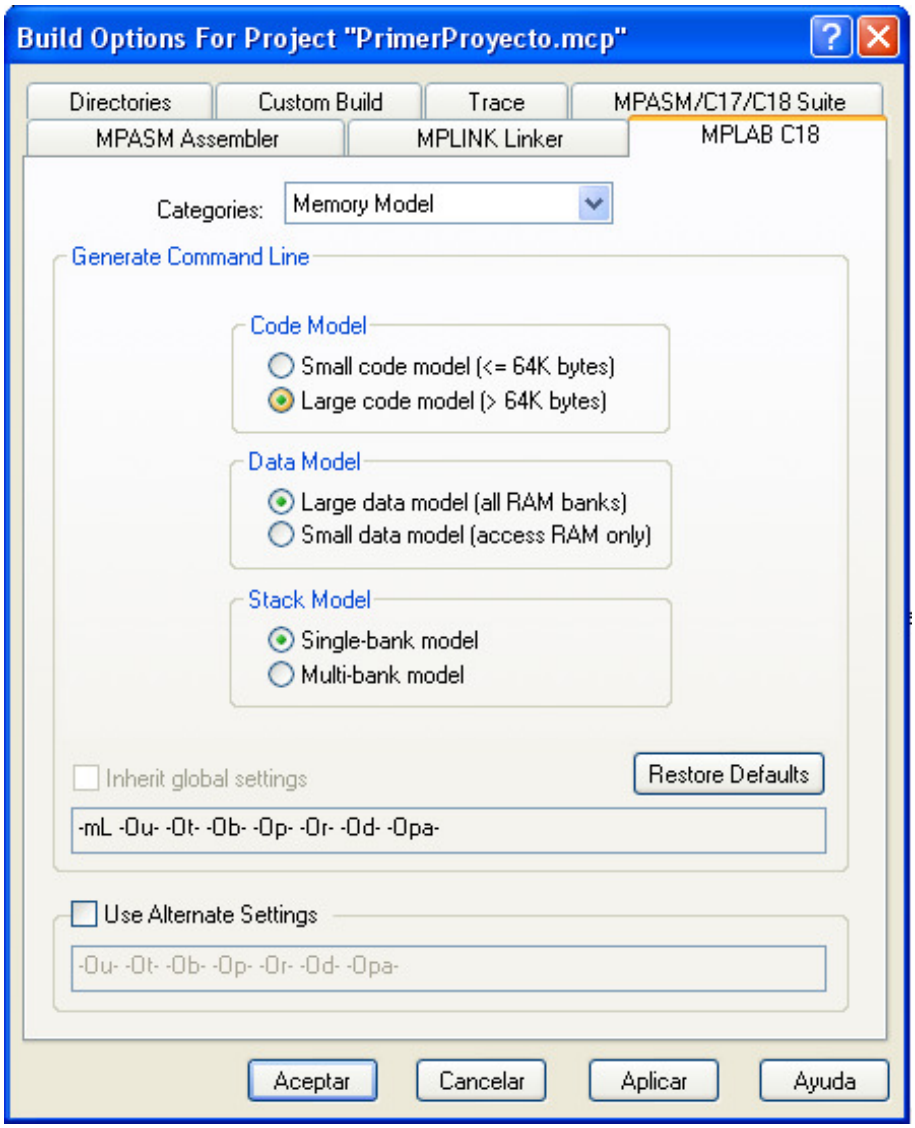
Entonces se puede reemplazar:

```
Canal0=ReadADC();
putrsUSART("Canal 0 = ");
itoa(Canal0, String);
putsUSART(String);
putrsUSART("\r");
```

Por:

```
#include <stdio.h>
.
.
.
Canal0=ReadADC();
printf("Canal 0 = %u \r", Canal0);
```

En el manual se usuario del compilador C18 indica que cuando se utiliza esta librería se debe modificar las opciones del poyecto, seleccionando large code model. Para ello vamos a Project/Build Options/Project, seleccionamos MPLAB C18, categoría Memory Model y chekeamos Large code model (¿64K bytes).


Build Options For Project "PrimerProyecto.mcp"
Directories
Custom Build
Trace
MPASM/C17/C18 Suite
MPASM Assembler
MPLINK Linker
MPLAB C18
Categories:
Memory Model
Generate Command Line
Code Model
Small code model (<= 64K bytes)
Large code model (> 64K bytes)
Data Model
Large data model (all RAM banks)
Small data model (access RAM only)
Stack Model
Single-bank model
Multi-bank model
Inherit global settings
Restore Defaults
-mL -Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-
Use Alternate Settings
-Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-
Aceptar
Cancelar
Aplicar
Ayuda

Capítulo 5

Librerías

5.1. Introducción

En el capítulo Comenzando con C18 mostramos como era el flujo para la generación del *.hex. Allí vimos que podemos disponer de archivos fuentes en *.c, archivos en *.asm, archivos objetos pre-compilados *.o, y librerías *.lib. En todos los ejemplos anteriormente mostrados solo hemos trabajado con un archivo principal *.c y las librerías pre-compiladas agrupadas en archivos *.lib, los cuales están ubicados en el subdirectorio MCC18/lib.
La librería estándar es ***clib.lib*** (para modo no-extendido) o ***clib_e.lib*** (para modo extendido). Esta librería es independiente de microcontrolador y el código fuente se encuentra en los directorios:

- src/traditional/math
- src/extended/math
- src/traditional/delays
- src/extended/delays
- src/traditional/stdclib
- src/extended/stdclib

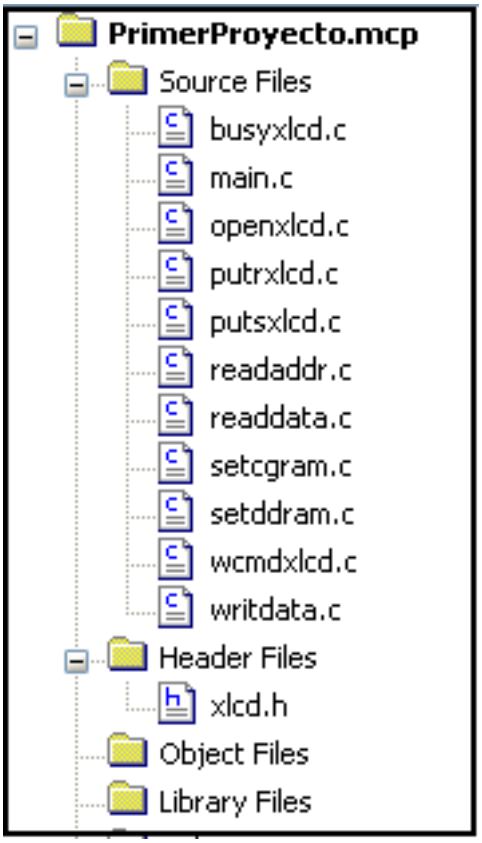
La librería específica que contiene funciones escritas para un miembro individual de la familia 18. El nombre de estas librerías depende del microcontrolador **p*processor*.lib** (No-extendido) y **p*processor_e*.lib** (Extendido). Ejemplo *p18f2550.lib*. El código fuente se encuentra en los directorios:

- src/pmc_common
- src/traditional/proc
- src/extended/proc

En este nuevo capitulo vamos a mostrar los pasos a seguir para poder modificar una librería del Compilador C18 y como desarrollar una librería.

5.2. Modificar las librerías

Se explicará como realizar cambios en las librerías de C18, para que estas se hagan efectivas a la hora de compilar. En este caso cambiaremos el Puerto para el control del LCD.
Primero que nada se debe realizar una copia de seguridad, en este caso de xlcd.h. Luego en nuestro programa debemos incluir a xlcd.h (MCC18/h) en Header Files, y los archivos fuentes de esta librería (MCC18/src/pmc_common/XLCD), quedando de la siguiente manera:

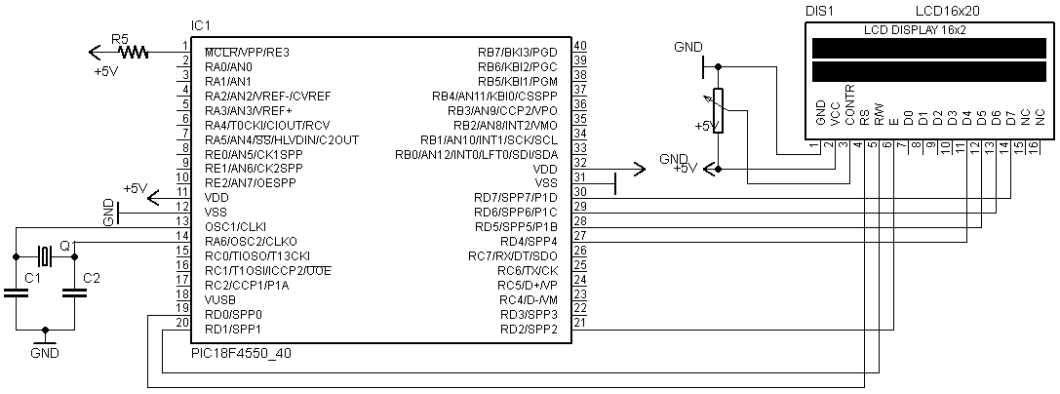


De esa forma se recompilan los archivos, creando sus respectivos archivos objetos (*.o) actualizados necesarios para el linkeo (MPLink) y así efectivizar los cambios realizados en las rutinas.

En los próximos dos ejemplos utilizaremos el hermano mayor, el *PIC18F4550*, dado que necesitamos un puerto completo para control del LCD(En el puerto C, RC3 no existe, RC4 y RC5 solo pueden ser entradas digitales). Vamos a modificar xlcd.h para que utilizar el Puerto D para controlar el LCD y los 4 bits más significativos del mismo (UPPER) para enviar los datos al LCD quedando de las siguiente manera:

```
#define UPPER
/* DATA_PORT defines the port to which the LCD data lines are connected */
```

```
#define DATA_PORT      PORTD
#define TRIS_DATA_PORT TRISD
/* CTRL_PORT defines the port where the control lines are connected.
* These are just samples, change to match your application.
*/
#define RW_PIN    LATDbits.LATD1    /* PORT for RW */
#define TRIS_RW   TRISDbits.TRISD1    /* TRIS for RW */
#define RS_PIN    LATDbits.LATD0    /* PORT for RS */
#define TRIS_RS   TRISDbits.TRISD0    /* TRIS for RS */
#define E_PIN     LATDbits.LATD2    /* PORT for D  */
#define TRIS_E    TRISDbits.TRISD2    /* TRIS for E  */
```



El código del programa queda:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f4550.h>
#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);
}
```

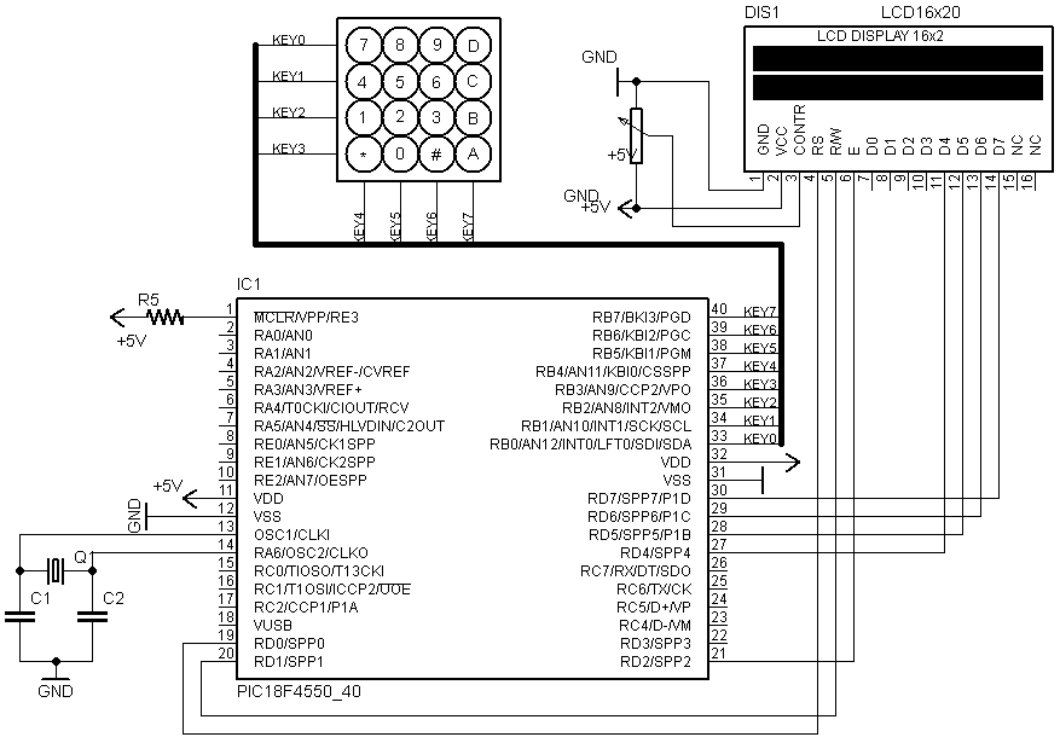
```
// Envia comando al LCD
void comandXLCD(unsigned char a){
    BusyXLCD();
    WriteCmdXLCD(a);
}
// Ubica cursor en (x = Posicion en linea, y = n&#186; de linea)
void gotoxyXLCD(unsigned char x, unsigned char y){
unsigned char direccion;

    if(y != 1)
        direccion = 0x40;
    else
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}

void main(void){

    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo
    comandXLCD(0x0C);                   // Encendemos LCD.-
    putrsXLCD("LCD en Puerto C");
    gotoxyXLCD(1,2);                    //Pasamos al oriden del Linea 2.-
    putrsXLCD("Por Suky");
    while(1){                           // Bucle infinito.
    }
}
```

5.3. Interrupción por cambio de estado RB4-RB7. Control de Teclado Matricial

5.3.1. Objetivo

Al accionar una tecla del Teclado Matricial, esta se mostrará en la pantalla LCD. Aprovecharemos la interrupción por cambio de estado RB4-RB7 para detectar que se ha presionado una tecla. Dentro de la interrupción se realiza el rastreo fino para determinar cual ha sido.

5.3.2. Hardware



Nota: Se usa el puerto C para el control del LCD, y para que la compilación sea correcta se deben seguir los pasos de la sección anterior.

5.3.3. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f4550.h>
#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


unsigned char Tecla;
const rom unsigned char Teclas[17]={'7','8','9','/',
```

```
                                    '4','5','6','x',
                                    '1','2','3','-',
                                    '.','0','=','+',''};

void ISRRB4_RB7(void);
void DelayFor18TCY(void);
void DelayPORXLCD(void);
void DelayXLCD(void);
void comandXLCD(unsigned char a);
void gotoxyXLCD(unsigned char x, unsigned char y);
char TestTeclado(void);

// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRRB4_RB7 _endasm
}
#pragma code // Cerramos seccion.-

//*************************************************************************
void main(void){
    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo

    TRISB=0xF0; //eeeessss.-
    PORTB=0x00;
    INTCON2 = 0X80; // Habilitamos resistencias pull up.
    INTCON = 0X88;//Configuracion de interrupcion por cambio de estado RB4-RB7

    comandXLCD(0x0C);                   // Encendemos LCD.-
    putrsXLCD("TeclaPulsada \" \"");
    gotoxyXLCD(2,2);                    //Pasamos al oriden del Linea 2.-
    putrsXLCD("<<Autor Suky>>");
    while(1){                           // Bucle infinito.
    }
}

//*************************************************************************
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 = 0x40;
    else
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}


/* Rastrea Tecla presionada, para ello, se va colocando a 0 logico las filas
  de a una por vez, y se testea si alguna columna tambien lo esta. Se utiliza
  la variable i para indicar que tecla esta siendo testeada, y de esta manera
  al detectar la pulsacion usar esta variable para seleccionar el caracter en
  el Buffer Teclas.
*/
char TestTeclado(void){
    unsigned char i,j;
    PORTB=0x0E; //xxxx1110.
    i=0;
    for(j=0;j<4;j++){
        if(PORTBbits.RB4==0){break;}
        i++;
        if(PORTBbits.RB5==0){break;}
        i++;
        if(PORTBbits.RB6==0){break;}
        i++;
        if(PORTBbits.RB7==0){break;}
        i++;
        LATB=PORTB<<1;  //Trasladamos el 0 a siguiente Pin.
    }
    return(Teclas[i]);
}

//*****************************************************************
// Rutina de Interrupcion.-
#pragma interrupt ISRRB4_RB7
void ISRRB4_RB7(void){
    if(INTCONbits.RBIF==1){
        //Realizamos test del Teclado, y nos devuelve tecla pulsada
        Tecla=TestTeclado();
        gotoxyXLCD(15,1);
        putcXLCD(Tecla);        // Visualizamos tecla pulsada.
        //Esperamos a que se suelte.-
        while(PORTBbits.RB4==0 ||PORTBbits.RB5==0 ||
              PORTBbits.RB6==0 ||PORTBbits.RB7==0){}
        PORTB=0x00;
        INTCONbits.RBIF = 0; //Borramos bandera
    }
}
```

5.4. Utilización de Printf para escribir en LCD

Para poder utilizar printf para formatear variables generando un string y enviarlo por ejemplo al LCD hay que realizar un par de modificaciones sencillas. Compiler C18 para seleccionar la

función que utilizará para enviar datos a través de printf define una variable llamada ***stdout** en el archivo *stdout.c*. Dependiendo del valor que posea seleccionará entre enviar datos por el puerto seríal (valor default) o utilizar la función que el usuario decida modificando el archivo *_user_putc.c*.
Entonces debemos agregar al proyecto el archivo *_user_putc.c* ubicado en *../MCC18/src/extended/stdclib* y a él hacerle las siguientes modificaciones:

```
extern putcXLCD(unsigned char c);

void
_user_putc (unsigned char c)
{
        putcXLCD(c);
}
```

Con esto ya podemos, modificando el valor de stdout, seleccionar entre escribir en el LCD o enviar por el puerto serial al utilizar la función printf.
Veamos un ejemplo:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
#include <delays.h>
#include <usart.h>
#include <xlcd.h>
#include <stdio.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 = 0x40;
    else
```

```
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}

void main(void){
    char Buffer[12]={"2 linea..."};
    unsigned char Variable=125;

    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo
    comandXLCD(0x0C);
    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
         USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH,25); // 9600,8,n,1

    // Modificamos para utilizar funci\'on del usuario, o sea escribir en LCD.-
    stdout =_H_USER;
    // Convertimos a string variable.-
    printf("Var= %u",Variable);
    gotoxyXLCD(1,2);
    // Enviamos string ccntenido en buffer.-
    printf("%s",Buffer);

    // Modificamos para utilizar nuevamente puerto serial.
    stdout =_H_USART;
    printf("Var= %u",Variable);

    while(1){}
}
```

5.5. Como crear una librería

Vamos a crear una librería para el control del circuito integrado ds1302, el cual es un reloj de tiempo real con interface de comunicación serial que nos permite leer o escribir registros internos que contienen los segundos, minutos, horas, día de la semana, fecha, mes y año.
Para tener amplio conocimiento de cómo funciona y como poder realizar la comunicación con él, como se ha venido recomendado, es necesario leer el datasheet. A groso modo se puede dar la idea de que necesitamos realizar funciones para leer y escribir los registros internos del dispositivo, y habrá que leer si se necesita inicializarlo y de que modo se reciben los datos. Una vez entendido el funcionamiento se pasa al proceso de realizar la librería.

Un proceso adecuado en el armado de una librería es realizar dos archivos, un archivo cabecera (*.h) que contendrá definiciones para el funcionamiento (Pines para controlar el dispositivo y otros para mejor entendimiento del código) y la declaración de las funciones, y un archivo fuente con la definición de las funciones y variables.
Para agilizar el proceso de compilación se realiza lo siguiente:

```
#ifndef __ds1302_H
#define __ds1302_H
        //Sentencias
#endif
```

De esta manera solo se compila una sola vez la librería en el caso de ser invocada varias veces en el proceso de compilación.

El archivo cabecera queda de la siguiente manera:

```
/** \file ds1302.h
* \brief Este fichero contiene las declaraciones de los pines y
*funciones utilizadas para el control del DS1302.
*/
#ifndef __ds1302_H
#define __ds1302_H

#include <p18cxxx.h>

//Definicion de los pines a utilizar.
#define SCLK PORTCbits.RC0              /**< Pin de Clock del DS1302*/
#define IO       PORTCbits.RC1          /**< Pin de Data del DS1302*/
#define RST      PORTCbits.RC2          /**< Pin de Enable del DS1302*/
// Definicion de los bits del Tris para configurar en modo escritura o lectura.
#define TrisSCLK        TRISCbits.TRISC0        /**< Tris del pin Clock*/
#define TrisIO          TRISCbits.TRISC1        /**< Tris del pin Data*/
#define TrisRST         TRISCbits.TRISC2        /**< Tris del pin Enable*/

/** \brief Funcion que envia el byte por el Pin seleccionado.
*
* \param Byte Valor a enviar al DS1302.
*/
void write_byte( unsigned char Byte);
/** \brief Envia los Datos al DS1302 segun el protocolo, un primer
* Byte de comando y el segundo de Data.
 *
* \param Byte Byte de comando.
* \param Data Byte de Dato.
*/
void write_ds1302( unsigned char Byte, unsigned char Data);
/** \brief Funcion que realiza lectura de 1 byte del DS1302, enviado
* un Byte de comando y luego realizando la lectura.
 *
* \param Byte Valor de Comando.
* \return Byte leido del DS1302.
*/
unsigned char read_ds1302( unsigned char Byte);
/** \brief Inicializacion del DS1302
*/
void ds1302_init(void);
/** \brief Conversion del valor decimal a enviar al DS1302.
*
* Por ejemplo, deseo enviar 36=0x24. Entonces, get_bcd(36) retorna 0x36.-
*
* \param Data Valor a convertir.
*\return Byte obtenido de la conversion.
*/
unsigned char get_bcd( unsigned char Data);
/** \brief Conversion del valor leido del DS1302 al decimal.
 *
```

```
* Por ejemplo, se lee 0x36, entonces al realizar rm_bcd(0x36) retorna 0x24=36.-
*
* \param Data Valor obtenido de la lectura.
* \retun Byte obtenido de la conversion.
*/
unsigned char rm_bcd( unsigned char Data);
/** \brief Funcion que carga al DS1302 valores de Hora y Fecha
*    seleccionados por el usuario.
*
* \param day Dia de la semana, valor entre 1 y 7.
* \param mth Mes, valor entre 1 y 12.
* \param year A\~no, valor entre 00 y 99.
* \param dow Fecha, valor entre 1 y 31
* \param hr Hora, Segun como se envie el dato puede ser entre
* 1 y 12 o 0 y 23. El ultimo nuestro caso.
* \param min Minutos, valor entre 0 y 59.
* \param sec Segundos, directamente se envia 0.
*/
void set_datetime( unsigned char day, unsigned char mth,
                                        unsigned char year, unsigned char dow,
                                        unsigned char hr, unsigned min);

/** \brief Funciones que realizan la lectura del DS1302 segun el dato pedido.
*/
unsigned char get_day();
unsigned char get_mth();
unsigned char get_year();
unsigned char get_dow();
unsigned char get_hr();
unsigned char get_min();
unsigned char get_sec();

/** \brief Demora que se necesita realizar en nuestro main, que
* dependera del oscilador que se este usando.
*/
extern void Delay_2us(void);
#endif
```

Luego queda crear el código de nuestra librería, donde al principio se debe invocar el archivo cabecera *.h que contiene las definiciones:

```
#include "ds1302.h"

void write_byte(unsigned char Byte){
   unsigned char i;
   TrisIO=0;
   for(i=0;i<=7;++i){
      IO=Byte&0x01;
      Byte>>=1;
      SCLK=1;
      SCLK=0;
   }
}

void write_ds1302(unsigned char Byte, unsigned char Data){

   RST=1;
```

```
   write_byte(Byte);
   write_byte(Data);
   RST=0;
}

unsigned char read_ds1302(unsigned char Byte){
   unsigned char i,Data;

   TrisSCLK=0;
   TrisRST=0;
   RST=1;
   write_byte(Byte);
   TrisIO=1;
   Nop();
   Data=0;
   for(i=0;i<=6;i++){
      if(IO==1){Data+=0x80;}
      Data>>=1;
      SCLK=1;
      Delay_2us(); //2us
      SCLK=0;
      Delay_2us(); //2us
   }
   RST=0;
   return(Data);
}

void ds1302_init(void){
   unsigned char j;

   TrisSCLK=0;
   TrisIO=0;
   TrisRST=0;

   RST=0;
   Delay_2us(); //2us
   SCLK=0;
   write_ds1302(0x8E,0);
   write_ds1302(0x90,0xA4);
   j=read_ds1302(0x81);
   if((j & 0x80)!=0){
      write_ds1302(0x80,0);
   }
}

unsigned char get_bcd(unsigned char Data){
   unsigned char NibleH,NibleL;
   NibleH=Data/10;
   NibleL=Data-(NibleH*10);
   NibleH<<=4;
   return(NibleH|NibleL);
}

unsigned char rm_bcd(unsigned char Data){
   unsigned char i;
   i=Data;
   Data=(i>>4)*10;
```

```
    Data=Data+(i<<4>>4);
    return(Data);
}

void set_datetime(unsigned char day, unsigned char mth,
          unsigned char year, unsigned char dow,
          unsigned char hr, unsigned min){
    TrisSCLK=0;
    TrisRST=0;

    write_ds1302(0x86,get_bcd(day));
    write_ds1302(0x88,get_bcd(mth));
    write_ds1302(0x8C,get_bcd(year));
    write_ds1302(0x8A,get_bcd(dow));
    write_ds1302(0x84,get_bcd(hr));
    write_ds1302(0x82,get_bcd(min));
    write_ds1302(0x80,get_bcd(0));
}

unsigned char get_day() {
         return(rm_bcd(read_ds1302(0x87)));
}
unsigned char get_mth(){
    return(rm_bcd(read_ds1302(0x89)));
}
unsigned char get_year(){
    return(rm_bcd(read_ds1302(0x8D)));
}
unsigned char get_dow(){
    return(rm_bcd(read_ds1302(0x8B)));
}
unsigned char get_hr(){
    return(rm_bcd(read_ds1302(0x85)));
}
unsigned char get_min(){
    return(rm_bcd(read_ds1302(0x83)));
}
unsigned char get_sec(){
    return(rm_bcd(read_ds1302(0x81)));
}
```

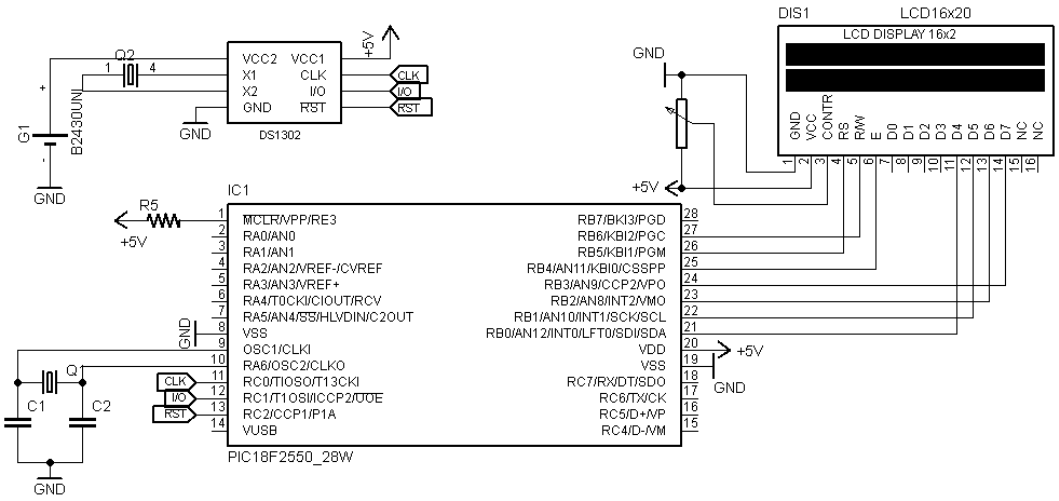
Estos 2 archivos los podemos colocar en el directorio de nuestro proyecto, o crear una carpeta especial para ir almacenando nuestras librerías, lo único que se debe hacer es en ***Project/Build Options/Project***, en la solapa ***Directories/Include Search Path*** indicar la dirección de donde esta ubicada, y agregar **ambos** (*.h y *.c) a nuestro proyecto para que realice la compilación correspondiente.

5.6. Ejemplo, uso del timer0

5.6.1. Objetivo

Aprovechado la creación de librería para el control del DS1302 se creará un Reloj/Calendario visualizandolo en un LCD. Utilizaremos el timer0 para generar una interrupción cada 1 segundo para leer el DS1302 y actualizar sus valores en el LCD.

5.6.2. Hardware



5.6.3. Configuración del Timer

Para determinar las funciones y parámetros que necesitamos para configurar el Timer seleccionado vamos a *C:/MCC18/doc/periph-lib* y abrimos *Timer.htm*
Allí debemos fijarnos, como venimos haciendo, en que grupo de funciones entra nuestro PIC en la primer tabla del documento, para el PIC18F2550 es TMR_V2.

Las funciones que disponemos son:

- Open_TimerX: Configura y habilita el Timer.
- Close_TimerX: Deshabilita el Timer.
- Write_TimerX: Escribe un valor en el TimerX.
- Read_TimerX: Realiza una lectura del TimerX.
- SetTmrCCPSrc: Configura el Timer para ser utilizado por algún Módulo CCP.

Ejemplo:

Open_Timer0(unsigned char config)

Configuración de Interrupción:

```
TIMER_INT_ON            Interrupcion habilitada
TIMER_INT_OFF           Interrupcion deshabilitada
```

Selección entre modo de 8 o 16 bits:

```
T0_8BIT                  Modo 8 bits
T0_16BIT                 Modo 16 bits
```

Fuente del Clock:

```
T0_SOURCE_EXT           Clock externo (I/O pin)
T0_SOURCE_INT           Clock interno (Tosc)
```

Para Clock externo, Selección de flanco:

```
T0_EDGE_FALL            Flanco descendente
T0_EDGE_RISE            Flanco ascendente
```

Valor de Preescaler:

```
T0_PS_1_1               1:1 prescale
T0_PS_1_2               1:2 prescale
T0_PS_1_4               1:4 prescale
T0_PS_1_8               1:8 prescale
T0_PS_1_16              1:16 prescale
T0_PS_1_32              1:32 prescale
T0_PS_1_64              1:64 prescale
T0_PS_1_128             1:128 prescale
T0_PS_1_256             1:256 prescale
```

En nuestro caso necesitamos realizar una interrupción cada 1 segundo para el refresco de los datos en el LCD, para ello configuramos el Timer0 en Modo 16 bits, Clock interno, Preescaler 1:16 y habilitación de la interrupción:

```
OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16);
```

Y para lograr una interrupción cada 1 segundo se debe setear el timer en 3036, calculado para oscilador de 4MHz:

```
WriteTimer0(3036);
```

Nota: Se puede ver claramente que el código de actualización de datos en pantalla puede hacerse mucho mas eficiente, actualizando solo parte según sean los datos recibidos, pero solo es un ejemplo de forma de uso.

5.6.4. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
#include <delays.h>
#include <xlcd.h>
#include <ds1302.h>
#include <stdlib.h>
#include <timers.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 day,mth,year,dow,hour,min,sec;   //variabes para ds1302
volatile unsigned char kbhit_update; // Para determinar cuando hay que actualizar.-

void DelayFor18TCY(void);
void DelayPORXLCD(void);
void DelayXLCD(void);
void comandXLCD(unsigned char a);
void gotoxyXLCD(unsigned char x, unsigned char y);
void printDate(unsigned char day, unsigned char mth, unsigned char year);
void printTime(unsigned char hour, unsigned char min, unsigned char sec);
void get_date(void);
void get_time(void);
void Delay_2us(void);
void ISRTimer0(void);


// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRTimer0 _endasm
}
#pragma code // Cerramos seccion.-

// Rutina de Interrupcion.-
// Esta interrupcion se ejecuta cada 1s.
#pragma interrupt ISRTimer0
void ISRTimer0(void){
    if(INTCONbits.TMR0IF==1){
        // Cargamos nuevamente Timer0 para interrupcion cada 1seg.
        WriteTimer0(3036);
        kbhit_update=1;
        INTCONbits.TMR0IF=0;    // Borramos bandera.-
    }
}


//*********************************************************************
void main(void){

    ds1302_init();                                  //inicializa ds1302.-
    //Configuracion del Timer1
    OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16);
    OpenXLCD(FOUR_BIT & LINES_5X7);     // Iniciamos LCD.-
    comandXLCD(0x06); // Nos aseguramos incremento de direccion, display fijo

    comandXLCD(0x0C);                   // Encendemos LCD.-
    putrsXLCD("Usando el ds1302");
    gotoxyXLCD(1,2);
    putrsXLCD("by Suky");
    day=7; mth=8; year=9;dow=18;
    hour=22; min=59;
    set_datetime(day,mth,year,dow,hour,min);
    Delay10KTCYx(100);
```

```
    comandXLCD(1); // Borramos display

    kbhit_update=0;
    // Iniciando el Timer en 3036 logramos un interrupcion cada 1s con 4MHz.
    WriteTimer0(3036);
    RCONbits.IPEN=0;     // Deshabilitamos Prioridades
    INTCONbits.PEIE=1;   // Habilitamos interrupcion de perifericos.-
    INTCONbits.GIE=1;    // Habilitamos interrupcion Global.

    while(1){
        if(kbhit_update==1){
            kbhit_update=0;
            get_date();    //Lee dia,mes,a\~no
            get_time();       //lee hora,minuto,segundo
            gotoxyXLCD(1,1);
            printDate(dow,mth,year);    //lcd
            gotoxyXLCD(1,2);
            printTime(hour,min,sec);
        }
    }
}

//*********************************************************************
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 = 0x40;
    else
        direccion=0;

    direccion += x-1;
    comandXLCD(0x80 | direccion);
}
// Demora de 2us para DS1302.
void Delay_2us(void){
    Nop();
    Nop();
}
// Funcion que imprime la Fecha en el LCD. Toma los valores ingresados,..
// ..los convierte a string y los imprime. En el caso que sea menor a 10,..
```

```
// .. se imprime un "0" antes del valor.
void printDate(unsigned char dow, unsigned char mth, unsigned char year){
    char String[4];

    putrsXLCD("Fecha:");
    itoa(dow, String);          // Convertimos entero a string.-
    if(dow<10){putrsXLCD("0");}
    putsXLCD(String);putrsXLCD("/");
    itoa(mth, String);          // Convertimos entero a string.-
    if(mth<10){putrsXLCD("0");}
    putsXLCD(String);putrsXLCD("/");
    itoa(year, String);         // Convertimos entero a string.-
    if(year<10){putrsXLCD("0");}
    putsXLCD(String);
}
// Funcion que imprime la Hora en el LCD. Toma los valores ingresados,..
// ..los convierte a string y los imprime. En el caso que sea menor a 10,..
// ..se imprime un "0" antes del valor.
void printTime(unsigned char hour, unsigned char min, unsigned char sec){
    char String[4];

    putrsXLCD("Hora:");
    itoa(hour, String);         // Convertimos entero a string.-
    if(hour<10){putrsXLCD("0");}
    putsXLCD(String);putrsXLCD(":");
    itoa(min, String);          // Convertimos entero a string.-
    if(min<10){putrsXLCD("0");}
    putsXLCD(String);putrsXLCD(":");
    itoa(sec, String);          // Convertimos entero a string.-
    if(sec<10){putrsXLCD("0");}
    putsXLCD(String);
}
// Funcion que realiza la lectura del todos los datos de la Fecha.
void get_date(void){   //Lee dia,mes,a\~no
    dow=get_dow();
    mth=get_mth();
    year=get_year();
    dow=get_dow();
}
// Funcion que realiza la lectura del todos los datos de la Hora.
void get_time(void){      //lee hora,minuto,segundo
    hour=get_hr();
    min=get_min();
    sec=get_sec();
}
```

5.7. Comunicación I^2C

Muchos de los dispositivos (memorias, RTC, sensores varios, conversor AD y DA, ect) que utilizamos diariamente en el desarrollo de proyectos utilizan el bus I^2C. Este solo requiere dos líneas de señal (Reloj y datos) y una de referencia (GND). Permite el intercambio de información entre muchos dispositivos a una velocidad aceptable, de unos 100 Kbits por segundo, aunque hay casos especiales en los que el reloj llega hasta los 1 MHz.
Microchip nos provee las librerías necesarias para poder controlar el módulo MSSP del micro-

controlador y de esa manera establecer comunicación utilizando el protocolo I²C. Para saber en que grupo de funciones cae nuestro PIC se debe ver la tabla en el documento I²C.htm que se encuentra en /MCC18/doc/periph-lib, en nuestro caso el *PIC18F2550* se encuentra dentro del grupo *I2C_V1*. Sabiendo esto, podemos ver los parámetros de cada una de las funciones que utilizaremos:

```
OpenI2C( unsigned char sync_mode, unsigned char slew );
```

sync_mode

- **SLAVE_7** I2C Modo Esclavo, 7-bit de dirección
- **SLAVE_10** I2C Modo Esclavo, 10-bit de dirección
- **MASTER** I2C Modo Maestro

slew

- **SLEW_OFF** Slew rate deshabilitado para modo 100 kHz
- **SLEW_ON** Slew rate habilitado para modo 400 kHz

Para configurar la velocidad, además debemos cargar el registro **SSPADD** , que dependerá de la velocidad del oscilador. *Clock=Fosc/(4.(SSPADD + 1))*

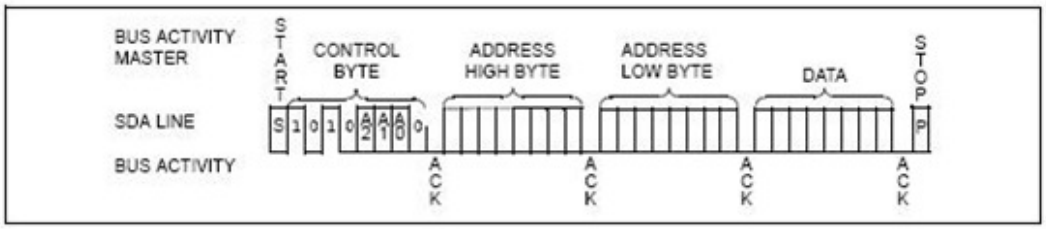
5.7.1. Primer ejemplo, estableciendo comunicación con memoria 24LC

Objetivos

Lograr comunicarnos con una memoria serial **24LC512** mediante el protocolo I²C. En este primer ejemplo vamos a recibir un dato desde PC, lo guardaremos en la memoria y luego de un tiempito realizaremos la lectura de la memoria reenviándolo a la PC.

Escritura de byte

Luego de la condición de inicio del Master, el código de control (cuatro bits), los bits de selección de chip (tres bits) y el bit de R / W (que es un "0") son registrados en el bus por el transmisor. Esto indica al esclavo direccionado que el byte de dirección superior seguirá después de que se haya generado un bit de reconocimiento durante el noveno ciclo de reloj. Por lo tanto, el próximo byte transmitido por el Master es el byte alto de la dirección de la palabra. El próximo es el byte de dirección menos significativo. Después de recibir otra señal de reconocimiento del 24XX512, el Master transmitirá el dato a ser escrito en la ubicación de memoria direccionada. El 24XX512 reconoce otra vez y el Master genera un condición de alto. Esto inicia el ciclo de escritura interno y durante esta vez, el 24XX512 no generará señales de reconocimiento (Figura). Después de un byte de comando de escritura, el contador de dirección interno apuntará a la ubicación de dirección siguiente.

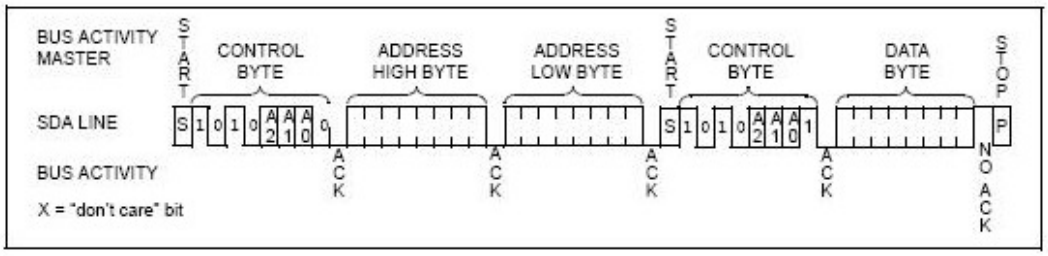


Operación de lectura

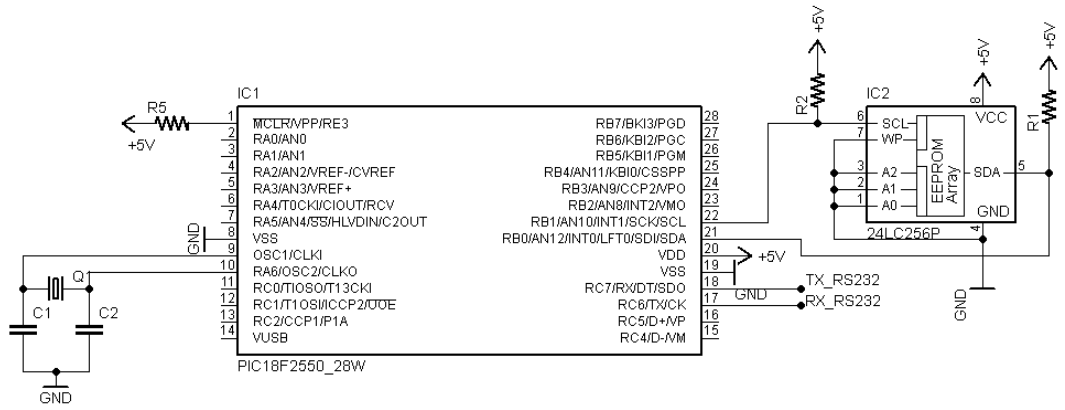
Las operaciones de lectura son iniciadas del mismo modo como las operaciones de escritura, con la excepción de que la parte de R / W del byte de control es puesta a "1". Hay tres tipos básicos de las operaciones de lectura: la lectura de dirección en curso, lectura aleatoria y lectura secuencial.

Lectura aleatoria

Las operaciones de lectura aleatorias permiten que el Master acceda a cualquier ubicación de la memoria en una manera aleatoria. Para llevar a cabo este tipo de operación de lectura, la dirección del dato a leer debe ser enviada primero. Esto se hace enviando la dirección del dato como una operación de escritura (R/ W a "0"). En cuanto la dirección del dato es enviada, el Master genera una de condición de inicio seguida al reconocimiento. El Master envía el byte de control otra vez, pero con el bit de R / W en 1.



5.7.2. Hardware



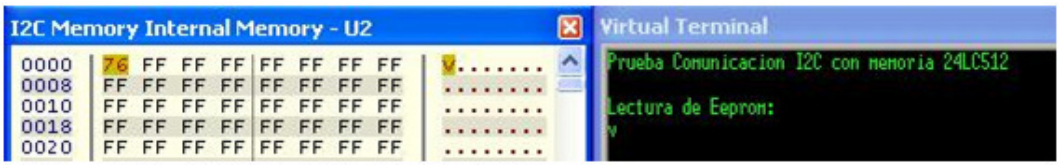
5.7.3. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
#include <delays.h>
#include <usart.h>
#include <i2c.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

//*****************************************************************
volatile char Data, Kbhit;

void ISRRecepcion(void);
void ByteWriteI2C( unsigned char ByteControl, unsigned char HighDireccion,
unsigned char LowDireccion, unsigned char DataI2C );
unsigned char  ByteReadI2C( unsigned char ByteControl, unsigned char HighDireccion,
unsigned char LowDireccion );
// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRRecepcion _endasm
}
#pragma code // Cerramos seccion.-

// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion
void ISRRecepcion(void){
    if(PIR1bits.RCIF==1){
```

```
        Data=getcUSART();       // Leemos Dato recibido
        Kbhit=1;                        // Indicamos que se ha recibido un dato.-
        PIR1bits.RCIF=0;        // Borramos bandera.-
    }
}

void main(void){

    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH,25);//9600,8,n,1
    OpenI2C(MASTER,SLEW_OFF);  //Master, 100KHz
    SSPADD = 9;//100KHz para 4MHz

    Kbhit=0;
    RCONbits.IPEN=0;    // Deshabilitamos Prioridades
    INTCONbits.PEIE=1;  // Habilitamos interrupcion de perifericos.-
    INTCONbits.GIE=1;           // Habilitamos interrupcion Global.

    putrsUSART("Prueba Comunicacion I2C con memoria 24LC512 \r\r");
    while(1){
        while(Kbhit==0){}; // esperamos a recibir dato.-
        Kbhit=0;
        ByteWriteI2C(0xA0,0x00,0x00,Data);
        Delay1KTCYx(5);
        putrsUSART("Lectura de Eeprom: \r");
        putcUSART(ByteReadI2C(0xA0,0x00,0x00));
        putrsUSART("\r");
    }
}
//*****************************************************************************
void ByteWriteI2C( unsigned char ByteControl, unsigned char HighDireccion,
                   unsigned char LowDireccion, unsigned char DataI2C ){

    IdleI2C();// El modulo esta activo?
    StartI2C();// Condicion de START
    while ( SSPCON2bits.SEN );
    WriteI2C( ByteControl );// Envia Byte de control
    WriteI2C( HighDireccion );
    WriteI2C( LowDireccion );
    WriteI2C ( DataI2C );// Guarda Data en Eeprom en la direccion establecida.
    StopI2C();// Condicion de STOP
    while ( SSPCON2bits.PEN );
    while (EEAckPolling(ByteControl));//Espera que se complete escritura.
}

unsigned char ByteReadI2C( unsigned char ByteControl, unsigned char HighDireccion,
                           unsigned char LowDireccion ){

    unsigned char Valor;

    IdleI2C();// El modulo esta activo?
    StartI2C();// Condicion de START
    while ( SSPCON2bits.SEN );
    WriteI2C( ByteControl );
    WriteI2C( HighDireccion );
    WriteI2C( LowDireccion );
    RestartI2C();// Envia ReStart
```

```
    while ( SSPCON2bits.RSEN );
    // Cambia bit0 de ByteControl a 1 para realizar lectura.
    WriteI2C( ByteControl | 0x01 );
    Valor=ReadI2C();// Realiza lectura.
    NotAckI2C();// Envia NACK
    while ( SSPCON2bits.ACKEN );
    StopI2C();// Condicion de STOP
    while ( SSPCON2bits.PEN );
    return ( Valor );// Retorna Lectura
}
```

Trabajando con proteus podemos realizar una prueba del código, por ejemplo, escribiendo y leyendo la letra v en la dirección 0x0000.

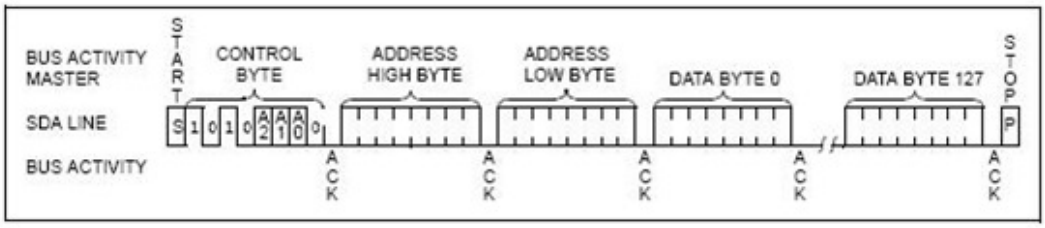


5.7.4. Segundo ejemplo de comunicación con memoria 24LC512 mediante I2C

En este caso vamos a recibir desde la PC un string, que guardaremos en la memoria y luego leeremos para reenviar a la PC. Utilizaremos escritura secuencial y lectura secuencial:

Proceso de escritura

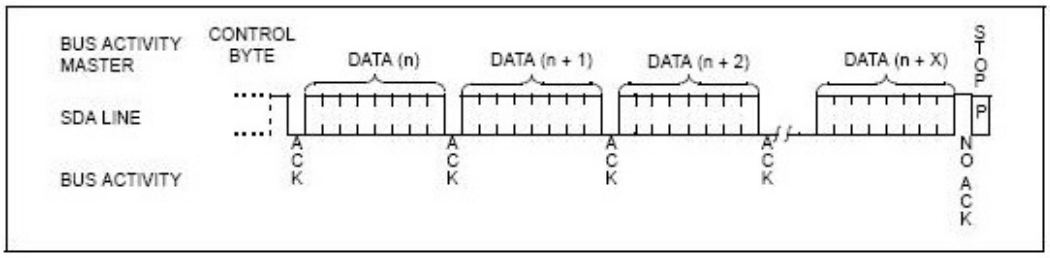
En este caso vamos a enviar un buffer directamente a la memoria, para ello enviaremos la dirección del primer Byte a enviar como se ha realizado en el caso anterior, y luego el buffer completo aprovechando que al realizar una escritura, el contador de dirección interno apuntará a la ubicación de dirección siguiente. La cantidad máxima de datos que podemos enviar de esta manera dependerá de la memoria utilizada, en nuestro caso la 24LC512 tiene páginas de 128 bytes.



Lectura secuencial

Las lecturas secuénciales son iniciadas del mismo modo como una lectura aleatoria excepto que después de que la 24LC512 transmite el primer byte de datos, el Master envía un reconocimiento.

Este ack ordena a la memoria transmitir la próxima palabra de 8 bits secuencialmente direccionada (Figura). Cuando se recibe el último dato, el Master no generará un ack, pero generará una condición de alto. Para proveer las lecturas secuénciales, la memoria contiene un puntero de dirección interno que es incrementado en uno en la terminación de cada operación.



5.7.5. Código

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
#include <delays.h>
#include <usart.h>
#include <i2c.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

//*************************************
volatile unsigned char Kbhit;
volatile unsigned char BufferE[19];
unsigned char BufferL[19];

void ISRRecepcion(void);
void BufferWriteI2C( unsigned char ByteControl, unsigned char HighDireccion,
                     unsigned char LowDireccion, unsigned char *DataI2C );
void BufferReadI2C( unsigned char ByteControl, unsigned char HighDireccion,
           unsigned char LowDireccion, unsigned char *Valor, unsigned char Lengh );

// Creamos una nueva seccion de codigo a partir de la direccion 0x08.-
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRRecepcion _endasm
}
#pragma code // Cerramos seccion.-

// Rutina de Interrupcion.-
#pragma interrupt ISRRecepcion
void ISRRecepcion(void){
```

```
    if(PIR1bits.RCIF==1){
        getsUSART(BufferE,19); // recibimos 19 caracteres.-
        Kbhit=1;// Indicamos que se ha recibido un dato.-
        PIR1bits.RCIF=0;        // Borramos bandera.-
    }
}


void main(void){
    unsigned char i;

    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH,25);//9600,8,n,1
    OpenI2C(MASTER,SLEW_OFF);  //Master
    SSPADD = 9;//100KHz para 4MHz

    Kbhit=0;
    RCONbits.IPEN=0;// Deshabilitamos Prioridades
    INTCONbits.PEIE=1;// Habilitamos interrupcion de perifericos.-
    INTCONbits.GIE=1;// Habilitamos interrupcion Global.

    putrsUSART("Prueba Comunicacion I2C con memoria 24LC512 \r\r");
    while(1){
    while(Kbhit==0){};// esperamos a recibir dato.-
        Kbhit=0;
        //Se guarda en memoria los datos recibidos.
        BufferWriteI2C(0xA0,0x00,0x00,&BufferE[0]);
        Delay1KTCYx(5);
        // Leemos de la memoria los datos guardados.-
        BufferReadI2C(0xA0,0x00,0x00,&BufferL[0],19);
        putrsUSART("Lectura de Eeprom: \r");
        putsUSART(BufferL); putrsUSART("\r");
    }
}
//***********************************************
void BufferWriteI2C( unsigned char ByteControl, unsigned char HighDireccion,
         unsigned char LowDireccion,unsigned char *BufferData){

    IdleI2C();// El modulo esta activo?
    StartI2C();// Condicion de START
    while ( SSPCON2bits.SEN );
    WriteI2C( ByteControl );// Envia Byte de control
    WriteI2C( HighDireccion );
    WriteI2C( LowDireccion );
    putsI2C(BufferData);// Enviamos buffer a memoria.
    StopI2C();// Condicion de STOP
    while ( SSPCON2bits.PEN );
    while (EEAckPolling(ByteControl));//Espera que se complete escritura.
}

void BufferReadI2C( unsigned char ByteControl, unsigned char HighDireccion,
           unsigned char LowDireccion,unsigned char *Valor, unsigned char Lengh ){

    IdleI2C();// El modulo esta activo?
    StartI2C();// Condicion de START
    while ( SSPCON2bits.SEN );
    WriteI2C( ByteControl );
```

```
    WriteI2C( HighDireccion );
    WriteI2C( LowDireccion );
    RestartI2C();// Envia ReStart
    while ( SSPCON2bits.RSEN );
    // Cambia bit0 de ByteControl a 1 para realizar lectura.
    WriteI2C( ByteControl | 0x01 );
    getsI2C(Valor,Lengh);// Realiza lectura de cantidad=Lengh Datos.
    NotAckI2C();// Envia NACK
    while ( SSPCON2bits.ACKEN );
    StopI2C();// Condicion de STOP
    while ( SSPCON2bits.PEN );
}
```

Ejemplo de grabación:

Capítulo 6

Introducción a estructuras, uniones y punteros

En este capitulo se pretende dar una pequeña introducción al uso de estructuras, uniones y punteros sin profundizar mucho en los conceptos. Se dan a conocer para tener una idea básica de su implementación y para despertar curiosidad de su utilización, ya que son herramientas útiles en el desarrollo de programas de mayor complejidad.

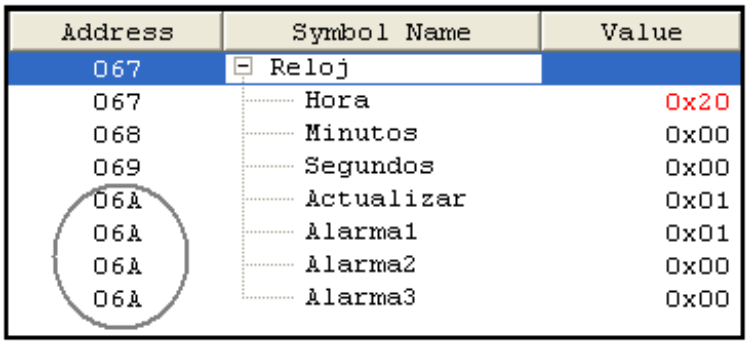
6.1. Estructuras

Una Estructura es una colección de variables, que pueden contener diferentes tipos de datos. Ayudan a organizar y manejar datos debido a que agrupan diferentes tipos de datos a las que se les trata como una sola unidad en lugar de ser vistas como unidades separadas. Por ejemplo en la realización de con reloj necesitamos definir las siguientes variables:

```
unsigned char Hora, Minutos, Segundos, Actualizar, Alarma1, Alarma2, Alarma3;
```

Todas ellas son tratadas individualmente, pero podemos agruparlas en una estructura de las siguiente forma:

```
struct{
        unsigned char Hora;
        unsigned char Minutos;
        unsigned char Segundos;
        unsigned Actualizar:1;
        unsigned Alarma1:1;
        unsigned Alarma2:1;
        unsigned Alarma3:1;
} Reloj;
```



Actualizar, Alarma1, Alarma2 y Alarma3 son banderas boolenas que utilizamos para determinar si ha ocurrido un evento y por lo tanto solo necesitan 1-bit. Esto se lo indicamos colocando :1 seguido de la definición.
Ahora para acceder a cada miembro de la estructura utilizamos el operador punto por ejemplo de la siguiente manera:

```
Reloj.Minutos=15;
if(Reloj.Actualizar==1){

}
```

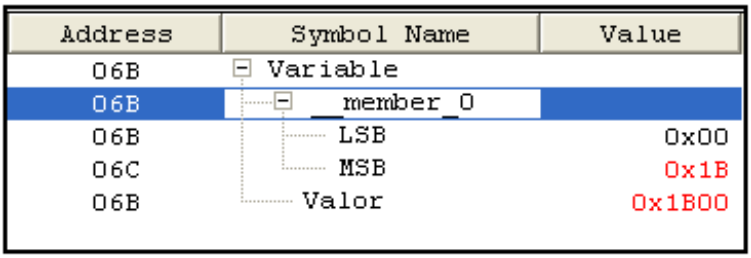
6.2. Uniones

Las uniones son similares a las estructuras, con la diferencia de que en las uniones se almacenan los campos solapándose unos con otros en la misma disposición. Se implementa para llamar de dos formas distintas a un mismo sector de memoria. Un ejemplo sencillo es la siguiente declaración:

```
union{
        unsigned int Valor;
        struct{
                unsigned LSB:8;
                unsigned MSB:8;
  };
}Variable;
```

Estamos reservando 16-bits en RAM para almacenar una variable del tipo int (Valor) pero que podemos acceder a sus bytes (Alto y bajo) de forma independiente. Para acceder a sus miembros también se utiliza el operador punto.

```
Variable.LSB=0x00;
Variable.MSB=0x1B;
```



Al leer *Variable.Valor* obtenemos 0x1B00.-

6.3. Typedef

Algo también interesante del lenguaje de programación C es la re definición de tipos de datos para crear nombres más cortos o significativos del tipo de datos. Hay que recalcar que no crea ningún nuevo tipo, solo define un nuevo identificador para un tipo que ya tiene su propio identificador.

Ejemplos:

```
typedef unsigned char UINT8;
typedef signed char SINT8;
typedef unsigned int COUNTERS;
typedef unsigned char ERRORS;
```

También es posible utilizar typedef al mismo tiempo que se declara una estructura o una unión. Ejemplo:

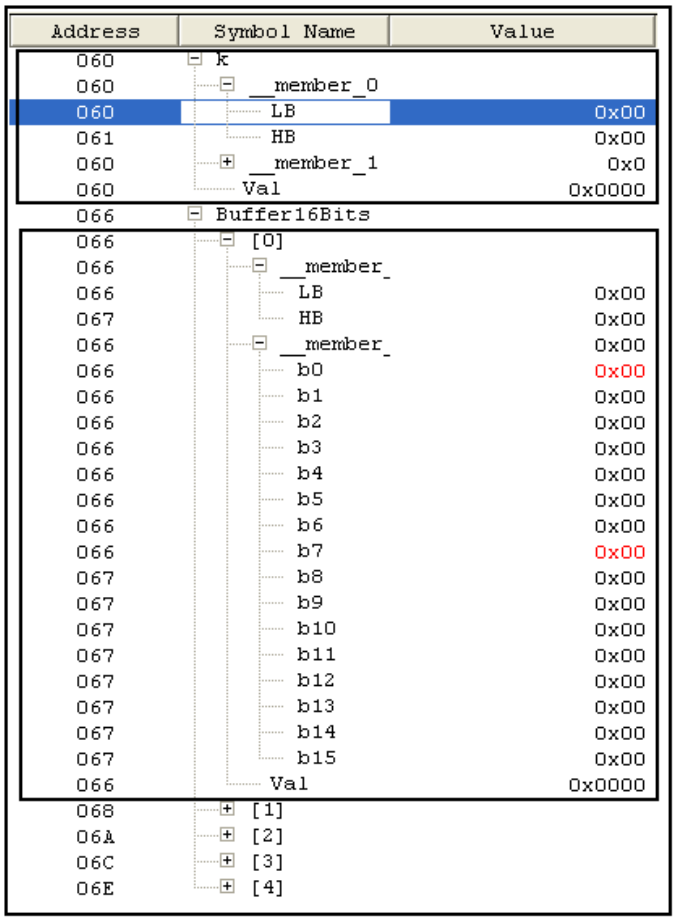
```
typedef union
{
    UINT16 Val;
    struct
    {
        UINT8 LB;
        UINT8 HB;
    };
    struct
    {
        UINT8 b0:1;
        UINT8 b1:1;
        UINT8 b2:1;
        UINT8 b3:1;
        UINT8 b4:1;
        UINT8 b5:1;
        UINT8 b6:1;
        UINT8 b7:1;
        UINT8 b8:1;
        UINT8 b9:1;
        UINT8 b10:1;
        UINT8 b11:1;
```

```
        UINT8 b12:1;
        UINT8 b13:1;
        UINT8 b14:1;
        UINT8 b15:1;
    };
} UINT16_VAL;
```

Lo anterior corresponde a la definición de una unión anónima que puede ser utilizada para crear varias variables que implementen tal estructura.

```
UINT16_VAL  k, j, Test;
UINT16_VAL Buffer16Bits[5];
```

Todas las variables creadas se corresponden con la estructura definida:



6.4. Punteros

Un apuntador o puntero es una variable de 16-bits o 24-bits (Variables en rom FAR) que contiene una dirección. Así si Data en un ***char*** y ***ptr*** es un puntero que apunta a él tendremos lo siguiente:

Direccion	Nombre	Valor
0x10	Data	0xAA
0x11	—	—
0x12	—	—
0x13	ptr	0x10

Definición

```
// Declaracion de una variable puntero a una variable char.
char *ptr;
// Declaracion de una variable puntero a una variable int.
int *ptr;
//Declaracion de una variable puntero a una variable unsigned char.
unsigned char *ptr;
```

Operadores unitarios

- **&: Operador de dirección**: Da la dirección de una variable, por lo que la expresión ptr=&Data asigna la dirección de Data a ptr, diciéndose entonces que ptr apunta a Data.

- ***: Operador de indirección**: Da acceso a la variable que apunta el apuntador. Por ejemplo la expresión *ptr=0xAA asigna a la variable que apunta prt el valor 0xAA, o sea Data=0xAA.-

Por ejemplo:
Declaramos tres variables, una ubicada en RAM, otra en ROM pero ubicada en los primeros bancos (near), y una tercera en ROM pero sin restricción de banco(far). También creamos los punteros para direccionarlas y en el bucle principal hacemos ese paso.

```
char VariableRAM=15;
rom near char VariableROMNear=20;
rom far char VariableROMFar=25;

char *PtrRAM;
rom near char *PtrROMNear;
rom far char *PtrROMFar;
```

```
PtrRAM=&VariableRAM;
PtrROMNear=&VariableROMNear;
PtrROMFar=&VariableROMFar;
```

Address	Symbol Name	Value
06A	PtrRAM	0x0073
06E	PtrROMFar	0x00141C
06C	PtrROMNear	0x141B
073	VariableRAM	15
P 141C	VariableROMFar	25
P 141B	VariableROMNear	20

Como pueden ver en el ejemplo, como una variable far puede estar ubicada en cualquier banco incluyendo aquellos sobre la dirección 64k se le asigna 24-bits para contener la dirección de la variable.

Más ejemplos:

```
// A lo que apunta ptr se le suma 10 y se le asigna a Data1.-
Data1= *ptr + 10;

// Se incrementa en 1 lo que apunta ptr.-
(*ptr)++;

// Se incrementa el puntero a la siguiente direccion de variable
// (Dependera de que tipo de variable se apunta)
*ptr++;

// Se incrementa en 1 lo que apunta ptr.-
++*ptr;

// se asigna al segundo puntero lo que apunta ptr.
// O sea los 2 punteros apuntan a la misma variable.-
ptr2=ptr;
```

Podemos acceder a los bytes de una variable de mayor de 8-bits utilizado punteros y el operando cast:

```
unsigned int Variable;
unsigned char LSB, MSB;
LSB=*((unsigned char *)&Variable);
MSB=*((unsigned char *)&Variable+1);
```

Con *(unsigned char *)* hacemos que la dirección obtenida de &*Variable* sea tomada como a una variable del tipo *char*, luego con **((..)&Variable)* transferimos el byte apuntado a la variable *LSB*.

Punteros y arreglos

Si declaramos un arreglo como:

```
char Datos[10];
```

Declaramos un ptr:

```
char *ptr;
```

Podemos asignar al puntero la primer posición del arreglo:

```
ptr=& Datos[0];
```

Y podemos cargar a Datos[0] y Datos[5] de la siguiente manera:

```
*ptr=0xAA;
*(ptr+5)=0x48;
```

Cargar un buffer dentro de otro:

```
unsigned int *ptr,*ptr2;
unsigned int Datos01[20], Datos02[5]={0xFFAA,0xCC45,0x7895,0x3568,0xCC21};

ptr=&Datos01[8]; // Vamos a cargar en el arreglo 1 a partir de la posici\'on 8.-
for(ptr2=Datos02;ptr2<Datos02+5;ptr2++,ptr++){
*ptr=*ptr2;
}
```

Punteros y funciones

En C los argumentos de las funciones se pasan por valor, así que no debería de ser posible en una función modificar variables de la función que la llama. Digo debería porque pruebas hechas en C18 si lo permite Grin pero vamos a tratar C convencional para que sirva para otras compiladores.

Usemos el ejemplo convencional para mostrar esto, por ejemplo si queremos reorganizar variables:

Forma incorrecta:

```
void funcion(void){
   char a,b;
   a=10;
   b=15;
   swap(a,b);
}
void swap(char q,char p){
```

```
    char temp;
    temp=q;
    q=p;
    p=temp;
}
```

Como vemos estamos modificando variables de la función que llama a swap. La forma correcta es hacerlo por pasaje de referencia, o sea le entregamos a la función swap la dirección de las variables:

```
void funcion(void){
    char a,b;
    a=10;
    b=15;
    swap(&a,&b);
}
void swap(char *q,char *p){
    char temp;
    temp=*q;
    *q=*p;
    *p=temp;
}
```

También el uso de punteros en argumentos de funciones nos es sumamente útil para que una función pueda retornar más de una variable entregando por referencia un arreglo.

Por ejemplo en un arreglo colocamos el nombre de un archivo más su extensión, y queremos una función que separe en dos arreglos distintos el nombre y la extensión:

```
void funcion(void){
    char NombreMasExt[13]= "Tutorial.pdf";
    char Nombre[9],Ext[4];

    separamos(&NombreMasExt[0],&Nombre[0],&Ext[0]);
}

void separamos(char *ptr,char *nptr,char *eptr){

   do{
       *nptr++=*ptr++;
   }while(*ptr!= '.'); // Hasta que se direccione el '.'
   //Siempre que se trabaje con string se debe indicar finalizacion con '\0'
   *nptr='\0';
   *ptr++;      // Pasamos a direccionar la siguiente letra, despues de '.'
   do{
       *eptr++=*ptr++;
   }while(*ptr!='\0');
   *eptr='\0';
}
```

Nota: Cuando se trabaja con string se debe tener en cuenta una posición que indica final del string '\0'.


InfoPIC
www.infopic.comlu.com