

MPLAB C18 Desde 0

[Escribir el subtítulo del documento]

07/07/2009

Suky

www.ucontrol.com.ar

Microchip MPLAB C18.

Entorno de programación MPLAB IDE.

- Ensamblador, enlazador, gestión de proyectos, depurador y simulador
- Gratuito, se puede descargar de www.microchip.com

Compilador MPLAB C18

- Compilador cruzado de lenguaje C para microcontroladores PIC18
- 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

Índice

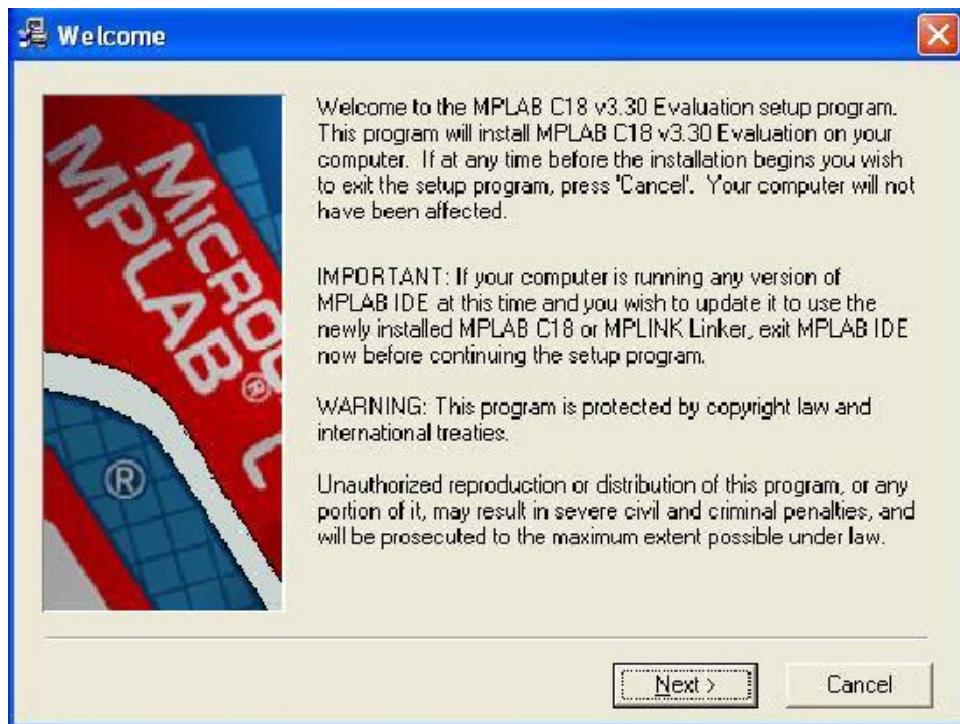
- [Instalación](#)
- [Creación de un proyecto](#)
- [1º Ejemplo.](#) Control de un led mediante un pulsador. (Definición de operadores, estructuras, y control de bits de registros)
- [2º Ejemplo.](#) Led titilando (Definición de Variables, librería de Demoras)
- [3º Ejemplo.](#) Led secuenciales.
- [4º Ejemplo.](#) Control Display 7 Segmentos (Definición de Arreglos de variables)
- [5º Ejemplo.](#) Control de Varios Display de 7 Segmentos por multiplexión (Declaración y definición de funciones)
- [6º Ejemplo.](#) Control LCD
- [7º Ejemplo.](#) Conversión analógica/digital.
- [Manejo de interrupciones.](#)
- [8º Ejemplo.](#) Comunicación Serial RS232
- [Modificación de una librería,](#) Cambiando el puerto de Control de LCD
- [Control de Teclado Matricial,](#) Interrupción por RB4-RB7
- [Creación de una librería,](#) DS1302
- [Reloj/Calendario con DS1302 y LCD,](#) Interrupción por desbordamiento Timer0
- [Comunicación I2C,](#) primer ejemplo Lectura/escritura Aleatorias
- [Comunicación I2C,](#) segundo ejemplo Lectura/Escritura secuenciales

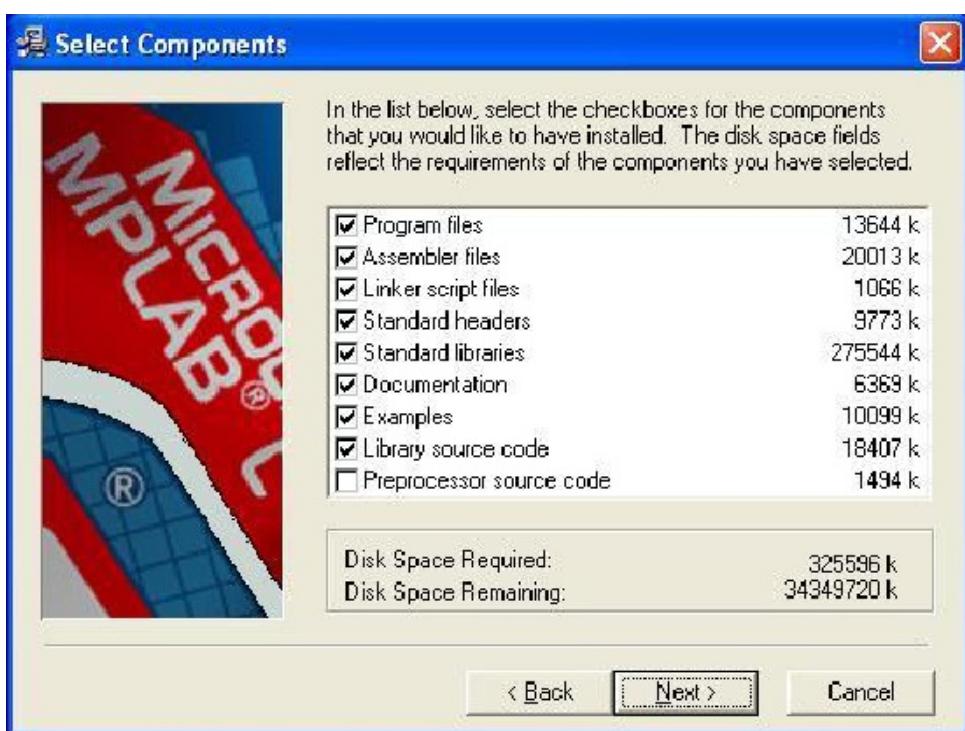
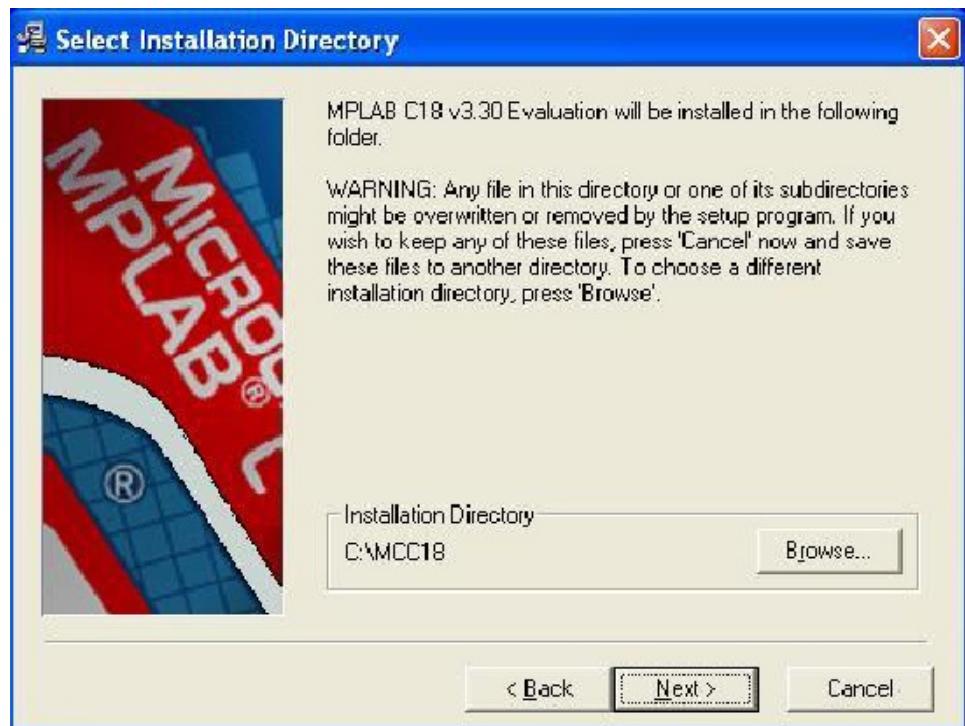
Instalación

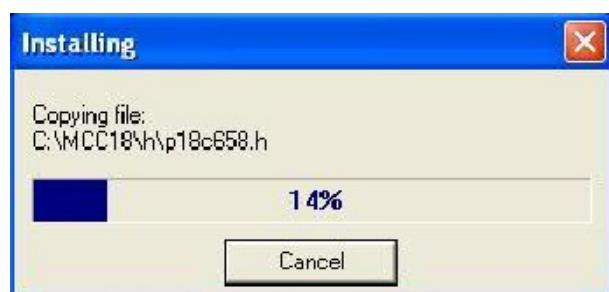
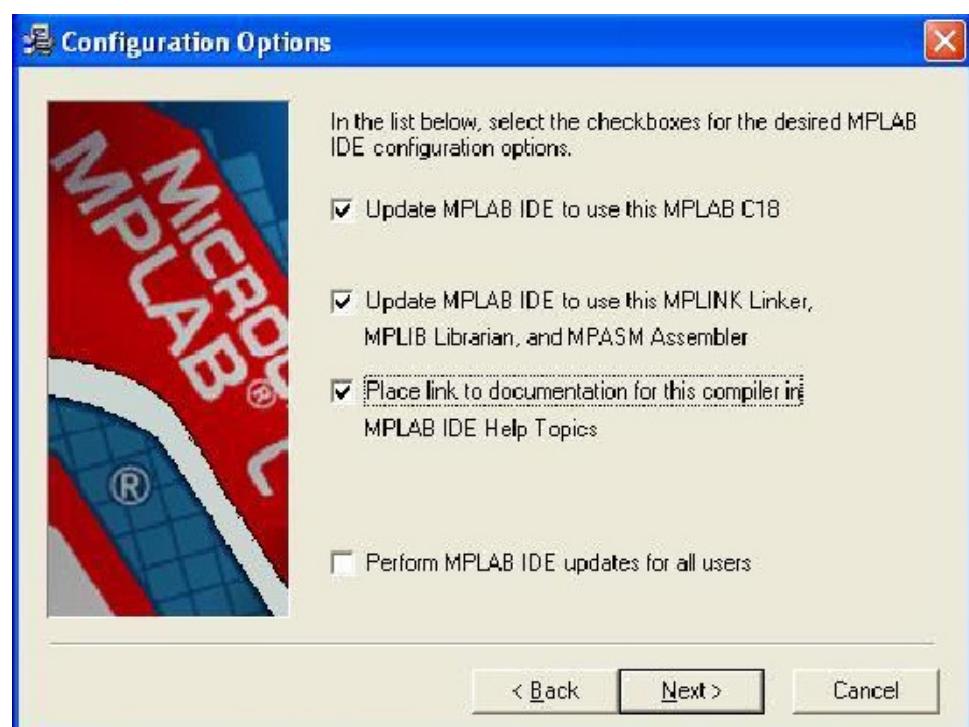
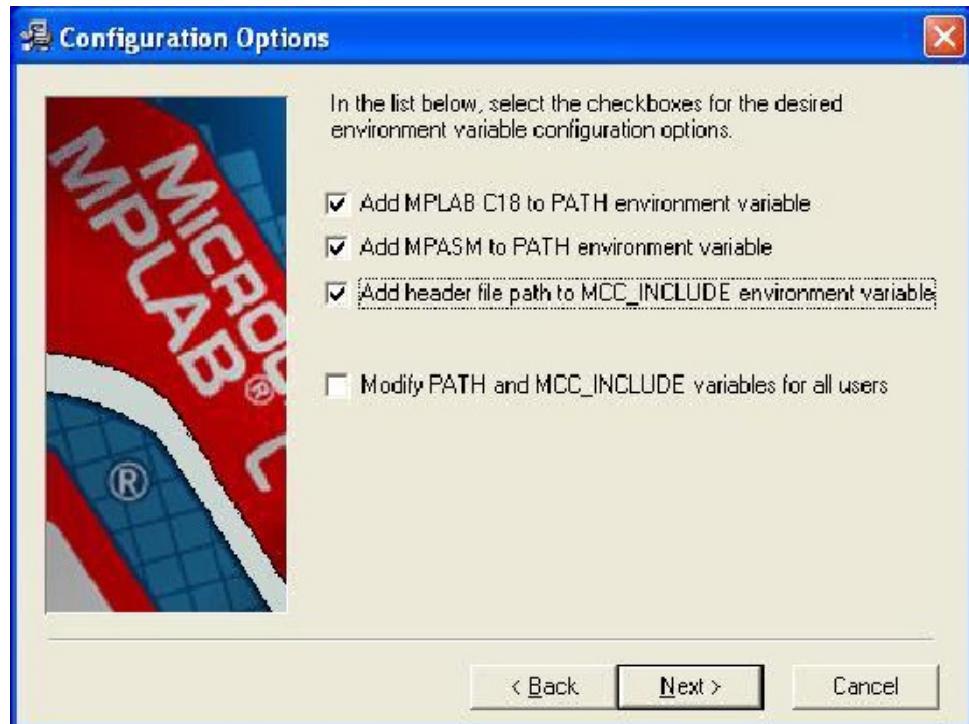
Se debe bajar directamente desde Microchip, hay disponible una versión gratuita para estudiantes que es un demo de 60 días. También para poder descargarlo es necesario registrarse.

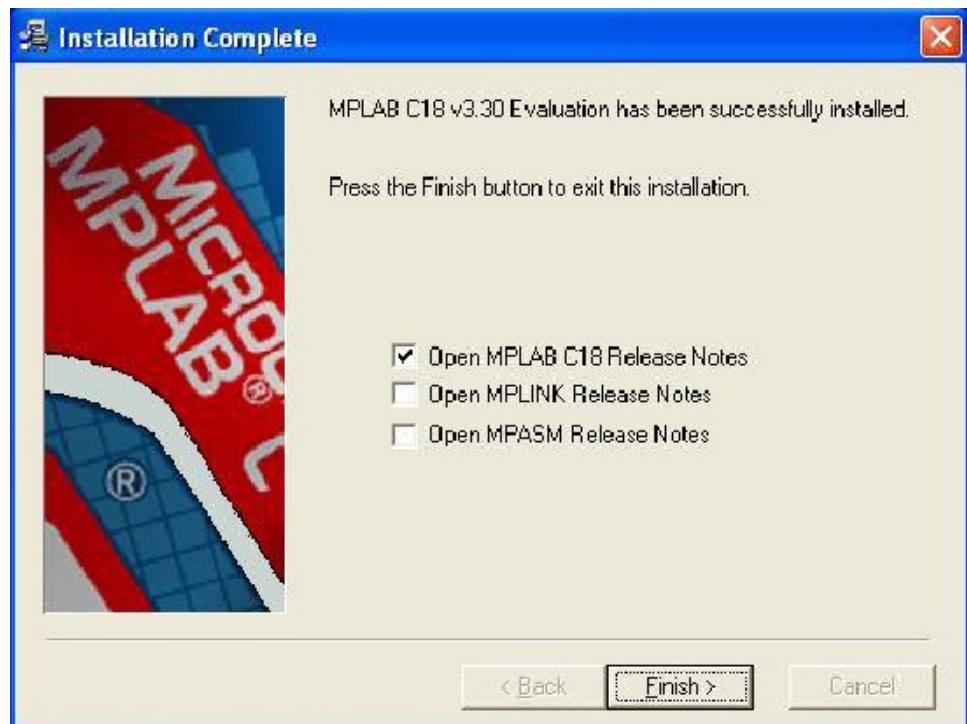
Una vez descargado ejecutar el instalador **MPLAB-C18-Evaluation-v3_30** versión actualmente disponible.

Para a instalación seguimos los siguientes pasos:





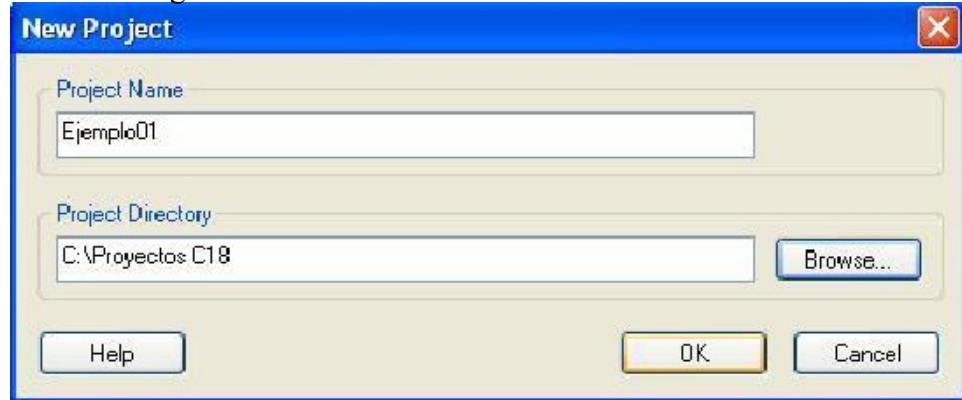




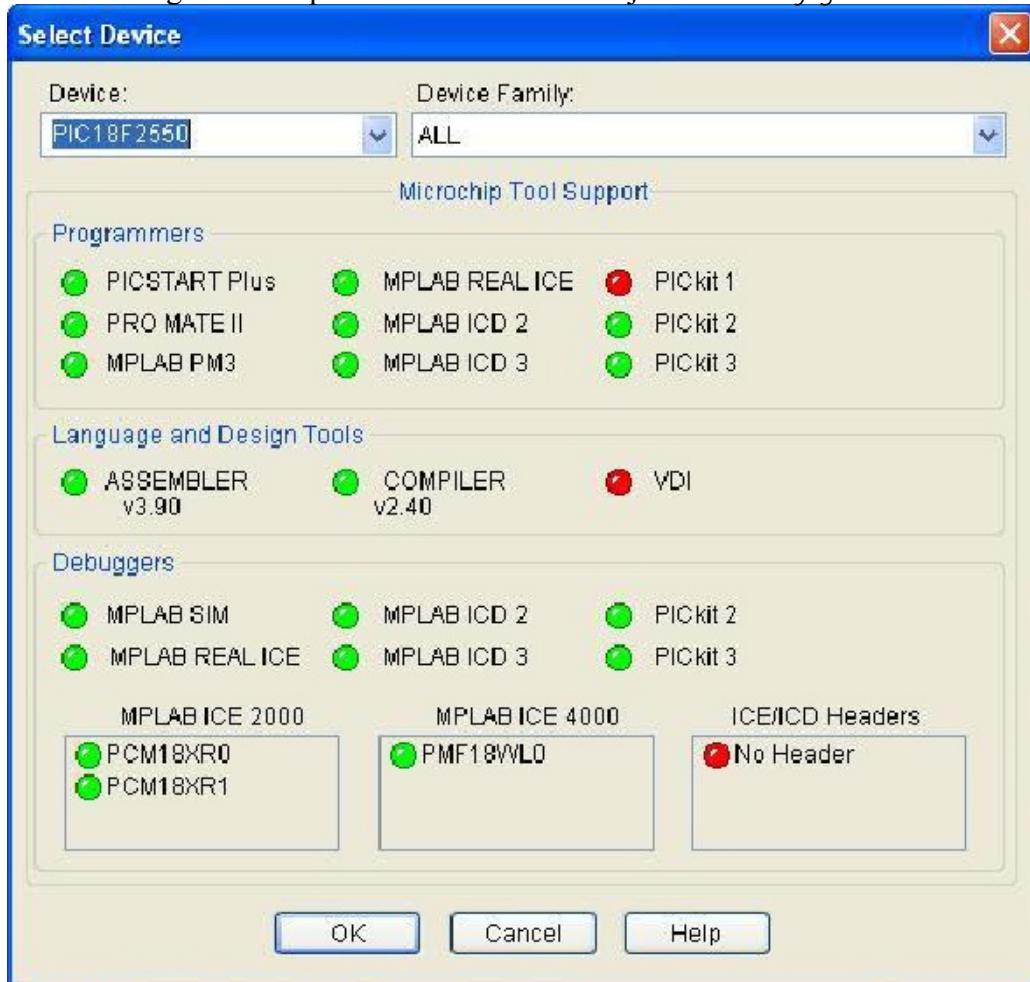
Creación de un nuevo proyecto.

Project-> New

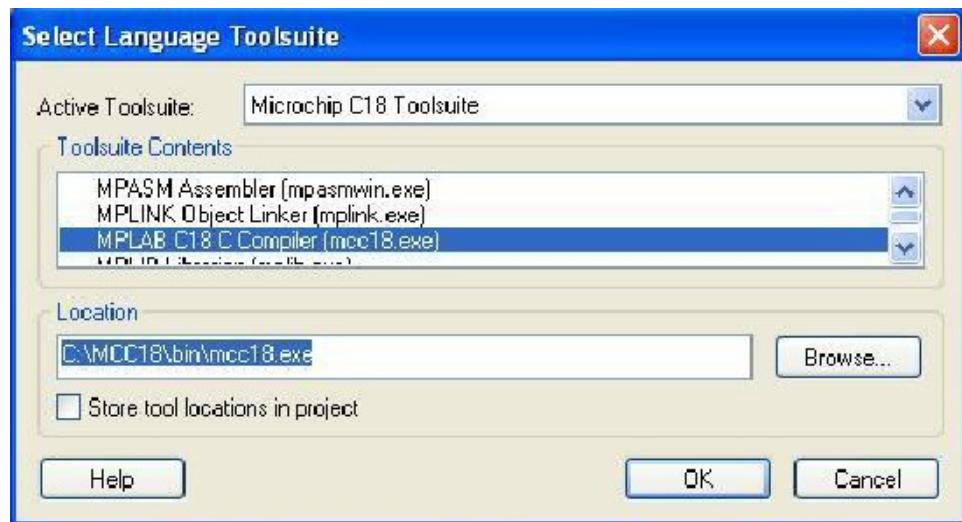
Nos aparecerá una pantalla donde le indicamos el nombre de nuestro proyecto y la carpeta donde será guardado.



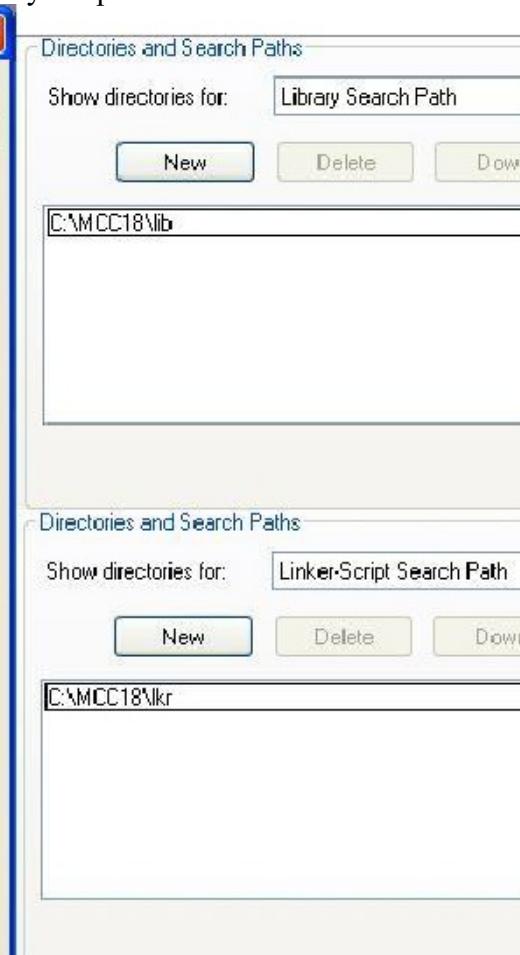
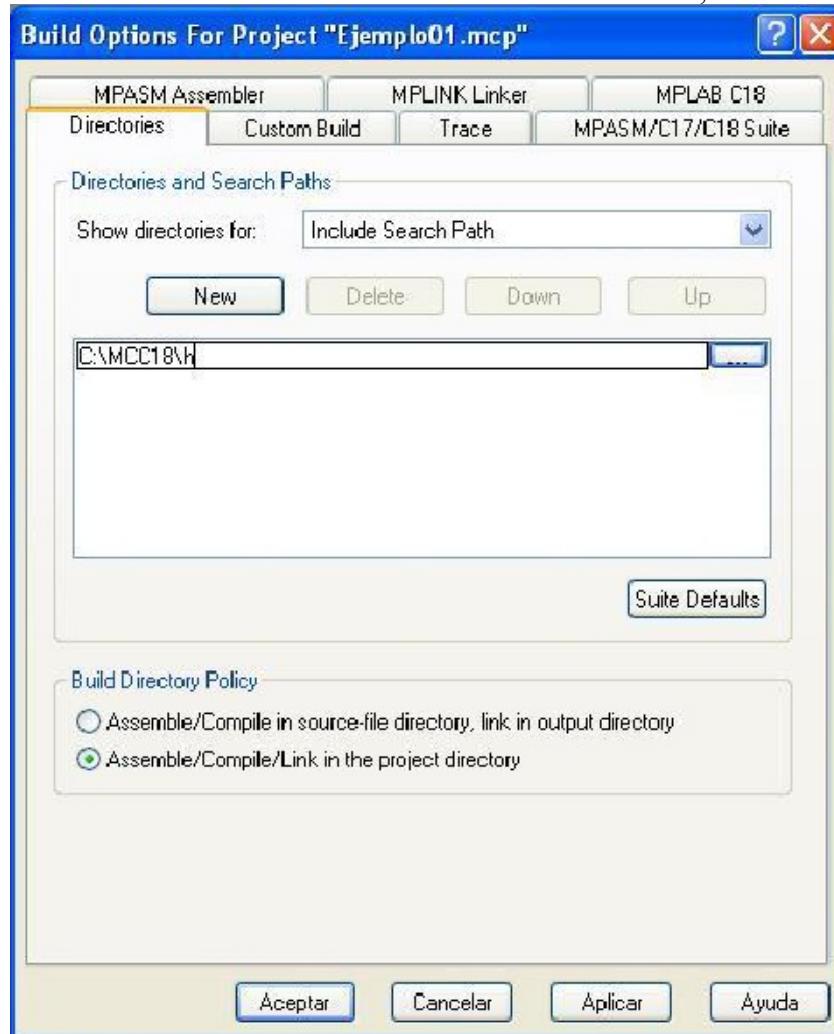
Pasamos a configurar el dispositivo con el cual trabajaremos: *Configure->Select Device*



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.



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

Introducción al 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.

Para cualquier desarrollo es importante leer el **datasheet** del PIC a utilizar. En mi caso voy a utilizar el **PIC18F2550** y [aquí](#) pueden encontrar una descripción en español del mismo.

Creando el código:

Primer que nada debemos incluir las librerías que se van a utilizar en el proyecto, la librería que siempre incluiremos será la del PIC a usar, la cual tiene las direcciones de los registros del mismo.

Código: (c)

```
#include <PIC18F2550.h>
```

Luego viene la configuración de los fuses del microcontrolador. Ósea 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\ hipPIC18ConfigSet

Definición de variables globales, son las que se utilizaran en el programa principal y funciones del proyecto.

Código: (c)

```
int variableA, variableB
```

Ahora viene el código de nuestro programa:

Código: (c)

```
main{
```

```
}
```

Primer ejemplo, control de leds con un pulsador.

Para este sencillo ejemplo vamos a necesitar definir operadores y estructuras de control:

Operadores:

Aquí definiremos todos los operadores utilizados por C18.-

Operadores de Comparación:

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

Operador	Descripción
<code>==</code>	Iguala
<code>!=</code>	Distinto a
<code><</code>	Menor que
<code>></code>	Mayor que
<code><=</code>	Menor o igual que
<code>>=</code>	Mayo o igual que

Operadores aritméticos:

Se utilizan para realizar cálculos matemáticos:

Operador	Descripción
<code>+</code>	Suma
<code>-</code>	Resta
<code>*</code>	Multiplicación
<code>/</code>	División
<code>++</code>	Incremento
<code>--</code>	decremento

Operadores lógicos:

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

Operador	Descripción
<code>&&</code>	AND
<code> </code>	OR
<code>!</code>	NOT

Operadores bitwise:

Son para modificar los bits de una variable:

Operador	Descripción
<code>&</code>	AND
<code> </code>	OR
<code>^</code>	XOR
<code>~</code>	Complemento
<code><<</code>	Rotar izquierda
<code>>></code>	Rotar derecha

Estructuras:***Estructura if:***

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

Código: (c)

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

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.

Código: (c)

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

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.

Código: (c)

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

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.

Código: (c)

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

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.

Código: (c)

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

Mas adelante explicaremos la estructura switch.

Accediendo a los bits de un registro:

Para acceder individualmente a los bits de un registro se escribe la siguiente sentencia:

Registrobits.bit

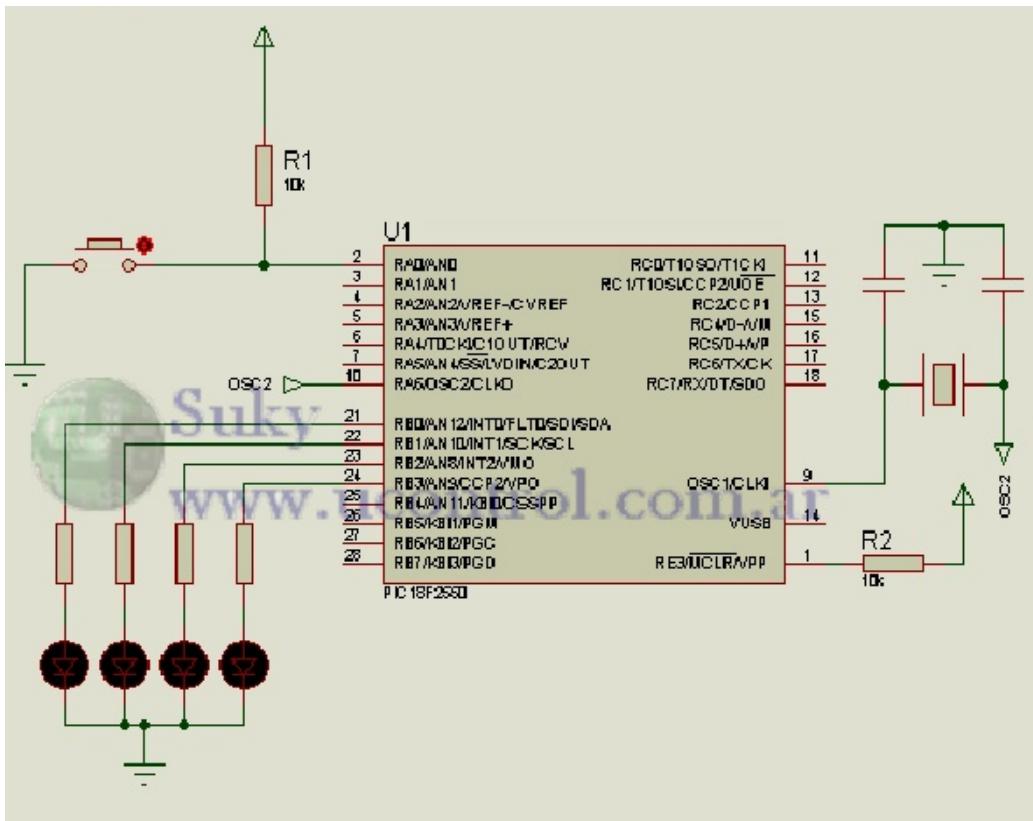
Ejemplo:

Código: (c)

```
PORTBbits.RB0
```

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

Hardware

**Código:****Código: (c)**

```
#include <p18f2550.h>
#pragma config FOSC = XT_XT, FCMEN = OFF, IESO = OFF, CPUDIV = OSC1_PLL2
#pragma config PWRT = ON, BOR = OFF, BORV = 0
#pragma config 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 = ON, CP1 = ON, CP2 = ON
#pragma config CPB = ON, CPD = ON
#pragma config WRT0 = ON, WRT1 = ON, WRT2 = ON
#pragma config WRTB = ON, WRTC = ON, WRTD = ON
#pragma config EBTR0 = ON, EBTR1 = ON, EBTR2 = ON
#pragma config EBTRB = ON

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

TRISA=0xFF; //Todos como entrada.-  

TRISB=0X00; //Todos como salida.-  

PORTB=0x00; //Leds apagados.-  

while(1){  

    if(PORTAbits.RA0==1){ // testeamos RB0.  

        PORTB=0x00; // Si esta en 1 Apagamos leds  

    }  

    else{  

        PORTB=0x0F; // Si esta en 0 Encendemos leds.  

    }  

}
}
```

Variables

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)

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.

Demoras:

Para utilizar demoras en nuestro código debemos incluir la librería delays.h. En ella tenemos 4 funciones:

Delay10TCYx(i) -> 10.Tcy.i
 Delay100TCYx(i) -> 100.Tcy.i
 Delay1KTCYx(i) -> 1000.Tcy.i
 Delay10KTCYx(i) -> 10000.Tcy.i
 Donde i puede tomar valores entre 0 y 255.

Ejemplo: Leds titilando.

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

Hardware: Idem anterior.

*Código:**Código: (c)*

```
#include <p18f2550.h>
#include <delays.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config WDT = OFF,WDPDS = 32768
#pragma config MCLRE = ON,LPT1OSC = OFF,PBADEN = OFF,CCP2MX = OFF
#pragma config STVREN = OFF,LVP = OFF,XINST = OFF,DEBUG = OFF
#pragma config CP0 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

unsigned char i; //Para contar 0 titilaciones.-

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

TRISA=0xFF; //Todos como entrada.-  

TRISB=0X00; //Todos como salida.-  

PORTB=0x00; //Leds apagados.-  

while(1){  

    if(PORTAbits.RA0==1){  

        PORTB=0x00;  

    }  

    else{  

        for(i=1;i<=10;++i){//Titila 10 veces  

            PORTB=0x0F;  

            Delay10KTCYx(30); //Demora 300ms  

            PORTB=0x00;  

            Delay10KTCYx(30); //Demora 300ms  

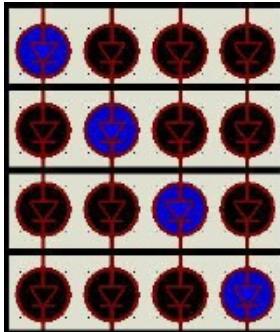
        }  

    }  

}
}
```

Ejemplo:Leds secuenciales:

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



Hardware: Idem anterior.

Código:**Código: (c)**

```
#include <p18f2550.h>
#include <delays.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF, CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

unsigned char i;           //Para contar 4 secuencias.-

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

TRISA=0xFF; //Todos como entrada.-  

TRISB=0X00;   //Todos como salida.-  

PORTB=0x00; //Leds apagados.-  

while(1){  

    if(PORTAbits.RA0==1){  

        PORTB=0x00;  

    }  

    else{  

        PORTB=0x01;  

        for(i=1;i<=4;++i){  

            Delay10KTCYx(30);  

            PORTB<<=1;  

        }  

    }  

}
}
```

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.

Código: (c)

```
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];
```

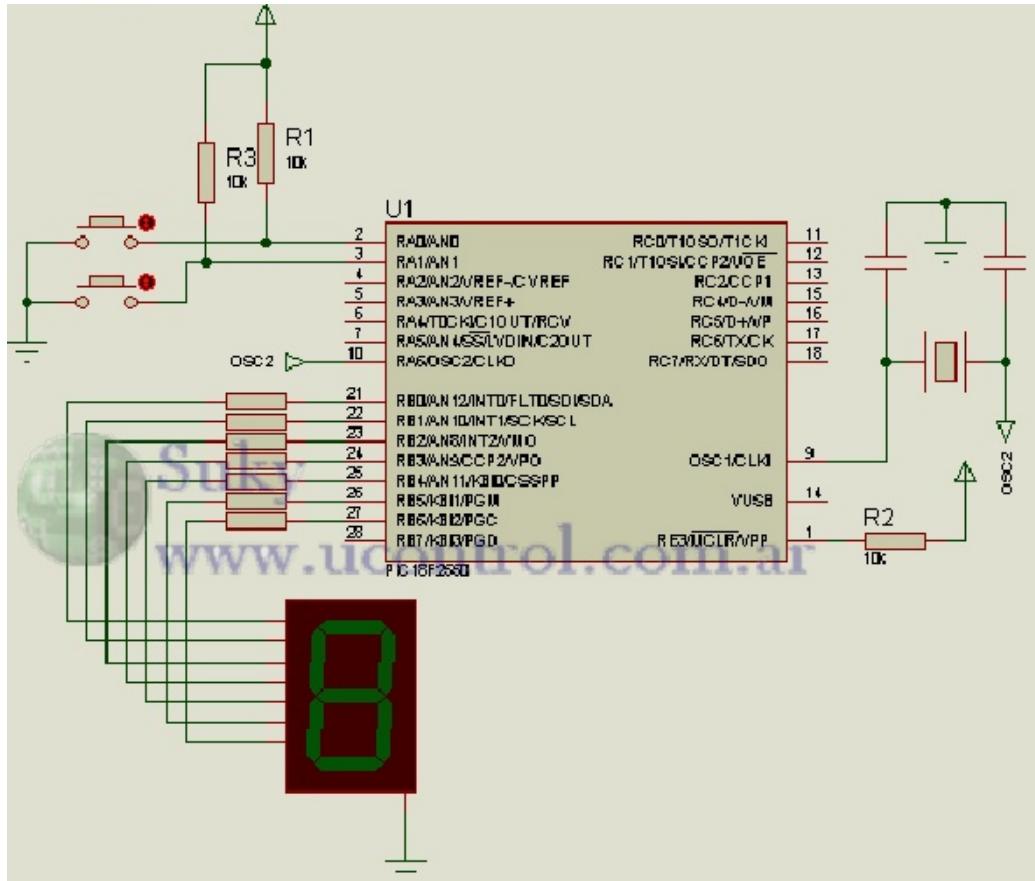
En la declaración se pueden pre cargar los valores de la siguiente forma:

Código: (c)

```
unsigned char Vector[3]={1,0x10,0b000101}
unsigned char Matriz[3][3]={1,2,3,4,5,6,7,8,9,10};
```

Ejemplo: Control de display 7 segmentos

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

Hardware:**Código:****Código: (c)**

```
#include <p18f2550.h>
#include <delays.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

unsigned char i; //Para controlar vizualización del Display.--.
unsigned char Display7Seg[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0xFF, 0x6F};

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

PORTB=0x3F; //Comienza en cero.-.
i=0;
while(1){
```

```
    if(PORTAbits.RA0==0 & PORTAbits.RA1==0){// Si se presionan los
2 a la vez se resetea.-
        i=0;
        PORTB=Display7Seg[i];
        Delay10KTCYx(30);
    }else if(PORTAbits.RA0==0){// Se incrementa cuenta.-
        ++i;
        if(i==10){i=0;}           //Volvemos a 0.-
        PORTB=Display7Seg[i];
        Delay10KTCYx(30);
    }else if(PORTAbits.RA1==0){// Se decrementa cuenta.-
        --i;
        if(i==255){i=9;} //Volvemos a 9.-
        PORTB=Display7Seg[i];
        Delay10KTCYx(30);
    }
}
```

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.

Sintaxis:

Código: (c)

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 vacío o se incluye la palabra void.

Ejemplos:

Código: (c)

```
unsigned int Suma(unsigned char A, unsigned char B) {
    unsigned int C;
    C=A+B;
    return(C);
}
```

Código: (c)

```
void Envio_Data(unsigned char A) {
    //Sentencia.-
```

}

Código: (c)

```
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.

Ejemplo:

Código: (c)

```
// Declaracion de la funcion
void Funcion(void);

.

.

.

void main(void) {

    .

    .

    // Llamo a la función.
    Funcion();
}
```

```
//Defincion de la funcion. (Puede estar en cualquier lugar del
programa)
void Funcion(void){
    // 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.

Ejemplo:

Código: (c)

```
.

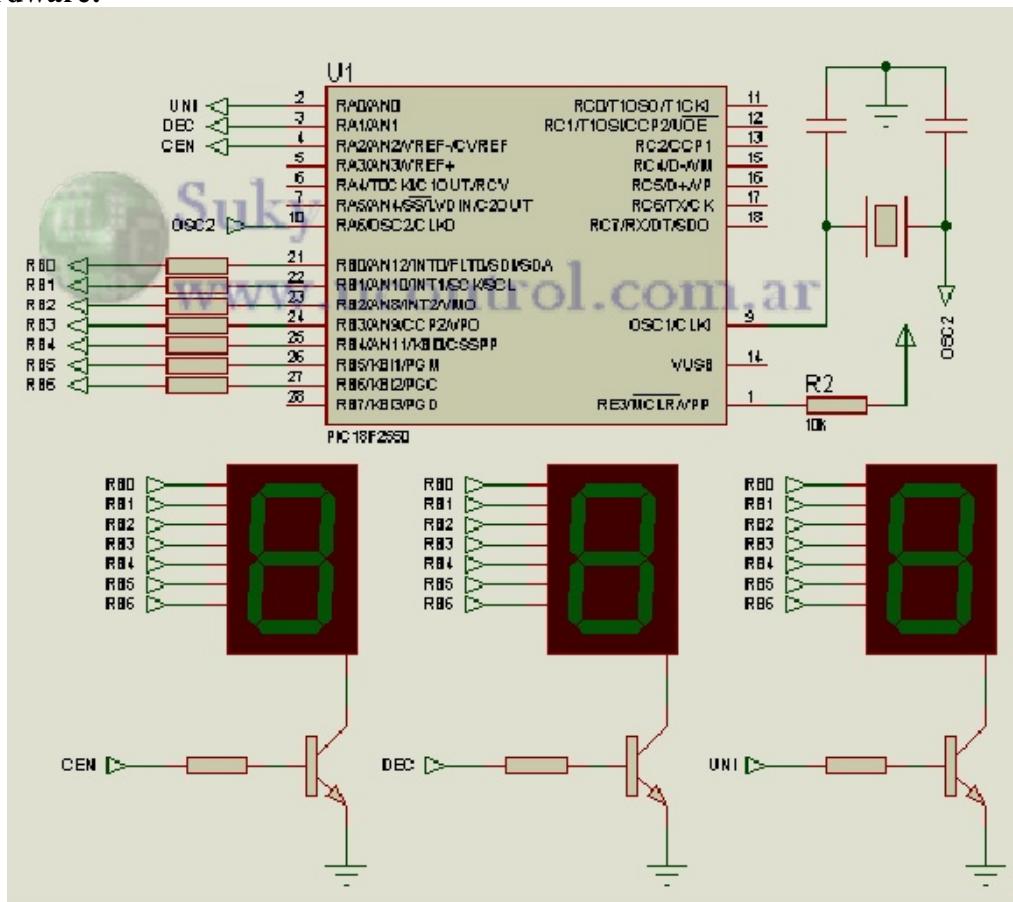
//Defincion de la funcion
void Funcion(void){
    // Sentencias
}
void main(void){

.

// Llamo a la función.
Funcion();
}
```

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

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

Hardware:**Código:****Código: (c)**

```
#include <p18f2550.h>
#include <delays.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

#define UnidadBit PORTAbits.RA0
#define DecenaBit PORTAbits.RA1
#define CentenaBit PORTAbits.RA2

unsigned char i, Unidad, Decena, Centena;
unsigned char Display7Seg[10]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0xFF, 0x6F};
void Visualizacion (void);

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

TRISA=0xF0; //Todos como entrada.-
```

```
TRISB=0X00;      //Todos como salida.-  
  
PORTA=0x00;  
Unidad=0;  
Decena=0;  
Centena=0;  
  
while(1){  
    Visualizacion();  
    ++Unidad;  
    if(Unidad==10){  
        Unidad=0;  
        ++Decena;  
        if(Decena==10){  
            Decena=0;  
            ++Centena;  
        }  
    }  
}  
void Visualizacion (void){  
    for(i=1;i<=20;++i){  
        PORTB=Display7Seg[Unidad];  
        UnidadBit=1;    //Enciendo Display Unidad.-  
        Delay1KTCYx(5); //Demora de 5 ms (XT=4MHz)  
        UnidadBit=0;  
        PORTB=Display7Seg[Decena];  
        DecenaBit=1;  
        Delay1KTCYx(5);  
        DecenaBit=0;  
        PORTB=Display7Seg[Centena];  
        CentenaBit=1;  
        Delay1KTCYx(5);  
        CentenaBit=0;   //Apago Display Centena.-  
    }  
}
```

Control de un LCD.

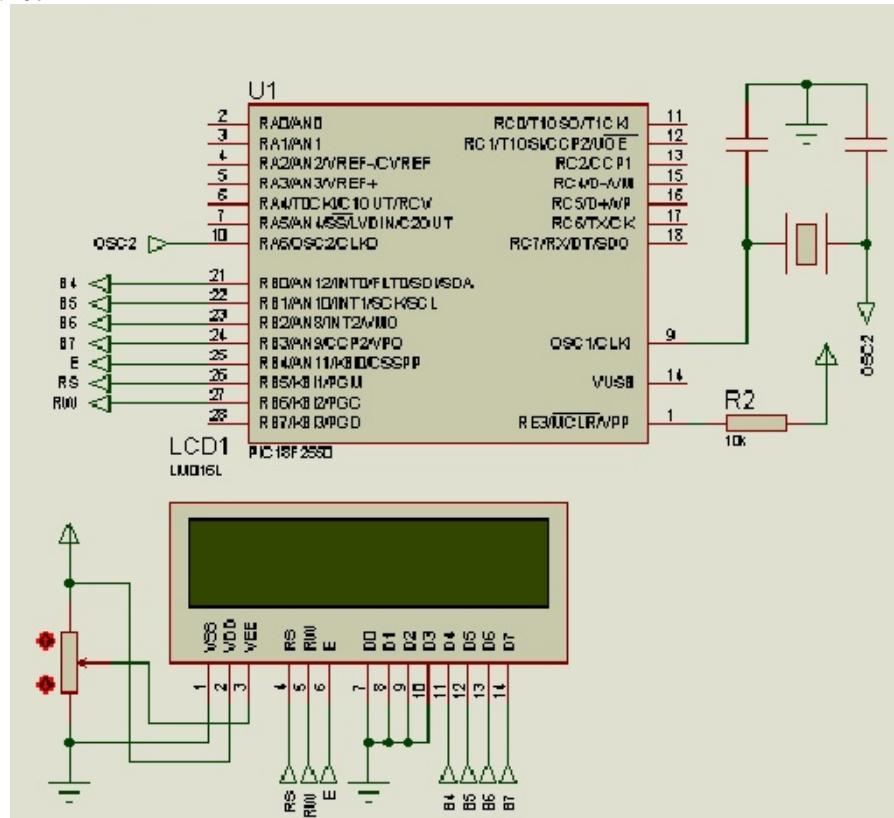
Para realizar el control de un LCD necesitamos usar la librería xlcd.h ubicada en C:\MCC18\h\. Esta librería es para un LCD con controlador Hitachi HD44780 o compatible, de 8 o 4 bits. El usuario debe proveer 3 delay para el correcto funcionamiento, DelayPORXLCD() de 15ms, DelayXLCD() de 5ms y DelayFor18TCY() de 18 Tcy.

En este caso no vamos a modificar la librería, pues lo vamos a controlar con el puerto B, pero en el caso de que se modifique sugiero siempre respaldarla con una copia de seguridad.

Ejemplo: Control de un LCD.

Objetivo: Vamos a escribir un simple mensaje en un LCD. Se crearon 2 funciones adicionales para un mejor el control, la primera 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: (c)

```
#include <p18f2550.h>
#include <delays.h>
#include <xlcd.h>

#pragma config FOSC = XT_XT, FCMEN = OFF, IESO = OFF, CPUDIV = OSC1_PLL2
#pragma config PWRT = ON, BOR = OFF, BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

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 = nº 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(0x0C);                         // Encendemos LCD.-.
    putrsXLCD("Probando LCD");                //Pasamos al orden del
Linea 2.-.
    putrsXLCD("Por Suky");
    while(1){                                  // Bucle infinito.
    }
}

```



Conversión analógica digital.

Para realizar una conversión analógica digital usaremos la librería adc.h, esta tiene las funciones necesarias para las configuraciones y lectura de dicho módulo. En el caso de las configuraciones, este posee varias y depende del microcontrolador que se use. Para saber en que grupo cae nuestro microcontrolador abrimos pconfig.h, y por ejemplo para el PIC18F2550 es ADC_V5.

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:

Código: (c)

```
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.

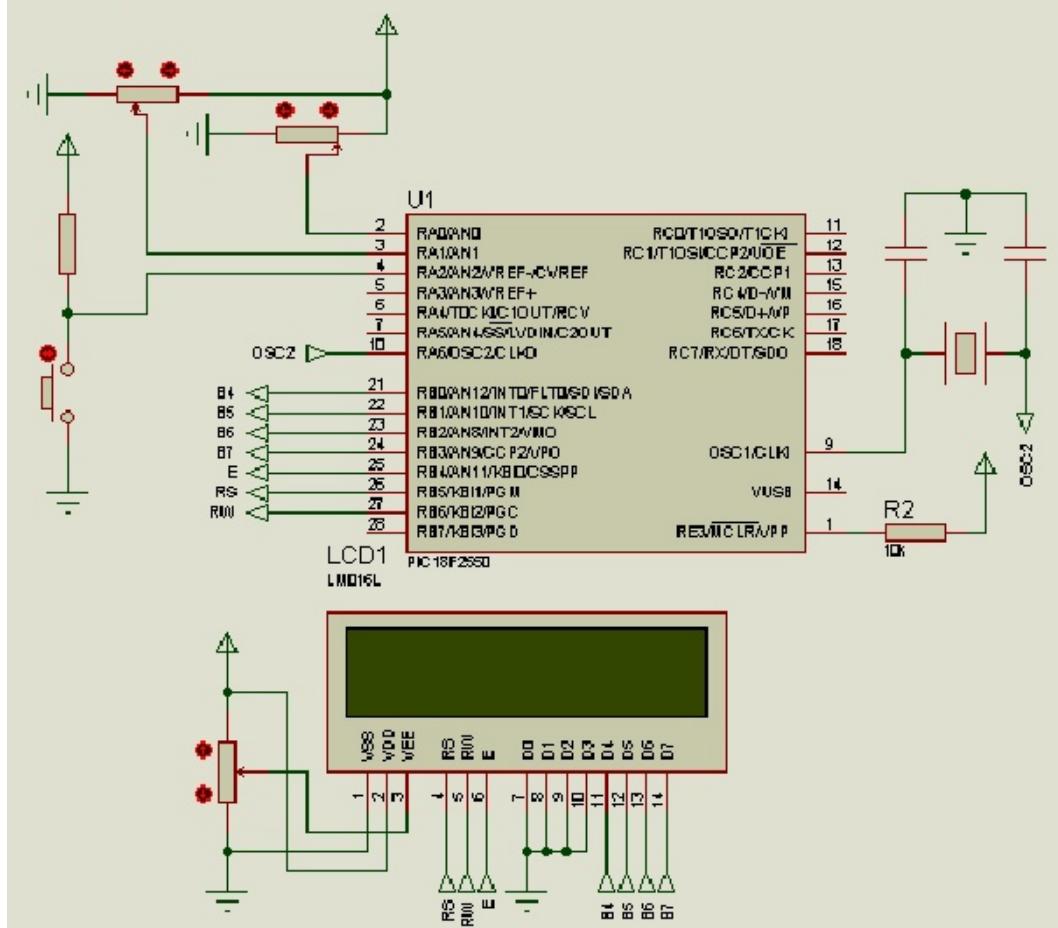
SelChanConvADC(Unsigned char); Selecciona canal y comienza conversión.

BusyADC(); Comprueba si la conversión a finalizado.

Int ReadADC(); devuelve la lectura del canal analógico seleccionado.

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: (c)**

```

#include <p18f2550.h>
#include <delays.h>
#include <xlcdb.h>
#include <stdlib.h> //Libreria para conversiones.
#include <adc.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

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.-.
    OpenADC(ADC_FOSC_RC & ADC_8_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(2); // 20us para que se cargue el capacitor
sample&hold.-.
        ConvertADC();                          // Comienza conversión.-.
        While(BusyADC()==1){} // Hasta que se finalice
conversion.-.
        Canal0= ReadADC();                   // Realizo lectura.-.

        SetChanADC(1); // Selecciono canal a convertir.-.
        Delay10TCYx(2); // 20us para que se cargue el capacitor
sample&hold.-.
        ConvertADC();                          // Comienza conversión.-.
        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);
    }
}
```



Interrupciones:

Los dispositivos PIC18 tienen múltiples fuentes de interrupción y una 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 está 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 está 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.

Rutinas de atención a interrupciones:

La directiva `#pragma interrupt low nombre` define rutina de servicio de interrupción (ISR) de baja prioridad y `#pragma interrupt nombre` se 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 den declarar como volatile

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

Código: (c)

```
#pragma interrupt low ISR_BajaPrioridad
Void ISR_BajaPrioridad(void){
    // Tratamiento de interrupción.-}

// Creamos una nueva sección de código a partir de la dirección 0x18.-
#pragma code PrioridadBaja = 0x18
Void Vector_BajaPrioridad(void){
    // Instrucción insertada en la dirección 0x18.-
    _asm goto ISR_BajaPrioridad _endasm
}
#pragma code // Cerramos sección.-
```

Código: (c)

```

#pragma interrupt ISRAltaPrioridad
Void ISRAltaPrioridad(void) {
    // Tratamiento de interrupción.-.
}

// Creamos una nueva sección de código a partir de la dirección 0x08.-
#pragma code PrioridadAlta = 0x08
Void VectorAltaPrioridad(void) {
    // Instrucción insertada en la dirección 0x08.-.
    _asm goto ISRAltaPrioridad _endasm
}
#pragma code // Cerramos sección.-.

```

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.

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);* Envía un string desde la memoria de datos.

*putrsUSART(const rom char *data);* Envía 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.

Estructura switch.

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

Código: (c)

```
switch(Variable) {
    Case 0x01:
        //Sentencias.-  

        Break;
    Case 0x02:
        //Sentencias.-  

        Break;
    Default:
        //Sentencias.-  

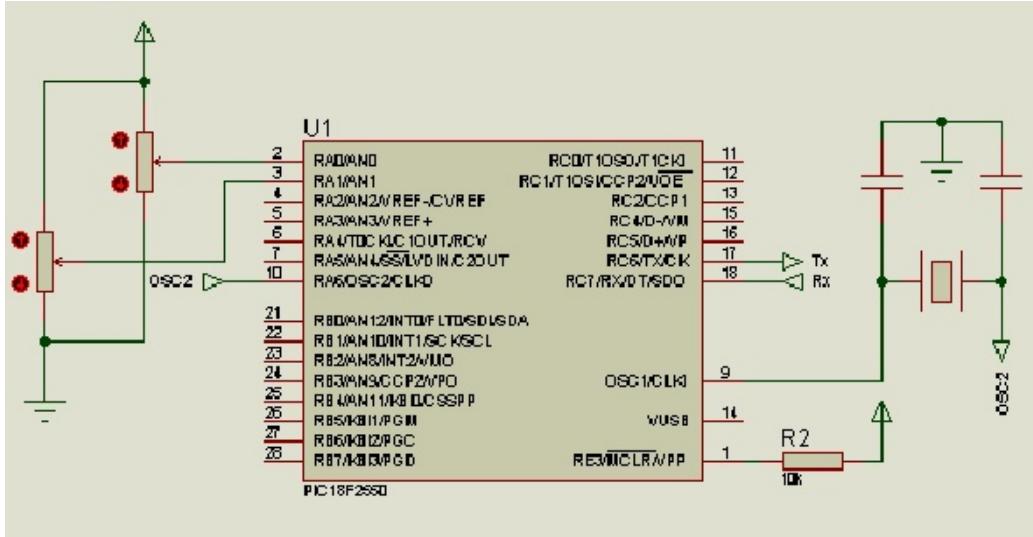
        Break;
}
```

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

Ejemplo: Comunicación Serial RS232.

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:



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

Código: (c)

```
#include <p18f2550.h>  

#include <delays.h>  

#include <stdlib.h> //Libreria para conversiones.  

#include <adc.h>  

#include <usart.h>  

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF, CPUDIV = OSC1_PLL2  

#pragma config PWRT = ON,BOR = OFF,BORV = 0  

#pragma config 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 = ON, CP1 = ON, CP2 = ON
#pragma config CPB = ON, CPD = ON
#pragma config WRT0 = ON, WRT1 = ON, WRT2 = ON
#pragma config WRTB = ON, WRTC = ON, WRTD = ON
#pragma config EBTR0 = ON, EBTR1 = ON, EBTR2 = ON
#pragma config EBTRB = ON

void ISRRecepcion(void);
volatile char Data, Kbhit;

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

// 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, Canall;
    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_8_TAD & ADC_RIGHT_JUST,
ADC_REF_VDD_VSS & ADC_INT_OFF, ADC_2ANA);
    Kbhit=0;
    RCONbits.IPEN=0;           // Desabilitamos Prioridades
    INTCONbits.PIE=1;          // Habilitamos interrupción de
perifericos.-          INTCONbits.GIE=1;           // Habilitamos interrupción
Global.

    putrsUSART("Prueba Comunicación 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(2); // 20us para que se
cargue el capacitor sample&hold.-          ConvertADC();           // Comienza
conversión.-          While(BusyADC ()==1){} // Hasta que se
finalice conversión.-          Canal0= ReadADC(); // Realizo
    }
}

```

```

lectura.-

Convertimos entero a string.-

        putrsUSART("Canal 0 = ");
        itoa(Canal0, String);           // 

cargue el capacitor sample&hold.-

        putsUSART(String);
        putrsUSART("\r");
        break;
    case 0x62:      // letra b
        SetChanADC(1); // Selecciono canal a
convertir.-

        Delay10TCYx(2); // 20us para que se

carga el capacitor sample&hold.-

        ConvertADC();                // Comienza
conversión.-

        While(BusyADC ()==1){} // Hasta que se

finalice conversión.-

        Canall= ReadADC();         // Realizo

lectura.-

        putrsUSART("Canal 1 = ");
        itoa(Canall, String);       //

Convertimos entero a string.-

        putsUSART(String);
        putrsUSART("\r");
        break;
    case 0x63:      // letra c
        SetChanADC(0); // Selecciono canal a
convertir.-

        Delay10TCYx(2); // 20us para que se

carga el capacitor sample&hold.-

        ConvertADC();                // Comienza
conversión.-

        While(BusyADC ()==1){} // Hasta que se

finalice conversión.-

        Canal0= ReadADC();         // Realizo

lectura.-

        SetChanADC(1); // Selecciono canal a
convertir.-

        Delay10TCYx(2); // 20us para que se

carga el capacitor sample&hold.-

        ConvertADC();                // Comienza
conversión.-

        While(BusyADC ()==1){} // Hasta que se

finalice conversión.-

        Canall= ReadADC();         // Realizo

lectura.-

        putrsUSART("Canal 0 = ");
        itoa(Canal0, String);       //

Convertimos entero a string.-

        putsUSART(String);
        putrsUSART("\r");
        putrsUSART("Canal 1 = ");
        itoa(Canall, String);       //

Convertimos entero a string.-

        putsUSART(String);
        putrsUSART("\r");
        break;
    }
}

```

Otra manera de enviar los datos, usando printf:

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:

Código: (c)

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

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:

Código: (c)

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

Por:

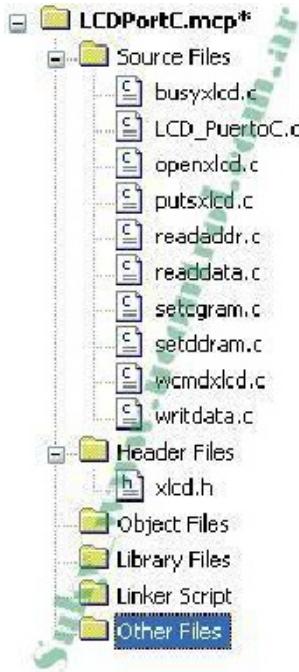
Código: (c)

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

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), excepto putrxlcd, quedando de la siguiente manera:



De esta manera se recompilan los archivos, creando sus respectivos archivos objetos (*.o) necesarios para el lindeo (MPLink) y así efectivizar los cambios realizados en las rutinas.

Vamos a modificar xlcd.h para que utilizar el Puerto C para controlar el LCD y los 4 bits más significativos del mismo (**UPPER**) para enviar los datos al LCD de la siguiente manera:

Código: (c)

```
#define UPPER
/* DATA_PORT defines the port to which the LCD data lines are
connected */
#define DATA_PORT      PORTC
#define TRIS_DATA_PORT TRISC
/* CTRL_PORT defines the port where the control lines are connected.
 * These are just samples, change to match your application.
 */
#define RW_PIN    LATCbits.LATC1 /* PORT for RW */
#define TRIS_RW   TRISCBits.TRISC1 /* TRIS for RW */
#define RS_PIN    LATCbits.LATC0 /* PORT for RS */
#define TRIS_RS   TRISCBits.TRISC0 /* TRIS for RS */
#define E_PIN     LATCbits.LATC2 /* PORT for D */
#define TRIS_E    TRISCBits.TRISC2 /* TRIS for E */
```

Cualquier cambio, por mínimo que sea se debe hacer de esta manera para que se efectivicen los cambios.

Código: (c)

```

#include <p18f2550.h>
#include <delays.h>
#include <xlc.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

void DelayFor18TCY(void) {
    Delay10TCYx(2);
}
void DelayPORXLCD(void) {
    Delay1KTCYx(15);
}
void DelayXLCD(void) {
    Delay1KTCYx(2);
}

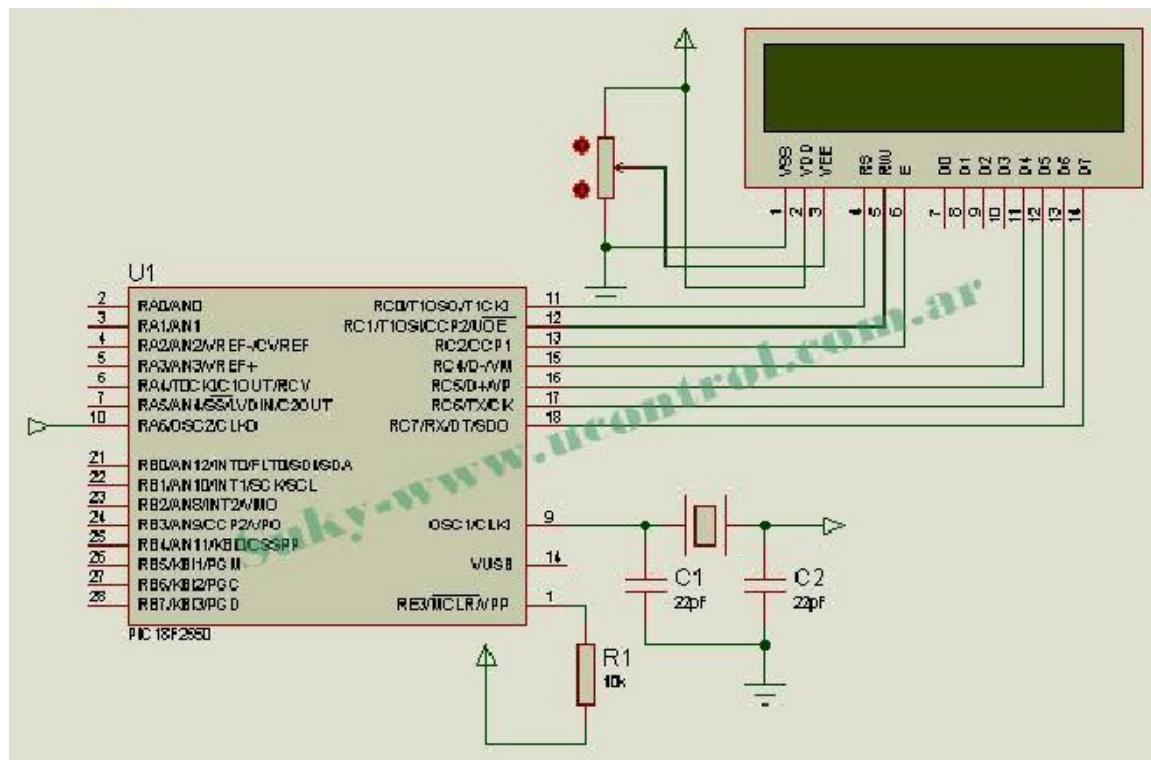
// Envía comando al LCD
void comandXLCD(unsigned char a){
    BusyLCD();
    WriteCmdLCD(a);
}
// Ubica cursor en (x = Posicion en linea, y = nº de linea)
void gotoxyXLCD(unsigned char x, unsigned char y){
    unsigned char direccion;

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

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

void main(void) {
    OpenLCD(FOUR_BIT & LINES_5X7);           // Iniciamos LCD.-.
    comandXLCD(0x0C);                         // Encendemos LCD.-.
    putrsLCD("LCD en Puerto C");              // Pasamos al orden del
    Linea 2.-.
    putrsLCD("Por Suky");
    while(1){                                  // Bucle infinito.
    }
}

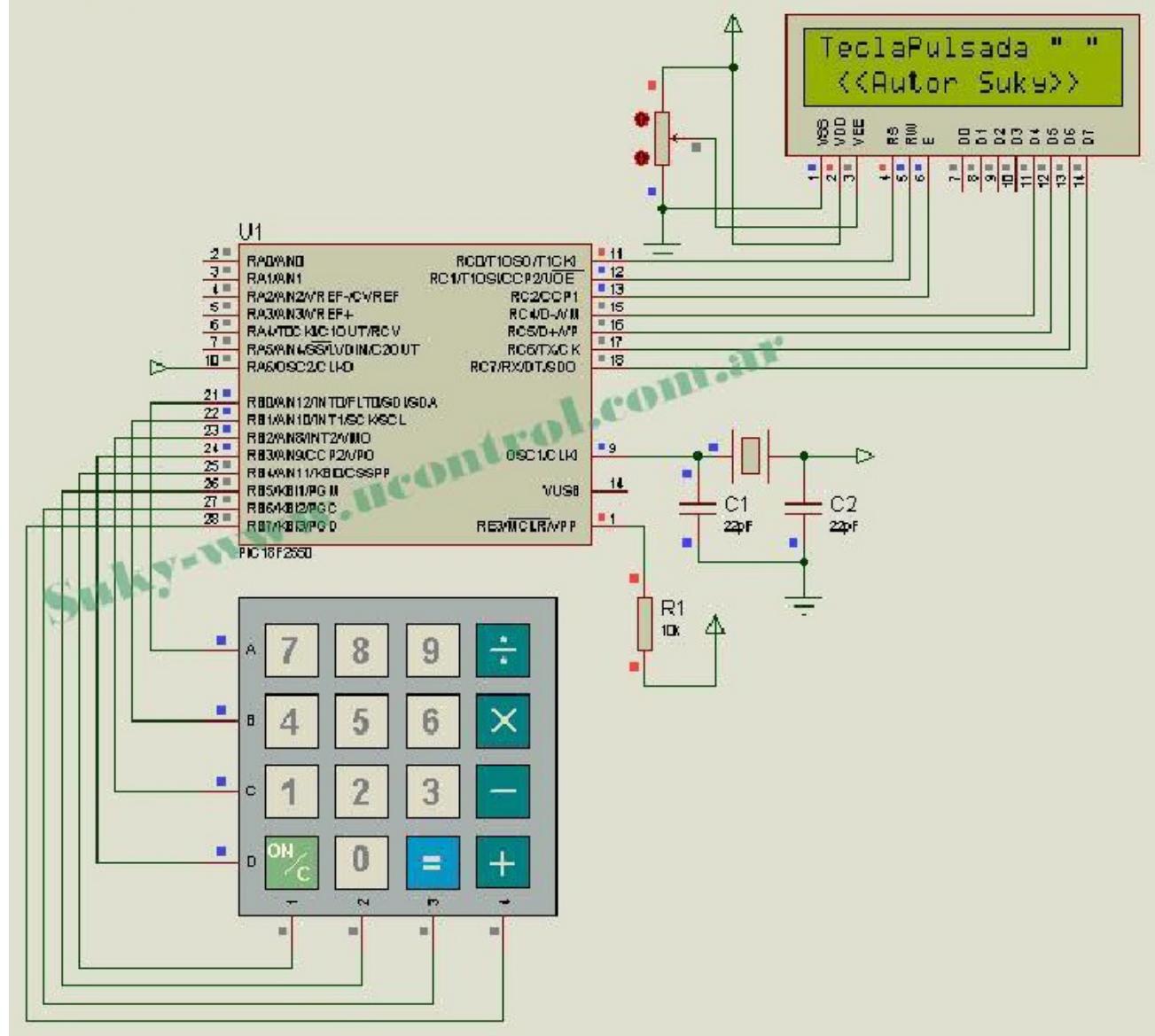
```



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

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.

Hardware:



Nota: En Proteus se deben agregar las resistencias pull-up a RB4-RB7 para simular correctamente. Además se usa el puerto C para el control del LCD, y para que la compilación sea correcta se deben seguir los pasos del mensaje anterior.

Código: (c)

```
#include <p18f2550.h>
#include <delays.h>
#include <xlc.h>

#pragma config FOSC = XT_XT, FCMEN = OFF, IESO = OFF, CPUDIV = OSC1_PLL2
```

```

#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config WDT = OFF,WDPDS = 32768
#pragma config MCLRE = ON,LPT1OSC = OFF,PBADEN = OFF,CCP2MX = OFF
#pragma config STVREN = OFF,LVP = OFF,XINST = OFF,DEBUG = OFF
#pragma config CP0 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON
unsigned char Tecla;
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 sección de código a partir de la dirección 0x08.- 
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRRB4_RB7 _endasm
}
#pragma code // Cerramos sección.-

//***** 
void main(void){
    OpenXLCD(FOUR_BIT & LINES_5X7);           // Iniciamos LCD.- 

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

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

//***** 
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 = nº 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 lógico 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++;
        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){
        Tecla=TestTeclado(); //Realizamos test del Teclado, y
nos devuelve tecla pulsada
        gotoxyXLCD(15,1);
        putcXLCD(Tecla); // Visualizamos tecla pulsada.
        while(PORTBbits.RB4==0 | PORTBbits.RB5==0
| PORTBbits.RB6==0 | PORTBbits.RB7==0){} //Esperamos a que se suelte.-.
        PORTB=0x00;
        INTCONbits.RBIF = 0; //Borramos bandera
    }
}

```

Como crear una librería.

Vamos a crear una librería para el control del CI ds1302, para ello necesitamos leer el datasheet del mismo ([DS1302](#)). Necesitamos crear el archivo .h que contendrá todas las definiciones de los pines y funciones a utilizar, y el/los archivo/s fuente/s con el código de nuestra librería.

Se debe definir la librería, esto se realiza colocando las líneas, por ejemplo para nuestro caso:

Código: (c)

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

Quedando ds1302.h:

Código: (c)

```
/** \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 <p18cxx.h>

//Definicion de los pines a utilizar.
#define SCLK PORTBbits.RB0           /**< Pin de Clock del DS1302*/
#define IO      PORTBbits.RB1           /**< Pin de Data del DS1302*/
#define RST     PORTBbits.RB2           /**< Pin de Enable del DS1302*/
// Definicion de los bits del Tris para configurar en modo escritura o
lectura.
#define TrisSCLK      TRISBbits.TRISB0      /**< Tris del pin Clock*/
#define TrisIO        TRISBbits.TRISB1      /**< Tris del pin Data*/
#define TrisRST        TRISBbits.TRISB2      /**< Tris del pin
Enable*/
/** \brief Función que envía el byte por el Pin seleccionado.
 */
* \param Byte Valor a enviar al DS1302.
*/
void write_byte( unsigned char Byte);
/** \brief Envía los Datos al DS1302 según 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 Función 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 leído del DS1302.
*/
unsigned char read_ds1302( unsigned char Byte);
/** \brief Inicialización del DS1302
*/
void ds1302_init(void);
/** \brief Conversión 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 conversión.
*/
unsigned char get_bcd( unsigned char Data);
/** \brief Conversión del valor leído 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.
* \return Byte obtenido de la conversión.
*/
unsigned char rm_bcd( unsigned char Data);
/** \brief Función que carga al DS1302 valores de Hora y Fecha
seleccionados por el usuario.
*
* \param day Día de la semana, valor entre 1 y 7.
* \param mth Mes, valor entre 1 y 12.
* \param year Año, valor entre 00 y 99.
* \param dow Fecha, valor entre 1 y 31
* \param hr Hora, Según como se envíe el dato puede ser entre 1 y 12 o
0 y 23. El último nuestro caso.
* \param min Minutos, valor entre 0 y 59.
* \param sec Segundos, directamente se envía 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 según 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
dependerá del oscilador que se este usando.
*/
extern void Delay_2us(void);
#endif

```

Luego queda crear el código de nuestra librería, esto se puede hacer creando un archivo para cada función como lo realiza Microchip o uno solo que contenga todas las funciones. En

nuestro caso hacemos este ultimo caso, donde al principio se debe invocar el archivo .h que contiene las definiciones:

Código: (c)

```
#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 NibbleH,NibbleL;
    NibbleH=Data/10;
    NibbleL=Data-(NibbleH*10);
    NibbleH<<=4;
    return(NibbleH|NibbleL);
}

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)));
}

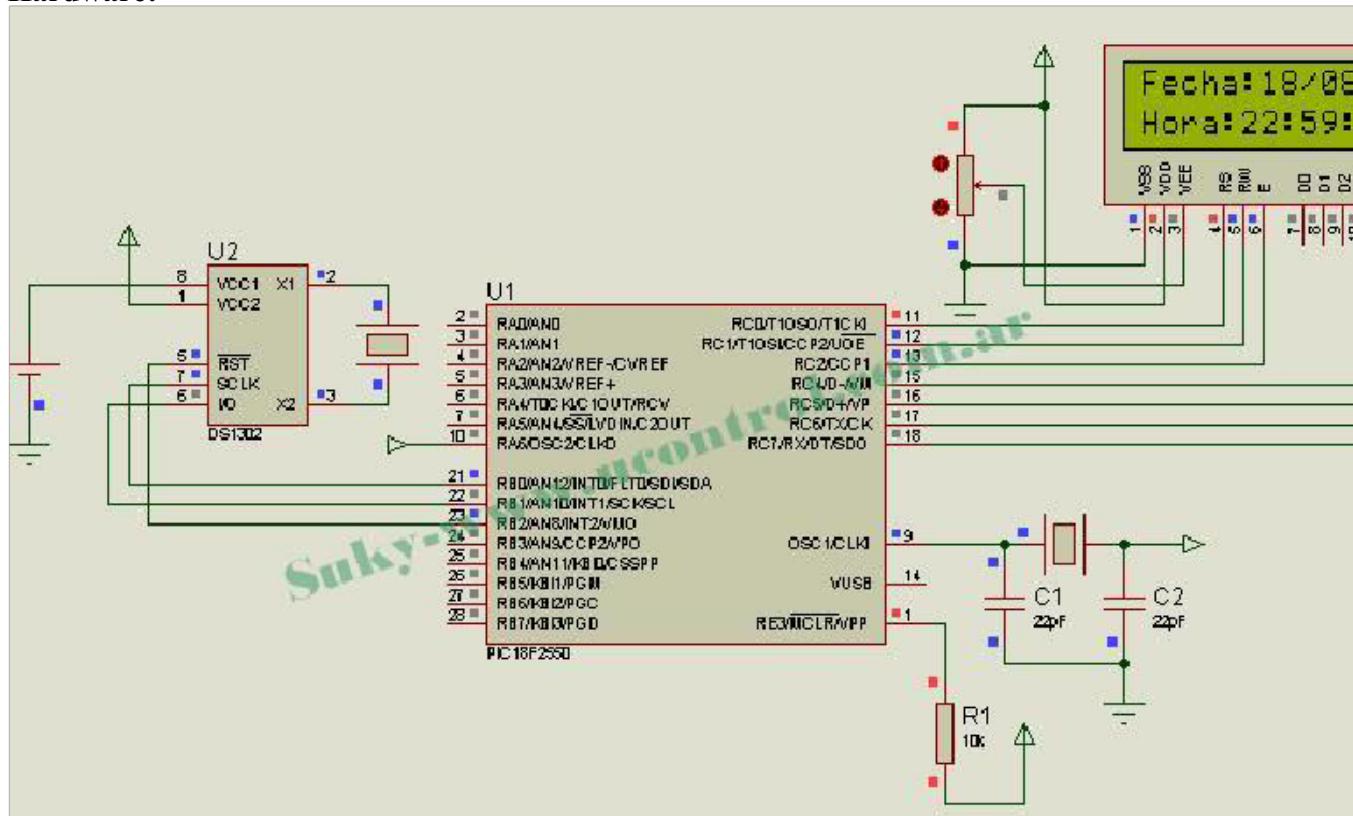
```

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.

Objetivo: Crear un Reloj/Calendario usando el DS1302 y LCD, para ello utilizaremos la librería creada, y para la actualización de los datos la interrupción por desbordamiento del Timer0 cada 1 segundo.

Hardware:



Configuración de los Timers

Para determinar las funciones y parámetros que necesitamos para configurar el Timer seleccionado vamos a *C:\MCC18\doc\periph-lib* y abrimos **Timer**. 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	Interrupción habilitada
TIMER_INT_OFF	Interrupción 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, Preescalador 1:16 y habilitación de la interrupción:

Código: (c)

```
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:

Código: (c)

```
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.

Código: (c)

```
#include <p18f2550.h>
#include <delays.h>
#include <xlcd.h>
#include <ds1302.h>
#include <stdlib.h> //Libreria para conversiones.
#include <timers.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
```

```

#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

unsigned char day,mth,year,dow,hour,min,sec; //variables para ds1302

//
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 sección de código a partir de la dirección 0x08.- 
#pragma code Interrupcion = 0X0008
void VectorInterrupcion(void){
    _asm goto ISRTimer0 _endasm
}
#pragma code // Cerramos sección.-

// Rutina de Interrupcion.- 
// Esta interrupción se ejecuta cada 1s.
#pragma interrupt ISRTimer0
void ISRTimer0(void){
    if(INTCONbits.TMR0IF==1){
        WriteTimer0(3036);           // Cargamos nuevamente
        Timer0 para interrupción cada 1seg.
        get_date();    //Lee dia,mes,año
        get_time();    //lee hora,minuto,segundo
        gotoxyXLCD(1,1);
        printDate(dow,mth,year);   //lcd
        gotoxyXLCD(1,2);
        printTime(hour,min,sec);
        INTCONbits.TMR0IF=0;      // Borramos bandera.- 
    }
}

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

    ds1302_init();                      //inicializa
ds1302.-
    OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT &
T0_PS_1_16); //Configuración del Timer1
    OpenXLCD(FOUR_BIT & LINES_5X7);      // Iniciamos LCD.- 

    comandXLCD(0x0C);                  // Encendemos LCD.- 
    putrsXLCD("Usando el ds1302");
}

```

```

        gotoxyLCD(1,2);
        putrsLCD("by Suky");
        day=7; mth=8; year=9;dow=18;
        hour=22; min=59;
        set_datetime(day,mth,year,dow,hour,min);
        Delay10KTCYx(100);
        comandLCD(1); // Borramos display

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

        while(1){
        }

//***** *****
void DelayFor18TCY(void) {
    Delay10TCYx(2);
}
void DelayPORLCD(void){
    Delay1KTCYx(15);
}
void DelayLCD(void){
    Delay1KTCYx(2);
}

// Envia comando al LCD
void comandLCD(unsigned char a){
    BusyLCD();
    WriteCmdLCD(a);
}
// Ubica cursor en (x = Posicion en linea, y = n° de linea)
void gotoxyLCD(unsigned char x, unsigned char y){
    unsigned char direccion;

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

    direccion += x-1;
    comandLCD(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];

```

```
putrsLCD("Fecha:");
itoa(dow, String); // Convertimos entero a string.-
if(dow<10){putrsLCD("0");}
putsLCD(String);putrsLCD("/");
itoa(mth, String); // Convertimos entero a string.-
if(mth<10){putrsLCD("0");}
putsLCD(String);putrsLCD("/");
itoa(year, String); // Convertimos entero a string.-
if(year<10){putrsLCD("0");}
putsLCD(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];

    putrsLCD("Hora:");
    itoa(hour, String); // Convertimos entero a string.-
    if(hour<10){putrsLCD("0");}
    putsLCD(String);putrsLCD(":");
    itoa(min, String); // Convertimos entero a string.-
    if(min<10){putrsLCD("0");}
    putsLCD(String);putrsLCD(":");
    itoa(sec, String); // Convertimos entero a string.-
    if(sec<10){putrsLCD("0");}
    putsLCD(String);
}
// Funcion que realiza la lectura de los todos los datos de la Fecha.
void get_date(void){ //Lee dia,mes,año
    dow=get_dow();
    mth=get_mth();
    year=get_year();
    dow=get_dow();
}
// Funcion que realiza la lectura de los todos los datos de la Hora.
void get_time(void){ //lee hora,minuto,segundo
    hour=get_hr();
    min=get_min();
    sec=get_sec();
}
```

Comunicación I2C.

Para saber en que grupo de funciones cae nuestro PIC se debe ver la tabla en el documento I2C que se encuentra en \MCC18\doc\periph-lib, en nuestro caso el PIC18F2550 es 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

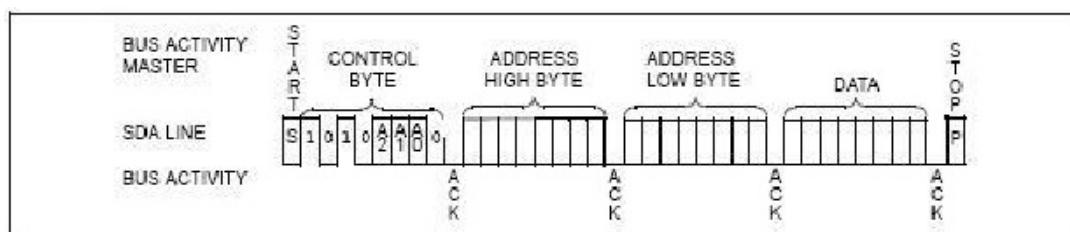
Para configurar la velocidad, además debemos cargar el registro **SSPADD**, que dependerá de la velocidad del oscilador. $\text{Clock}=\text{Fosc}/(4.(\text{SSPADD} + 1))$

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.*



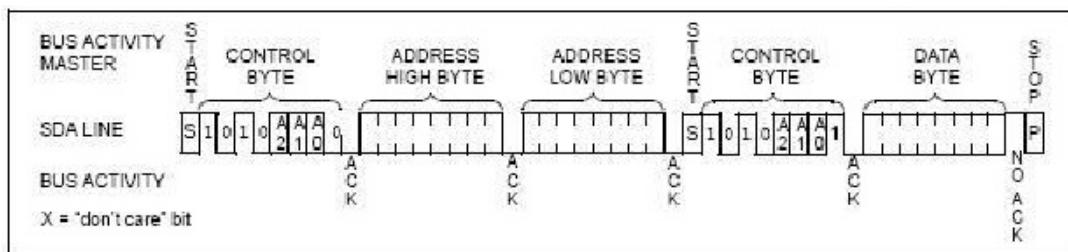
Operación de lectura

Las operaciones de lectura son iniciadas del mismo modo como las operaciones de

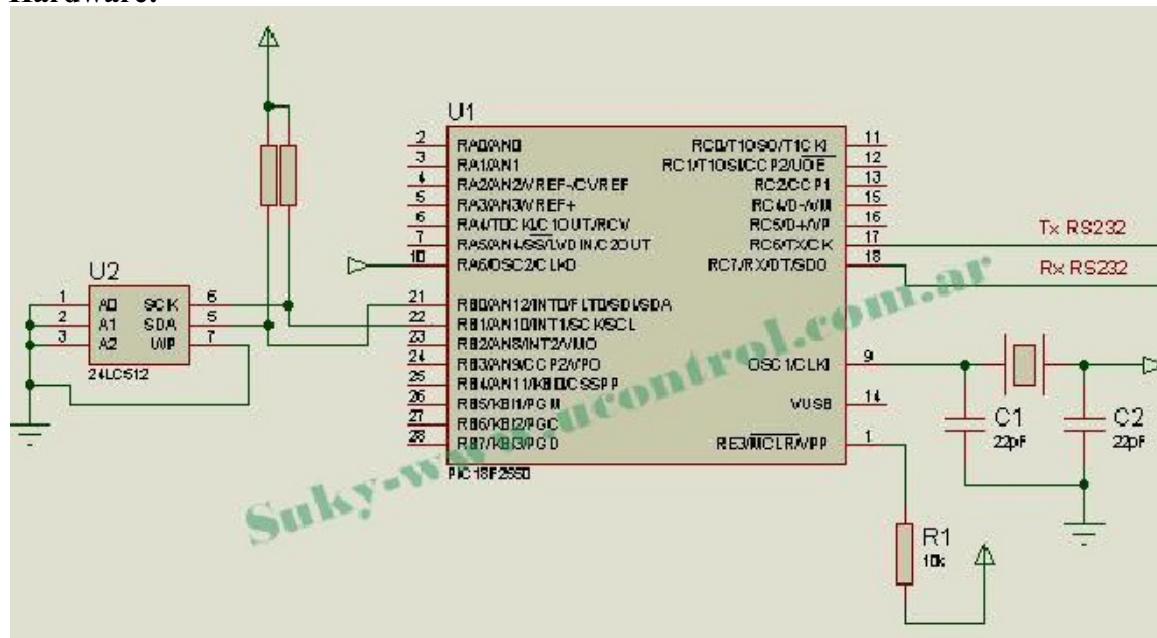
escritura, con la excepción de que la parte de R / W del byte de control es puesta a "1". Hay tres tipos básicos de las operaciones de lectura: la lectura de dirección en curso, lectura aleatoria y lectura secuencial.

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.



Hardware:



Código: (c)

```
#include <p18f2550.h>
#include <delays.h>
#include <usart.h>
#include <i2c.h>

#pragma config FOSC = XT_XT, FCMEN = OFF, IESO = OFF, CPUDIV = OSC1_PLL2
#pragma config PWRT = ON, BOR = OFF, BORV = 0
#pragma config 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 = ON, CP1 = ON, CP2 = ON
```

```

#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
#pragma config EBTRB = ON

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

void ISRRepcion(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 ISRRepcion _endasm
}
#pragma code // Cerramos seccion.-

// Rutina de Interrupcion.-

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

```

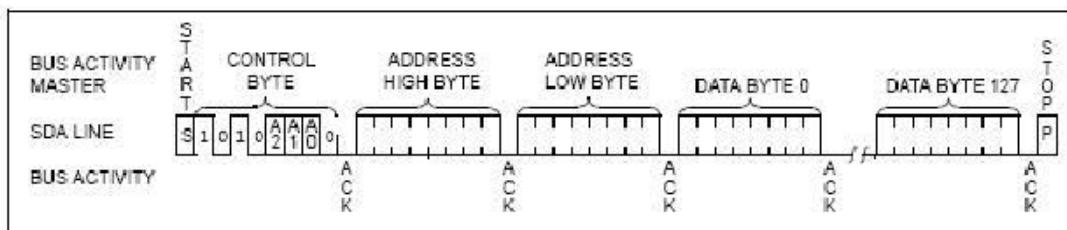
Ejemplo, escribiendo y leyendo la letra v en la dirección **0x0000**.



Segundo ejemplo de comunicación con memoria 24LC512 mediante I2C.

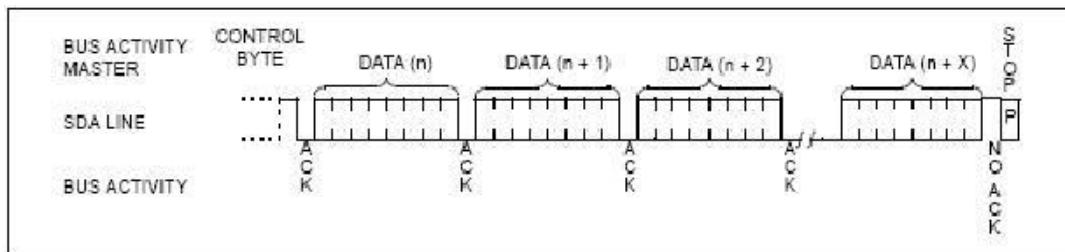
En este caso vamos a recibir desde la PC un string, que guardaremos en la memoria y luego leeremos para reenviar a la PC. Utilizaremos escritura secuencial y lectura secuencial:

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 secuenciales 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 secuenciales, la memoria contiene un puntero de dirección interno que es incrementado en uno en la terminación de cada operación.



Código: (c)

```
#include <p18f2550.h>
#include <delays.h>
#include <usart.h>
#include <i2c.h>

#pragma config FOSC = XT_XT,FCMEN = OFF,IESO = OFF,CPUDIV = OSC1_PLL2
#pragma config PWRT = ON,BOR = OFF,BORV = 0
#pragma config 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 = ON,CP1 = ON,CP2 = ON
#pragma config CPB = ON,CPD = ON
#pragma config WRT0 = ON,WRT1 = ON,WRT2 = ON
#pragma config WRTB = ON,WRTC = ON,WRTD = ON
#pragma config EBTR0 = ON,EBTR1 = ON,EBTR2 = ON
```

```

#pragma config EBTRB = ON

//*****volatile unsigned char Kbhit;
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_SYNCH_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; // Desabilitamos Prioridades
    INTCONbits.PIE1=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;
        BufferWriteI2C(0xA0,0x00,0x00,BufferE); //Se guarda en
memoria los datos recibidos.
        Delay1KTCYx(5);
        BufferReadI2C(0xA0,0x00,0x00,BufferL,19); // Leemos de

```

```

la memoria los datos guardados.-

    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 );
    WriteI2C( ByteControl | 0x01 ); // Cambia bit0 de ByteControl a
1 para realizar lectura.
    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:



Tutorial en construcción



Tutorial creado por [Suky](#).

Versión: 1.00

<http://www.ucontrol.com.ar/forosmf/programacion-en-c/tutorial-mplab-c18-desde-0/>

Portado a PDF por [Meta](#).

<http://electronica-pic.blogspot.com>

Cualquier sugerencia, dudas o cualquier comentario, puede realizarlas en el foro.

<http://www.ucontrol.com.ar/forosmf/programacion-en-c/tutorial-mplab-c18-desde-0-comentarios-dudas-yu-opiniones>

Puede haber futuras actualizaciones, mejoras y contenidos.

www.ucontrol.com.ar

