Mikroc Pro para PIC

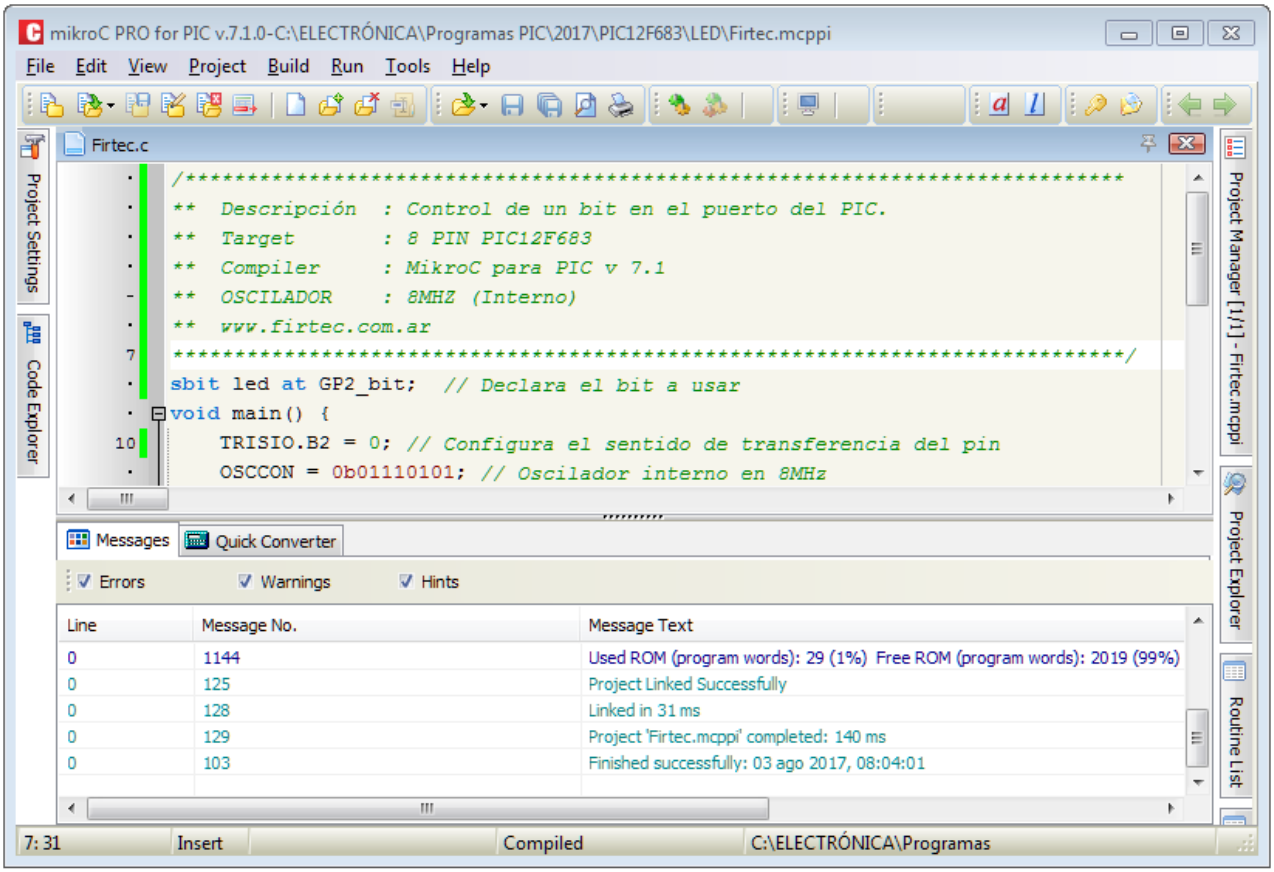
Índice General de Temas

Events that wake processor from sleep	
MCLR	Master Clear Pin Asserted (pulled low)
WDT	Watchdog Timer Timeout
INT	INT Pin Interrupt
TMR1	Timer 1 Interrupt (or also TMR3 on PIC18)
ADC	A/D Conversion Complete Interrupt
CMP	Comparator Output Change Interrupt
CCP	Input Capture Event
PORTB	PORTB Interrupt on Change
SSP	Synchronous Serial Port (I^2C Mode) Start / Stop Bit Detect Interrupt
PSP	Parallel Slave Port Read or Write

En la imagen anterior se aprecian las formas de sacar el microcontrolador del estado de Sleep.

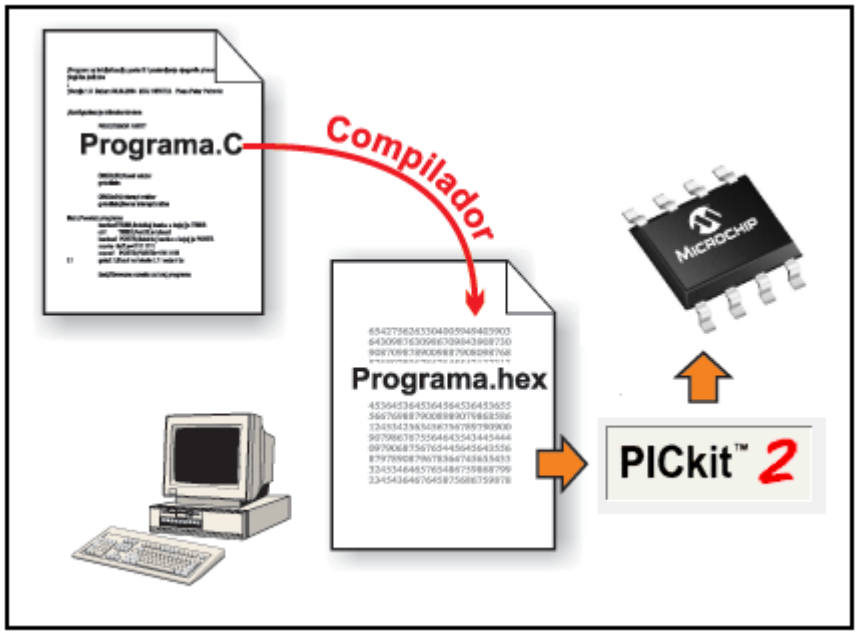
Entorno de trabajo MikroC.

Se descargar directamente desde sitio oficial de Mikroelektronika y en su versión libre permite generar código con limitación en su tamaño. Ensamblador, enlazador, gestión de proyectos y depurador, todo se realiza desde el mismo IDE.



La interfaz gráfica de usuario sirve como un único entorno para escribir, compilar y depurar código para aplicaciones con PIC.
También desde el IDE se inicia el proceso de programación, siempre que la compilación haya resultado exitosa.
Es simple, intuitivo y agradable de usar desplegando en una serie de pestañas todas las herramientas

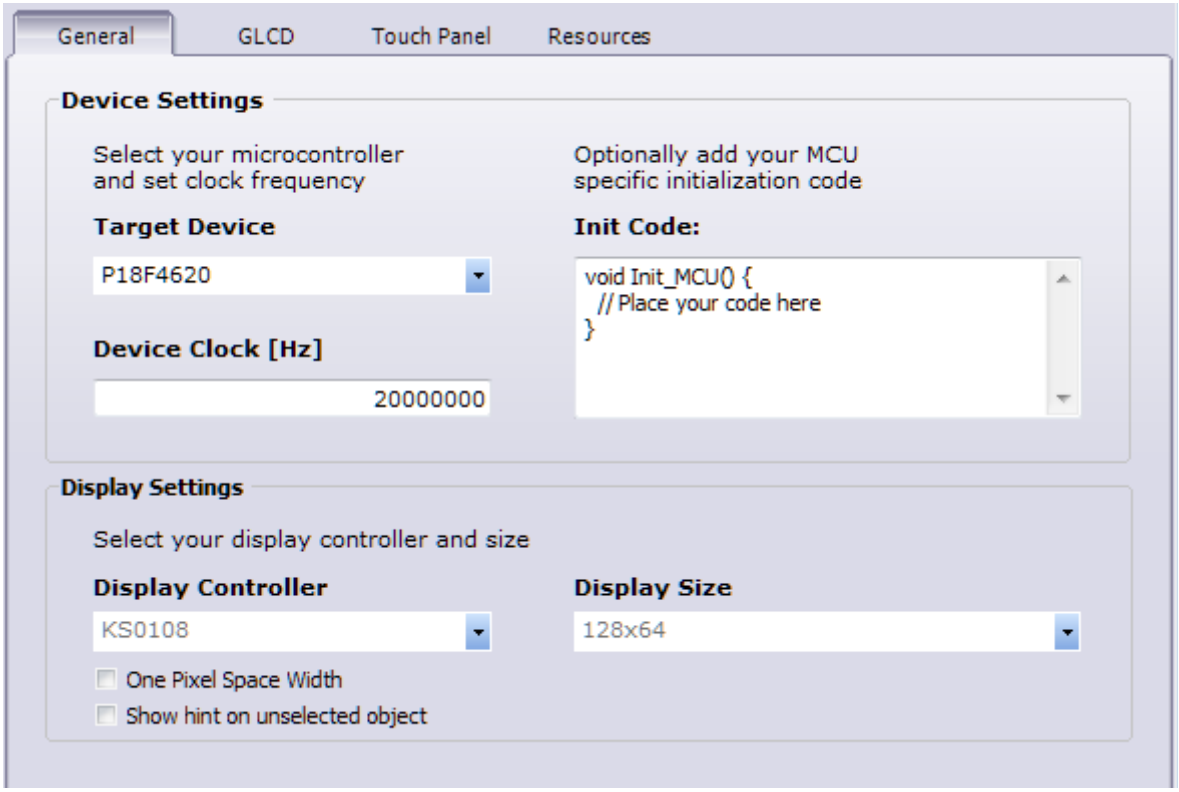
Cada vez que se compile el proyecto y el archivo hexadecimal sea actualizado (alterado) automáticamente Pickit2 programará el PIC.



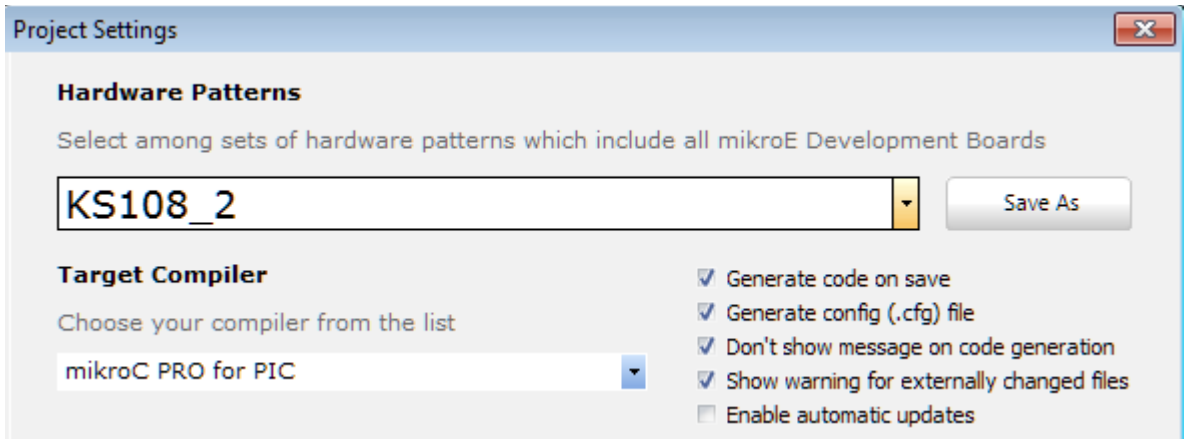
Estructura de un programa en C para PIC.

El lenguaje C está organizado en funciones, haciendo una mala comparación podemos decir que las funciones se parecen a las subrutinas del ensamblador, la comparación no es del todo justa puesto que las funciones son mas poderosas que las subrutinas pero vale para entender que una función es un trozo de código que hace algo específico.
De la mano de las funciones nace la programación modular, la idea será crear módulos (funciones) que resuelvan tareas puntuales, por ejemplo medir temperatura, pesaje, comunicaciones y cualquier actividad que el micro tenga que realizar. Al tener los módulos resueltos y funcionando podemos exportarlos a distintos programas (clásico copiar y pegar).
Podemos incluso crear un archivo que contenga todos los módulos de uso común y agregarlo al proyecto cuando sea necesario usar alguna de estas funciones.

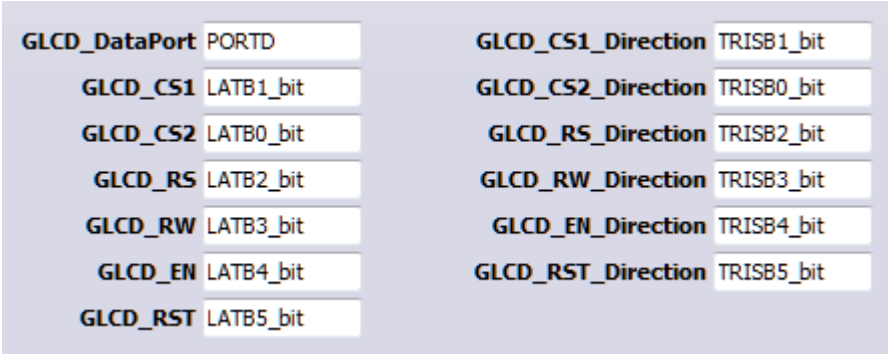
Todo programa en C es en si mismo una función, la función *main()*, punto de entrada al programa, código inicia en esa función y solo puede existir una función *main()* en todo el proyecto.
El nombre de la función es *main*, nombre obligado que se debe respetar, luego de la función viene



En la configuración de Visual GLCD indicamos el tipo de micro y su velocidad.
También el controlador que tiene la pantalla gráfica a usar.



Necesitamos indicar que pines serán usados para el control de la pantalla.

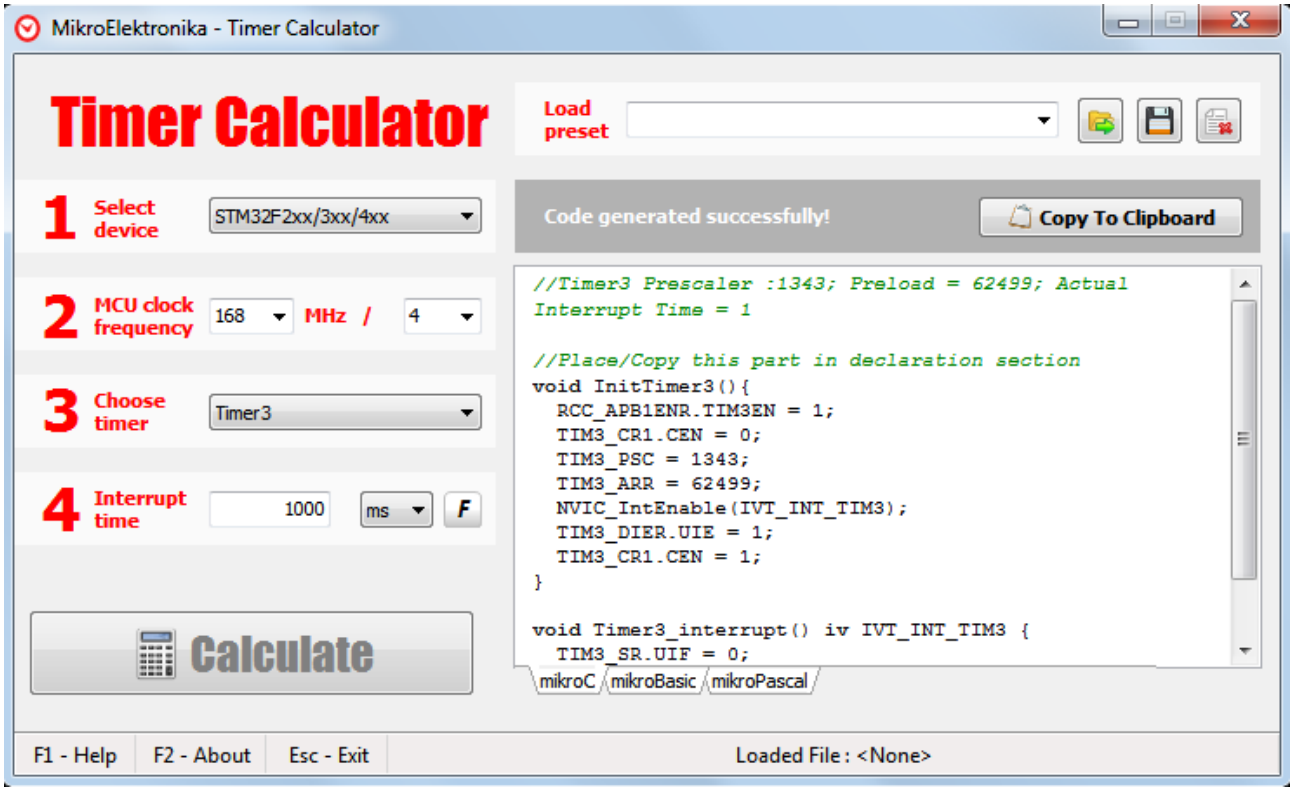


Para conocer los pines disponibles en su pantalla consulte la hoja de datos, en la imagen anterior se observa una típica configuración para el controlador KS180 pero será diferente para otros tipos de pantallas.
Como el KS108 no tiene *Touch*, no necesitamos nada mas para trabajar con esta pantalla.
Visual GLCD soporta varios tipos de pantallas, el KS108 es la mas simple y económica, las

```
  GIE_bit      = 1;     // Habilita las interrupciones globales

  while(1) {
          // Espera la interrupción en INT0
  }
}
```

Timer0 por Interrupción.

Para trabajar con los temporizadores *Mikroelektonika* tiene una herramienta que resuelve prácticamente todo el código de configuración, se descarga desde su sitio web.



Timer Calculator genera código para una gran variedad de microcontroladores y distintas arquitecturas. Solo debemos indicar el micro que estamos usando, su frecuencia de trabajo y cual temporizador vamos a configurar, la herramienta genera el código que solo debemos pegar en nuestro proyecto.

Un *timer* básicamente es un contador, en algunos microcontroladores estos contadores pueden ser ascendentes o descendentes, pero en muchos solo operan en forma ascendente. (*Consultar la hoja de datos para conocer el funcionamiento de los temporizadores de su microcontrolador*).

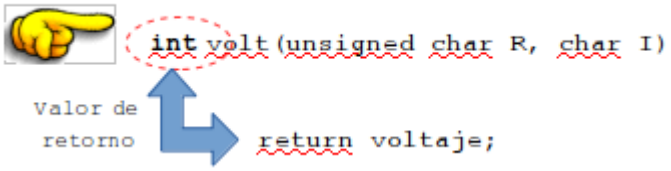
Suponiendo que cuenta con un temporizador que cuenta en forma ascendente, su funcionamiento es muy simple, si el temporizador es de 16 bits sus estados de cuenta podrán ser desde 0 a 65535, en el momento que el contador se desborda se genera una interrupción, conociendo la velocidad de cuenta (*el período del reloj*), puede fácilmente calcular cuanto tiempo tardará el temporizador en desbordar de acuerdo a los estados de cuenta.

En los temporizadores se puede definir el "*modulo*", por ejemplo cargando el modulo con el número 60000 el contador del temporizador contará 535 estados de cuenta y se dispara la interrupción.

```
/*************************************************************************
**  Descripción  : Contador de tres dígitos, el multiplexador para los
**                 dígitos funciona controlado por una interrupción que
**                 es generada desde el Timer 0. Un LED en el pin 33
**                 cambia de estado en cada interrupción.
**                 Sirve solo como indicador para saber que la interrupción
**                 está funcionando.
```

por referencia. Los utilizados en la función anterior son del tipo por valor, pues no alteran directamente el contenido de la variable, sino que se realiza una copia de la variable sin alterar el contenido.
El cuerpo realiza la labor específica de la función, en este caso, calcula el voltaje. Inicialmente se declara una variable temporal voltaje, las variables declaradas internamente en la función, solo existen dentro de ella, cuando la función cumple su cometido (regresando a la función principal) la variable voltaje desaparecerá del sistema.

Este concepto sobre funciones y variables es muy importante porque una variable declarada como global sera accesible desde cualquier parte del programa sin embargo variables locales a las funciones solo con accedidas dentro de la función, las variables locales existen solo cuando son invocadas en la función, las variables globales están siempre en memoria disponibles en cualquier momento y por cualquier parte del programa.
En el encabezado de la función se especifica la variable que va a retornar, volviendo a la función volt(), esta retorna un tipo entero.



Si observamos más a fondo el cuerpo de la función, la primera línea de la izquierda nos describe que variable va a ser retornada en el caso que tuviera retorno, una función que no retorna valor y no recibe argumentos sería:

```
void Nombre_Función(void);
```

Esta variable (voltaje) también fue declarada tipo entero, sino el compilador dará advertencias parciales sobre la variable a retornar. No necesariamente tuviésemos que haber escrito la función con una variable temporal de cálculo, la misma acción pudo haberse escrito de la siguiente manera:

```
int volt(unsigned char R, char I) {
 return (R*I);
}
```

La función tiene el mismo fin o propósito que la original y dará el mismo resultado.
La función debe ser llamada especificando el mismo tipo de variable como argumentos a utilizar.
Observe esta parte del código:

```
unsigned char a;
char b;
int resultado;
a = 2;
b = 5;
resultado = volt(a,b);  /* Utilizando variables */
resultado = volt(9,-1); /* Utilizando constantes */
```

Preferiblemente se prefiere escribir la declaración de la función en los archivos *.h y la definición en los archivos *.c, de tal manera que solo llamáramos la librería correspondiente a todas las funciones y utilizáramos solo las que necesitamos.

Estructuras en C.

Una estructura es una colección de variables que comparten el mismo nombre. Las estructuras son un tipo de acción poderosa de C que nos permite ordenar en grupos las variables. Una estructura se

- *8 x TAD.*
- *6 x TAD.*
- *4 x TAD.*
- *2 x TAD.*
- *0 x TAD.*

Para las conversiones A/D correctas, el reloj de conversión A/D (TAD) debe ser tan corto como sea posible pero mayor que un mínimo TAD (para más información véase datasheet del PIC utilizado) por ejemplo para la serie 16F en 1.6us y para la serie 18F en 0.7us.
Salir fuera de los rangos operativos puede terminar en un daño permanente del conversor.

Veamos un ejemplo:
Trabajando a 20Mhz. 1/20 = 50nS
32 x 50nS = **1600nS** (1,6uS) El limite a respetar es ***0,7uS.***
No podemos tener un tiempo menor a 700nS.
El tiempo mínimo de adquisición no puede ser menor a ***2,45uS.***
2 x 1,6uS = **3,2uS**
También podríamos haber usados los siguientes valores:
16 x 50nS = **800 nS** 4 x 0,8uS = **3,2 uS**

Recordar:

- ***Tosc*** *Periodo del circuito de oscilación (en nuestro caso el cristal).*
- ***Tacqt*** *Significa Periodo de Adquisición, es el tiempo necesario para que el circuito sample/hold guarde la muestra antes de ser convertida, es decir el tiempo que el condensador de muestreo se cargue con el voltaje a convertir. Esto se hace para evitar errores en la conversión.*
- ***Tad*** *Significa Periodo de conversión análogo digital, es el tiempo en que el ADC realiza la conversión de cada bit.*

Algunos ejemplos de selección del Tad.

Si uso 96MHz/2=48MHz
*Tad=64*Tosc=64/Fosc=64/48MHz=1.3333us*
*Utilizo 96MHz/4=24MHz porque Tad=32*Tosc=32/Fosc=32/24MHz=1.3333us*
*Utilizo 96MHz/6=16MHz porque Tad=16*Tosc=16/Fosc=16/16MHz=1us*
Pero el tiempo de cada ciclo aumenta lo que disminuye el tiempo total de conversión
*Si uso un cristal de 20MHz, Tad=16*Tosc=16/Fosc=16/20MHz=800ns*
*Si uso el oscilador interno a 8MHz, Tad=8*Tosc=8/Fosc=8/8MHz=1us.*

Para utilizar el módulo de conversión analógica-digital Mikroc proporciona una librería que contiene todas las funciones necesarias para la configuración e implementación del mismo.

Este módulo posee varias opciones de configuración que dependerán de la aplicación y para más información leer la hoja de datos del microcontrolador utilizado.
El siguiente es un ejemplo de uso del conversor A/D.

```
/************************************************************************
**  Descripción  : Voltímetro Hexadecimal [000 a 3FF]
**  Target       : PIC18F4620 de 40 pines
**  Compiler     : MikroC para PIC v7.1
**  XTAL         : 20MHZ
**  Autor        : Firtec - www.firtec.com.ar
** ************************************************************************/

void Leer_Conversor(void);  // Se declara la función que leerá el conversor A/D
```

El uso de la memoria es muy sencillo, suponiendo que voy a guardar el dato 236 en la posición de memoria 120 el código seria como sigue:

```
EEPROM_Write(120, 236);  // Escribe la memoria
delay_ms(20);            // Espera a que terminela escritura
```

Para recuperar el dato se lee la posición de memoria.

```
dato = EEPROM_Read(120);  // Lee la memoria
```

La memoria *EEPROM* interna también se puede alterar (escribir/borrar) desde el propio programador.

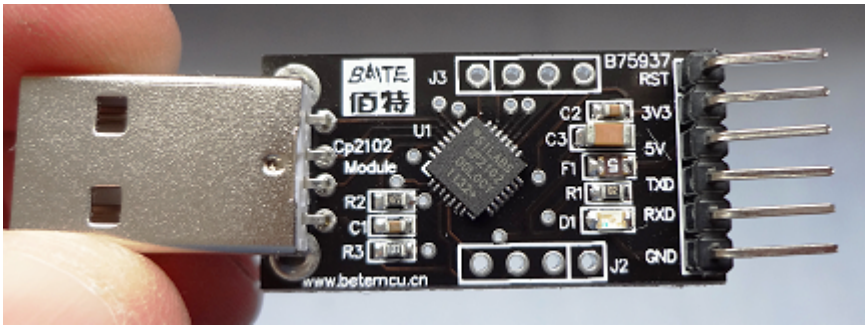
Trabajo practico

Realice un programa para el control de acceso de una puerta que cumpla con las siguientes consignas:

- La apertura de la puerta se hará efectiva cuando se ingrese una clave numérica de cuatro cifras que será comparada con un número almacenado en EEPROM.
- Los números de la clave serán generados con un potenciómetro colocado en un canal A/D que irán desfilando en una pantalla de uno en uno mientras el usuario mueve el potenciómetro, los números mostrados solo pueden ir de 1 a 9.
- Cada vez que un número que forma la clave de cuatro cifras se hace visible en la pantalla se lo valida con el botón de usuario que oficiará como botón de "Enter".
- Para indicar que la acción ha sido válida utilizar un LED.

Funcionamiento de la UART

En la actualidad y ante la ausencia de puertos RS232 en las computadoras, acostumbramos a trabajar con puentes *USB-RS232* que solucionan muchos o todos los problemas a la hora de vincularnos con computadoras a través del viejo protocolo 232.
Es interesante comentar que debido a la desaparición de estos puertos en arquitecturas de computadoras uno tiende a pensar que el puerto RS-232 está extinguido, nada mas alejado de la realidad ya que arquitecturas en 16 y 32 bits no solo incorporan uno de estos puestos sino varios.
El controlador no resuelve la capa física, es decir que para implementar las comunicaciones deberemos utilizar ya sea el clásico MAX232 o los modernos puentes USB-232.



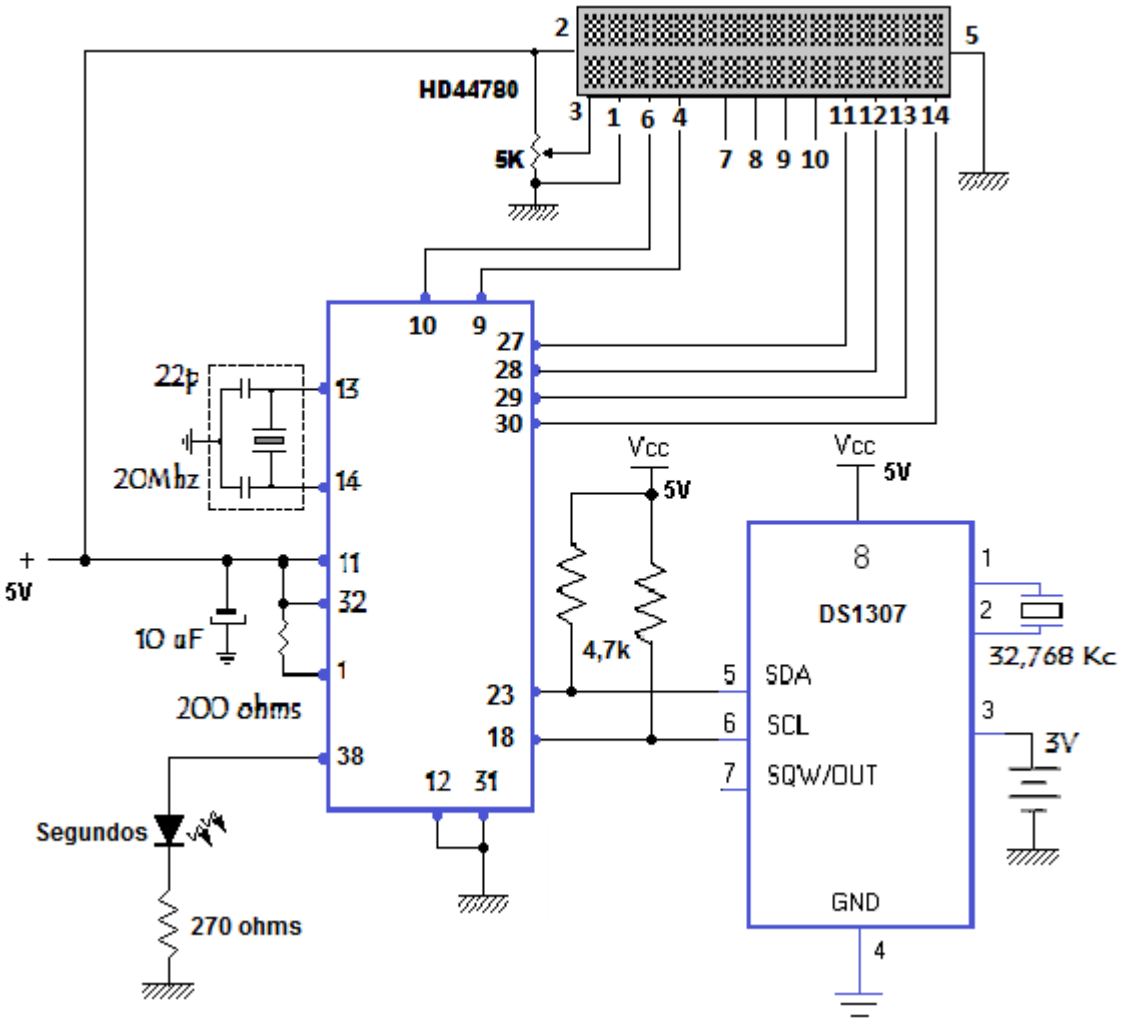
Esto es particularmente interesante porque con esta tecnología podemos reutilizar los viejos

```
        Lcd_out(1, 6, time);
        Lcd_out(2, 7, date);
        Delay_ms(100);
  }
}
```

Se recomienda la lectura de la hoja de datos del DS1307 para mas detalles de funcionamiento.

Trabajo practico

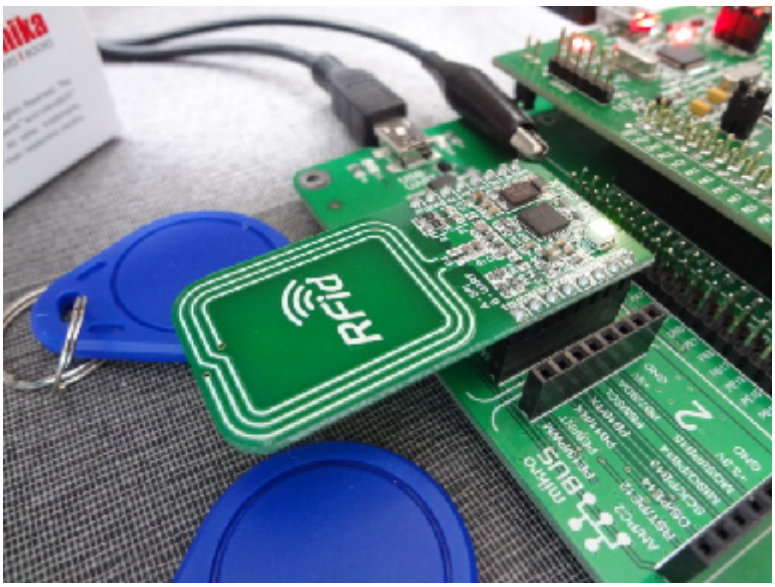
1- Realice un programa para programar el reloj calendario mediante botones.
2- Agregue el código necesario para que LED colocado en el pin 38 se encienda cada segundo, un segundo enciende siguiente segundo apaga y así sucesivamente.



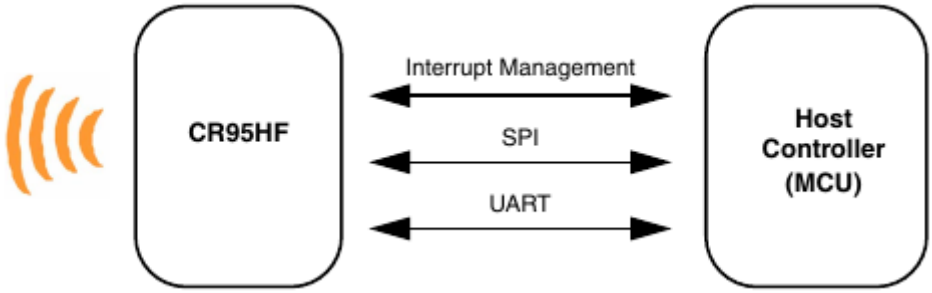
Circuito de la Aplicación

Que es RFID.

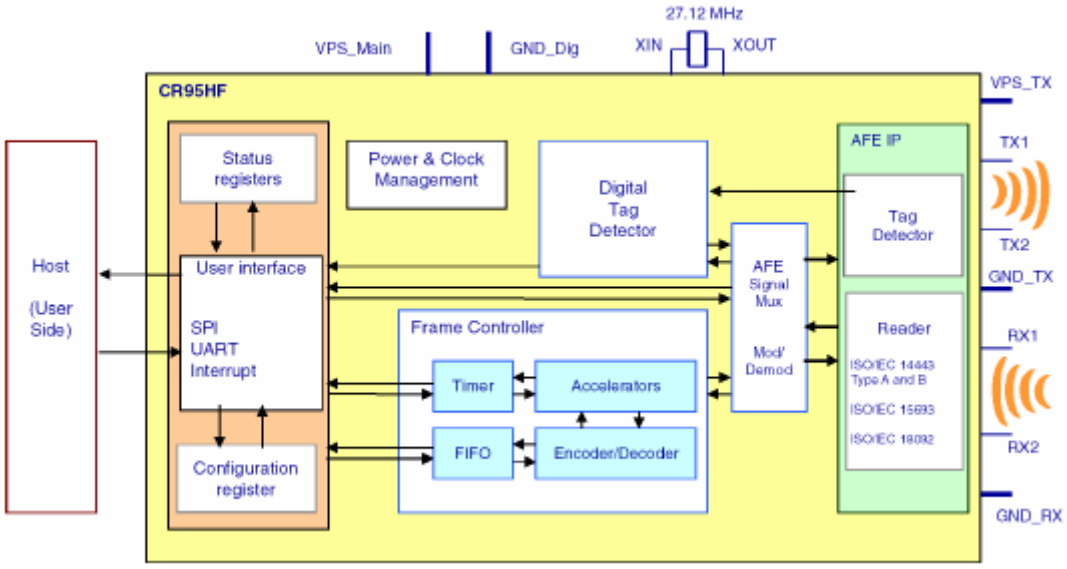
RFID o identificación por radiofrecuencia es un sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o Tags RFID. El propósito fundamental de la tecnología RFID es transmitir la identidad de un objeto o persona mediante ondas de radio.
Las etiquetas RFID o Tags, son unos dispositivos pequeños, similares a una calco autoadesiva, que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contiene la antena



Existen muchos chips disponibles en el mercado para resolver las comunicaciones RFID, el ***CR95HF*** es solo uno de estos chips que que hemos elegido por su simpleza de conexión y gran eficiencia en su funcionamiento, esto lo convierte en una opción a considerar cuando estamos pensando en construir aplicaciones que implementan el reconocimiento de *tags* RFID.
Este circuito integrado maneja varios protocolos de comunicaciones RFID y se puede vincular al host (microcontrolador de control) mediante SPI o UART.



Los protocolos RFID soportados son ISO/IEC 14443-3 Tipo A y B, ISO/IEC 15693, ISO/IEC 18092, la comunicación se realiza 13.56 MHZ.
En la imagen siguiente podemos ver los módulos internos de este chip, su funcionamiento es bastante complejo a nivel de electrónica sin embargo esta complejidad es transparente al usuario gracias a las capas de software necesarias para su funcionamiento.



```
  Lcd_Cmd(_LCD_CURSOR_OFF);
  Lcd_Out(1,5,"ROM-CODE");
  do {
    Ow_Reset(&PORTA, 4);         // Reset del bus 1 Wire
    Ow_Write(&PORTA, 4, 0x33);   // Envía comando Read_ROM
    Delay_us(120);

       tmp = 17;
       for(i = 0; i <= 7; i++)
          {
            tmp = tmp - 2;
            sernum = Ow_Read(&PORTA, 4);
            bytetohex(sernum, sernum_hex);
            Lcd_Out(2, tmp, sernum_hex);
          }
    Delay_ms(500);
  } while (1);
}
```

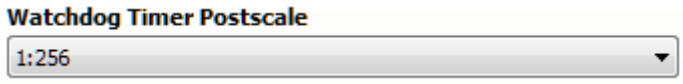
Se puede ver en la imagen el resultado obtenido al correr el código para la extracción del ID de los sensores DS18x20.

Trabajo practico

En base a lo aprendido escriba un programa que envíe por el puerto *USART* la temperatura de un sensor *DS18B20* y visualizar los datos en una la computadora.

```
void main() {
  ADCON1 = 0X0F;
  TRISA=0X01;
  Lcd_Init();
  Lcd_Cmd(_LCD_CLEAR);
  Lcd_Cmd(_LCD_CURSOR_OFF);
  Lcd_Cmd(_LCD_CLEAR);
  WDTCON=0x0F;          // Configura Prescaler
  OPTION_REG=0xEF;      // Configura Postscaler

  WDTCON.SWDTEN = 1;    // Watchdog activo

  if(WatchDog)
    Lcd_Out(1, 1, "WDT Normal");
  else
      Lcd_Out(1, 1,"WDT Activado!!");

   while(1){
  if(PORTA.RA0){
    asm{ CLRWDT }
    //asm SLEEP
   }
  }
 }
```

Los valores de configuración para los tiempos de espera del watchdog se pueden ajustar directamente desde la propia configuración del proyecto.



Un ejemplo de uso práctico para el watchdog.

Imaginemos que tenemos un sistema de control que debe verificar la posición de una válvula de vapor en una caldera, como sabemos que es posible que este sensor se dañe y el microcontrolador se quede esperando (colgado) indefinidamente la señal de este sensor, al llegar a este punto borramos el temporizador del watchdog y esperamos la señal del sensor pero en la memoria EEPROM del PIC colocamos una “marca”, un número que identifica donde se encuentra el programa en ese momento.
Si la señal del sensor no llega, como no se ejecuta el comando de borrado, el watchdog se dispara y el microcontrolador arranca nuevamente pero al hacerlo el programador agregó al código de arranque una función para verificar el estado del registro RCON y si se detecta un reset por watchdog solo debemos “mirar” que número tenemos en nuestra memoria EEPROM para saber donde ha ocurrido el fallo.
El paso siguiente será informar al usuario, el clásico cartel de “*Error xx llame al técnico*”, o informar del problema específico, “*Error en sensor de válvula x*”.
Como vemos el watchdog no evita que el microcontrolador se “cuelgue” pero al menos lo llevamos a una zona en donde puede comunicar el evento, incluso cuando el cliente nos consulte y nos diga el número de error ya sabemos que tipo de reparación tenemos que realizar.

Ningún proyecto comercial debería salir al mercado sin la correcta configuración de este sistema de seguridad que muchas veces incluso ni siquiera está activado.

Sensor de temperatura y Humedad DHT22.

Con este sensor podemos medir temperatura y humedad, es un sensor simple útil en proyecto semi profesionales o domésticos.

Observe que los tres primeros bits del registro **CMCON** (*CM2-CM1-CM0*) configura el tipo de montaje que se aplicará a los comparadores.
En la practica el uso de los comparadores puede ser útil cuando se debe comparar dos voltajes por ejemplo una temperatura respecto de un set-point, si bien esto lo podemos hacer con los conversores analógicos, el uso de los comparadores evita el código necesario para procesar el dato desde el conversor.

Control PWM con PIC12F683.

El módulo PWM del microcontrlador genera una onda cuadrada con una frecuencia determinada en general bastante alta, por ejemplo 10 KHz. Luego nosotros podemos ir cambiando el ciclo de trabajo (*% del periodo en alto*) de la señal sin alterar la frecuencia.
El parámetro fundamental de una modulación PWM es la frecuencia (o su inverso el periodo) de modulación.
En los PIC dicha frecuencia es programable sin embargo tiene claras limitaciones en base a varias variables que son las siguientes.

- *La frecuencia del oscilador principal Fosc.*
- *El pre-scaler (PRE) o divisor previo del timer TMR2 que puede tomar los valores 1:1, 1:4 o 1:16.*
- *El registro PR2 (0-255) asociado al timer TMR2.*

La frecuencia PWM responde a la fórmula:

$$F\_pwm = F\_osc / [4 \times PRE \times (PR2+1)]$$

o lo que es lo mismo, el periodo del PWM será el inverso de dicha frecuencia:

$$T\_pwm = [(PR2+1) \times 4 \times PRE] \times Tosc$$

El valor máximo del divisor previo *PRE* es 16 y el de (*PR2+1*) es 256.
Por lo tanto la frecuencia PWM más baja posible será Fosc/16384. Para un oscilador de 20 MHz tenemos una Fpwm mínima de 1.22 KHz (20000/16384).
El módulo PWM usa el timer TMR2, por lo que éste no podrá usarse como temporizador de propósito general mientras se esté usando PWM.
Si que es posible usarlo (y ahorrarnos perder otro timer) si queremos disparar una interrupción cada cierto tiempo. El postscaler del TMR2 no tiene efecto sobre la frecuencia PWM, pero si influye sobre cuando se dispara (si está habilitada) la correspondiente interrupción (TMR2_flag).
Si por ejemplo el post-scaler es 1:16 entonces la interrupción del TMR2 se activa cada 16 periodos del PWM.
Lo primero que tenemos que hacer para usar el módulo PWM es habilitarlo indicando que va a usarse como generador de onda PWM, ya que dicho módulo es compartido con otras funciones (*Capture/Compare*). La forma de hacerlo es poner a uno los 4 bits menos significativos del registro CCP1CON.
Los PIC18 suelen tener 2 módulos PWM por lo que existe un segundo registro CCP2CON.
Podemos habilitar uno o los dos módulos independientemente. Sin embargo, como ambos usan el registro PR2 y el timer TMR0 como base de tiempos, la frecuencia programada será la misma en ambos módulos.
Lo que si es posible variar por separado es el ciclo de trabajo, el duty de la frecuencia de cada módulo. El ciclo de trabajo se codifica con un número de 10 bits (0-1023) almacenado de la siguiente forma.

CCPR1L : 8 bits más significativos del ciclo de trabajo.

CCP1CON.DC1B0 y DC1B (bits 5 y 6 de CCP1CON): Guardan los 2 bits menos significativos.

```
            // Tiempo aleatorio para la velocidad
            flicker_random_speed_start = random(flicker_speed_min,
                   flicker_speed_max);
            flicker_random_speed_end = random(flicker_speed_min,
                   flicker_speed_max);

            // low -> high
            for (i = flicker_random_low_start; i<flicker_random_high; i++) {
                PWM1_Set_Duty(i);
                Delay_cyc(flicker_random_speed_start);
            }

           VDelay_ms(random(flicker_hold_min, flicker_hold_max));

            // high -> low
      for (i = flicker_random_high; i>=flicker_random_low_end; i--) {
           PWM1_Set_Duty(i);
           Delay_cyc(flicker_random_speed_end);
      }
       VDelay_ms(random(flicker_pause_min, flicker_pause_max));
  }
}
```

Manejo de archivos en formato FAT.

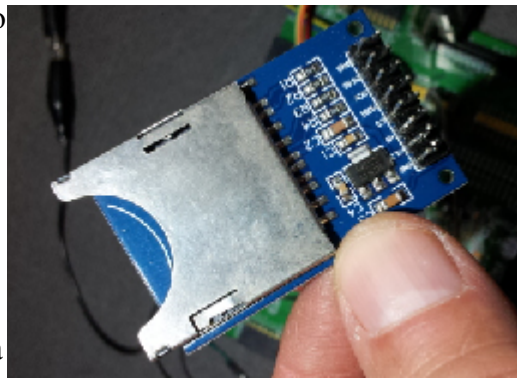
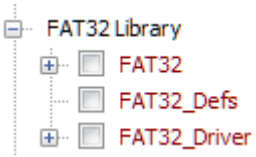
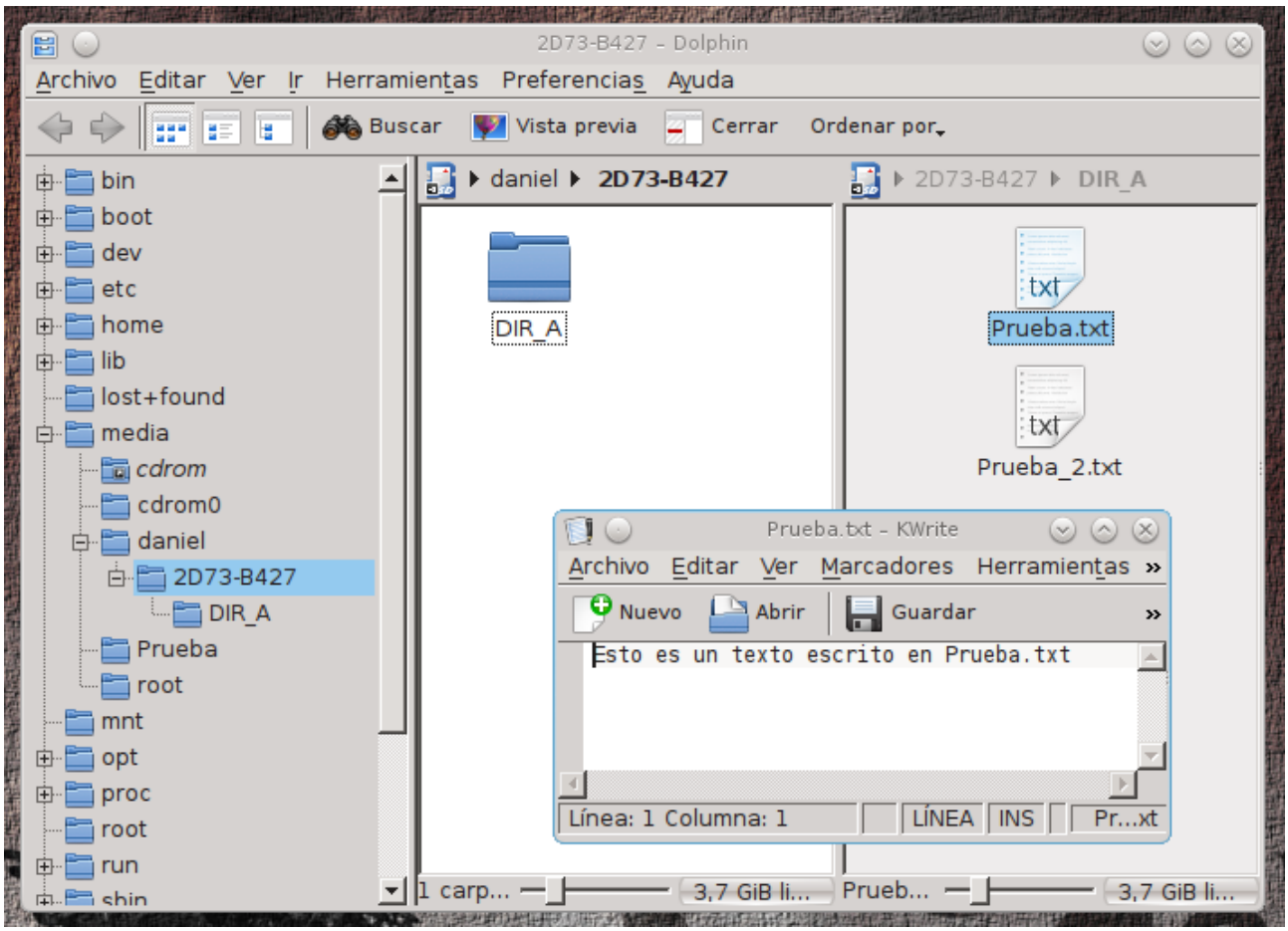


Tabla de asignación de archivos, comúnmente conocido como FAT (del inglés file allocation table), es un sistema de archivos desarrollado para MS-DOS, así como el sistema de archivos principal de Microsoft Windows hasta Windows Me.
FAT es relativamente sencillo por lo que es un formato popular para discos admitido prácticamente por todos los sistemas operativos existentes. Se utiliza también como mecanismo de intercambio de datos entre sistemas operativos distintos que coexisten en la misma computadora, lo que se conoce como entorno multi-arranque.
También se utiliza en tarjetas de memoria y dispositivos similares.
Las implementaciones más extendidas de FAT tienen algunas desventajas por ejemplo cuando se borran y se escriben nuevos archivos tiende a dejar fragmentos dispersos de éstos por todo el soporte y con el tiempo esto hace que el proceso de lectura o escritura sea cada vez más lento.
La denominada desfragmentación es la solución a esto, pero es un proceso largo que debe repetirse regularmente para mantener el sistema de archivos en perfectas condiciones.
FAT tampoco fue diseñado para ser redundante ante fallos, inicialmente solamente soportaba nombres cortos de archivo: ocho caracteres para el nombre más tres para la extensión.
También carece de permisos de seguridad lo que significa que cualquier usuario puede acceder a cualquier archivo.
Afortunadamente Mikroc tiene todo el soporte para el manejo de FAT.



Esta librería de la biblioteca nos ofrece todas las funciones necesarias para escribir, borrar incluso formatear una memoria SD en FAT32.
En nuestro ejemplo como hardware y para facilitar el armado, usaremos una placa ya construida que



En la imagen anterior se aprecia el resultado de la ejecución del ejemplo propuesto para FAT.

```
                * se podrían perder
                */
               }
        }
```

Sensor de temperatura & Ethernet.

Otro ejemplo podría ser enviar la temperatura leída desde un sensor DS18S20 y enviar los datos por la red Ethernet.
Tenemos una página embebida en el microcontrolador con el siguiente formato.

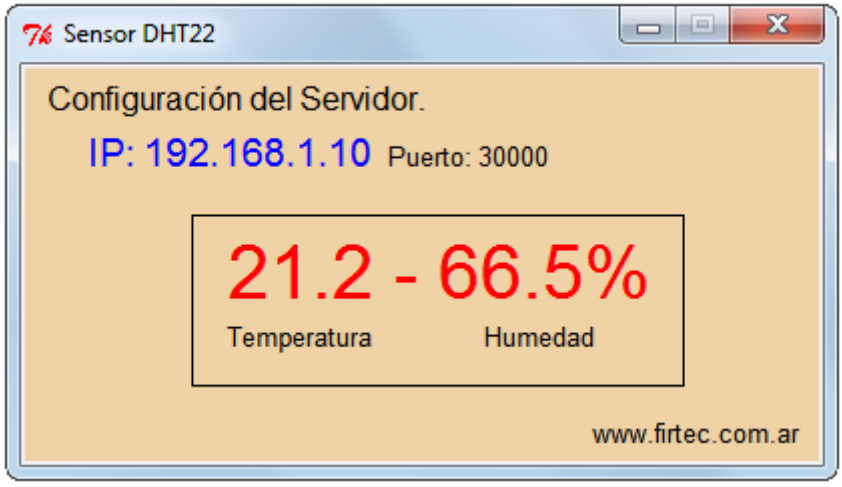
El programa actualiza de manera automática la temperatura mostrada en la página web.
El ejemplo a sido testeado con el navegador *Firefox*.

```
#include "Lib_1Wire_DS18s20.h"

void Mesure()
{
 Reading_Temperature_DS18s20();

}

char uart_rd;
char  buffer[32];


sbit SPI_Ethernet_Rst at RC0_bit;
sbit SPI_Ethernet_CS at RC1_bit;

sbit SPI_Ethernet_Rst_Direction at TRISC0_bit;
sbit SPI_Ethernet_CS_Direction at TRISC1_bit;
```

En el caso que nuestro módulo este en modo 3, tendremos como respuesta dos direcciones IP, una será 192.168.4.1 que es la que corresponde al modo 2 asignada por el propio DHCP interno, (esta dirección no se puede cambiar) y la otra será la que nos asigna el servidor DHCP de nuestra red.

Midiendo Temperatura y Humedad por Wi-Fi.

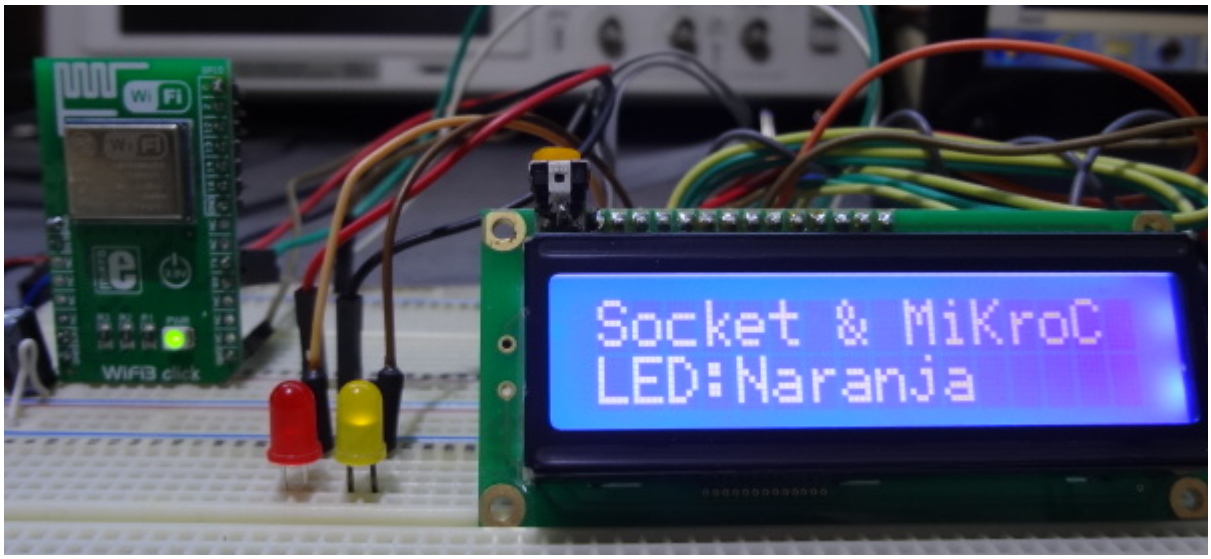
Vamos a usar nuevamente el sensor DHT22, pero en esta ocasión pretendemos enviar los datos de su medición a través de la red y recibirlos en un socket servidor.
El resultado será como el que se aprecia en la siguiente imagen.



El código Python nos informará cua es la dirección IP que tiene la computadora dentro de la red, en el socket cliente (*lo que va en el microcontrolador*) debemos conectarnos a esta dirección que es la que espera recibir lo datos. Cuando la conexión se establece nos informa la dirección IP del cliente y el puerto. (*192.168.1.10 puerto 30000 en este caso*).
El código completo del servidor Python es el siguiente. (***Python v2.7***)

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import socket
import sys
import errno
import time

from Tkinter import *
from tkMessageBox import showinfo

bandera = 0

class MyGui(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)

#********** Función para conocer el IP del servidor *************************
def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(('10.255.255.255', 0))
        IP = s.getsockname()[0]
    except:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP
#*******************************************************************************
```

También se mostrará en una pantalla LCD cual LED se encuentra activado.



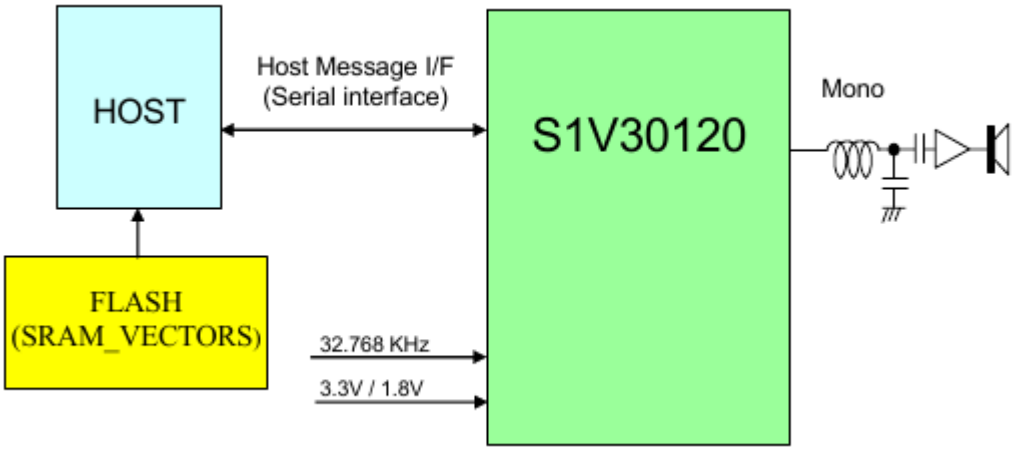
Para interactuar con la electrónica vamos a usar un pequeño servidor escrito en Python con el siguiente aspecto para el usuario.

Se puede observar en la ventana que como en el ejemplo anterior se publica la dirección a donde debemos conectarnos desde nuestro PIC, en este caso la IP 192.168.1.13 y el puerto 30000.
(Asegúrese de otorgar los correspondientes permisos en el cortafuegos del sistema operativo).
En este caso también se puede ver la dirección origen como así también el puerto.
Cada vez que actuemos sobre uno de los botones el LED correspondiente cambia de estado.
En nuestro canal de yotube (*firadmin*) puede ver un vídeo de este proyecto funcionando.

El código Python para el interprete 2.7 es el siguiente.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import socket
import sys
import errno
import time

from Tkinter import *
from tkMessageBox import showinfo

bandera = 0
```

```
class MyGui(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)

#********** Función para conocer el IP del servidor *************************
def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(('10.255.255.255', 0))
        IP = s.getsockname()[0]
    except:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP
#*******************************************************************************
UDP_PORT = 30000    # Puerto del socket en el Servidor
Dir_IP = get_ip()   # Obtiene la dirección del Servidor
time.sleep(0.02)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Crea Socket UDP
sock.bind(("", UDP_PORT)) # Recibir de cualquier cliente
sock.setblocking(0)   # Socket NO Bloqueante

ventana = Tk()    # Crea la ventana
ventana.title('UDP_LED')  # Nombre de la ventana
ventana.config(bg="beige")      # Color de fondo
ventana.geometry("400x200")     # Tamaño de la ventana
ventana.resizable(0,0) # Evita que se pueda cambiar de tamaño la ventana

x = 0    # Variable del sistema
y = 0    # Variable del sistema

#************************ FUNCIÓN RECURSIVA *********************************
def update_label():
    try:
        data,addr = sock.recvfrom(40)
    except socket.error, e:
        err = e.args[0]
        if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
            time.sleep(0.01)
            #print 'No hay datos!!'
            ventana.after(1, update_label)
    else:
        global bandera
        if bandera == 1:
            sock.sendto('4', addr) # Envía comando para el Led Rojo
            bandera = 0
        if bandera == 2:
            sock.sendto('5', addr) # Envía comando para el Led Naranja
            bandera = 0
        label_cliente.config(text = addr) # Muestra info del cliente
        ventana.after(1, update_label)

#******************** Función del Boton LED *********************************
def azul( ):
    global bandera  # Bandera el es comando para encender los led´s
    bandera = 1
def naranja( ):
    global bandera  # Bandera el es comando para encender los led´s
    bandera = 2

label_firtec = Label(ventana, text="www.firtec.com.ar", bg="beige", fg="black",
font=("bold", 13))
```



Este circuito integrado contiene un motor de síntesis de voz de Fonix DECtalk® v5 que puede hacer que un dispositivo hable en Ingles o en Español clásico o latino. Con un control de nueve voces predefinidas se puede ajustar su funcionamiento para que el resultado final sea muy próximo al sonido de la voz humana..
El motor DECtalk incluye un analizador que ofrece a los usuarios un control preciso de la calidad, el tono y la entonación de la voz sintetizada.
El audio se reproduce a una velocidad de muestreo de 11.025kHz lo que genera una fonética muy clara y limpia.
En el ejemplo estamos usando una placa fabricada por Mikroelektronika llamada TextToSpeech click, la empresa fabricante publica el diagrama electrónica de esta placa que por su costo no justifica el trabajo y tiempo de construirla uno mismo.

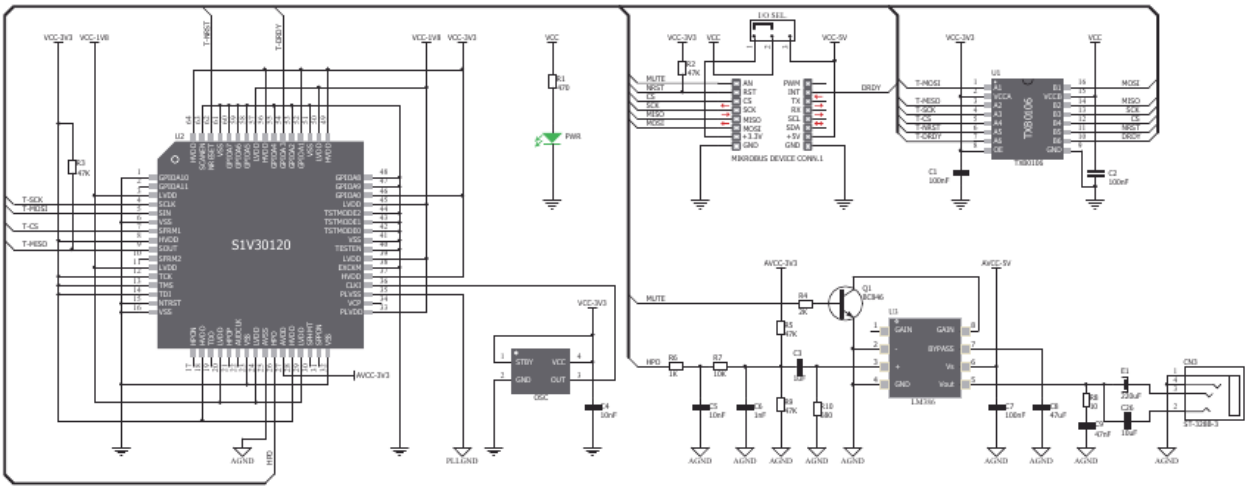


Diagrama de la placa TextToSpeech click.

Para poder trabajar con el chip SiV30120 debemos agregar al Mikroc un paquete que contiene todas las funciones y algoritmos necesarios para el manejo del sintetizador de voz..
El paquete se descarga desde el mismo sitio de MikroeleKtronika y se instala con el administrador de paquetes, que también descargamos desde el mismo sitio.
Primero debemos descargar el paquete para el manejo de sintetizador de voz, seleccionamos el que corresponde a Mikroc Pro para PIC.