Índice General de Temas

Acerca de este libro.

El eje central del libro es la programación para PIC de Microchip en ocho bits y en treinta dos bits para la arquitectura de ARM con el compilador MikroC de MikroElektronika

Compilador largamente probado y sin duda una de las herramientas que junto con los compiladores oficiales de las correspondientes marcas, generan código confiable y muy eficiente en el uso de los recursos de los microcontroladores.

MikroC ofrece gran cantidad de código resuelto, funciones y drivers contenidos en una extensa biblioteca que hacen el trabajo del programador mucho más sencillo acortando los tiempos de desarrollo y depuración de código.
Para la mayoría de los programadores de microcontroladores incorporar a su esquema de trabajo la arquitectura de ARM puede ser un paso complejo debido a las grandes diferencias que existen con otras arquitecturas como PIC, Atmel, etc.

Pensando en esto se ha desarrollado el presente trabajo que pretende hacer más fácil aprender a programar tanto ARM con su núcleo Cortex como también PIC con el compilador MikroC.

También usaremos Visual TFT, un software creado por MikroElektronika para el desarrollo de interfaces gráficas con pantallas TFT que genera código para ser compilado directamente por sus compiladores, esto acelera y facilita mucho el diseño de interfaces con pantallas táctiles.

Si bien considero este trabajo como introductorio a la programación en el lenguaje C, encontrará una gran cantidad de ejemplos, rutinas de programación, librerías y textos explicativos sobre una diversidad de temas que pueden ser de utilidad no solo en el proceso de entender cada una de las arquitecturas, sino también para aplicar en desarrollos electrónicos en general.

Todos los ejemplos propuestos están pensados para su realización práctica con electrónica real, esto no significa que no pueda usar simuladores para corroborar su funcionamiento sin embargo la experiencia nos enseña que no siempre los resultados obtenidos en el simulador coinciden con el comportamiento en el terreno físico.

Recuerde que la simulación se ejecuta en un entorno ideal (la memoria del computador), sin ruidos eléctricos ni interferencias del mundo real.

Comentarios del autor.

Como casi todos los que contamos con algunos años en electrónica, iniciamos nuestro camino en la programación de la mano de Motorola, sin duda confiables y poderosos microcontroladores pero trabajar

con ellos en aquellos lejanos tiempos era todo un desafío, borrado ultravioleta, voltajes de programación altos, procesos lentos y tediosos, el lenguaje era ensamblador, doscientos comandos que alguna vez supimos escribir de corrido sin consultar la tabla de referencia, algo normal para la época.

Cuando los microcontroladores PIC hicieron su aparición estos ofrecían la posibilidad de programación y borrado eléctrico gracias a su memoria EEPROM, ya no se necesitaba luz ultravioleta ni un complejo hardware de programación, incluso se podía armar el programador con componentes simples que se adquieren en cualquier negocio de electrónica.
Pero claro, si algo evoluciona rápido esto es la tecnología y pronto los ocho bits sonaron a poco para las exigencias crecientes del mercado electrónico y así llegamos a los nuevos núcleos de 32 bits con arquitectura ARM que también tratamos en las siguientes páginas.

No se espera que al terminar la lectura de este libro usted sea un experto programador en C, pero sí que tenga una idea clara de lo que es el lenguaje con uno de los compiladores más simple de usar.
Un comentario que siempre hago a mis alumnos es que la evolución es un paso hacia adelante, un cambio superador y en el campo de la electrónica programable eso es C.
A partir de este punto dependerá de usted hasta donde quiera llegar y que tan largo será el camino.

Gracias por permitirme ser parte de ese camino.

Capítulo I

Los PIC's y un poco de su historia.

Los PIC son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instrument.

El nombre actual no es un acrónimo, en realidad el nombre completo es PICmicro, aunque generalmente se utiliza como **P**eripheral **I**nterface **C**ontroller (*Controlador de Interfaz Periférico*).

El PIC original se diseñó para ser usado con la CPU CP16000. CPU que en general tenía prestaciones pobres de entradas y salidas, el PIC de 8 bits se desarrolló en 1975 para mejorar el rendimiento de la CP16000 mejorando las prestaciones de entradas y salidas.

El PIC utilizaba microcódigo simple almacenado en ROM para realizar estas tareas.

Aunque el término no se usaba esos años, se trata de un diseño RISC que ejecuta una instrucción cada 4 ciclos del oscilador.

En 1985 la división de microelectrónica de General Instrument se separa como compañía independiente y cambia el nombre a Microchip Technology.

El PIC, sin embargo, se mejoró con EPROM para conseguir un controlador de canal programable. Hoy en día multitud de PIC´s vienen con varios periféricos incluidos (módulos de comunicación serie,

UART, núcleos de control de motores, etc.) y con memoria de programa desde 512 a 32000 palabras (una palabra corresponde a una instrucción en lenguaje ensamblador, y puede ser de 12, 14, 16 o 32 bits, dependiendo de la familia específica de PIC.
El PIC usa un juego de instrucciones, cuyo número puede variar desde 35 para PIC de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada *sleep*.
Microchip proporciona un entorno "*Oficial*" de desarrollo libre llamado MPLAB que incluye un simulador software y un ensamblador. En nuestro caso estaremos usando MikroC Pro para PIC. Un IDE mas compilador capaz de generar código prácticamente para todos los PIC.

Para transferir el código desde una PC al PIC normalmente se usa un dispositivo llamado programador. La mayoría de los PIC´s que Microchip distribuye incorporan ICSP (*In Circuit Serial Programming, programación serie incorporada*) o LVP (*Low Voltage Programming, programación a bajo voltaje*), lo que permite programar el PIC directamente en el circuito destino.
Para la ICSP se usan los pines RB6 y RB7 como reloj y datos (En algunos modelos pueden usarse otros pines como el GP0 y GP1 o el RA0 y RA1), el MCLR para activar el modo programación aplicando un voltaje de 13 voltios.

Existen muchos programadores de PIC, desde los más simples que dejan al software los detalles de comunicaciones, a los más complejos, que pueden verificar el dispositivo a diversas tensiones de alimentación e implementan en hardware casi todas las funcionalidades. Muchos de estos programadores complejos incluyen ellos mismos PIC pre-programados como interfaz para enviar las órdenes al PIC que se desea programar.

La arquitectura se dirige a la maximizar la velocidad, PIC fue uno de los primeros diseños escalares de CPU, y sigue siendo uno de los más simples y más baratos. La arquitectura de Harvard en el que las instrucciones y los datos provienen de diferentes fuentes-simplifica la sincronización y diseño de microcircuitos.
El conjunto de instrucciones es adecuado para la aplicación de tablas de búsqueda rápida en el espacio de programa. Estas búsquedas tienen una instrucción y dos ciclos de instrucción. Muchas de las funciones se pueden modelar de esta manera.

Como funciona un Microcontrolador PIC.

Para entender un poco como funcionan las cosas dentro de un microcontrolador podríamos comenzar definiendo sus partes.

CPU

Dentro de la CPU está la Unidad de Control (UC) quien se encarga de que todo suceda de acuerdo a lo previsto, la UC es pues quien decide cuando y como las acciones se llevan a cabo.
Se lee una instrucción desde la memoria de programa se la pasa al decodificador de instrucciones y el contador de programa (PC) se in-

crementa en uno para leer en la siguiente dirección en la memoria de programa.

También dentro de la CPU encontramos la ALU que es la encargada de resolver los problemas matemáticos y lógicos, cuando por ejemplo hay que resolver una operación de suma, la UC le pasa a la ALU las variables esta lo resuelve y le entrega los resultados a la UC para que esta disponga de ellos, entonces la UC consulta el programa para saber que debe hacer con el resultado.

I/O Ports.

Controlan los pines que son el vínculo con el exterior, estos pines se pueden configurar como entradas o salidas dependiendo de la necesidad.

Memoria de Programa.

Es la memoria donde reside el código de la aplicación. Si por ejemplo hemos programado una alarma, es en esta memoria donde el código de la alarma reside.

La memoria de programa es la que programamos con nuestro programador, esta memoria es del tipo FLASH.

Memoria Ram.

La memoria RAM es una memoria de lectura y escritura pero temporal es decir los datos en ella se mantienen el tiempo que se haya energía en el sistema, si la memoria se queda sin energía la información se pierde. En un microcontrolador es la memoria destinada a guardar las variables temporales de proceso, variables de programa

que solo son importantes en ese momento. Todos los microcontroladores tienen memoria RAM, algunos más otros menos pero todos disponen de esta memoria, no solo es usada por el programa para guardar los datos generados en tiempo de ejecución sino que también los registros propios del microcontrolador radican en RAM, registros de configuración, registros de estado, se puede decir que la CPU se comunica con nosotros a través de estos registros son los llamados SFR (Registro de Funciones Especiales).

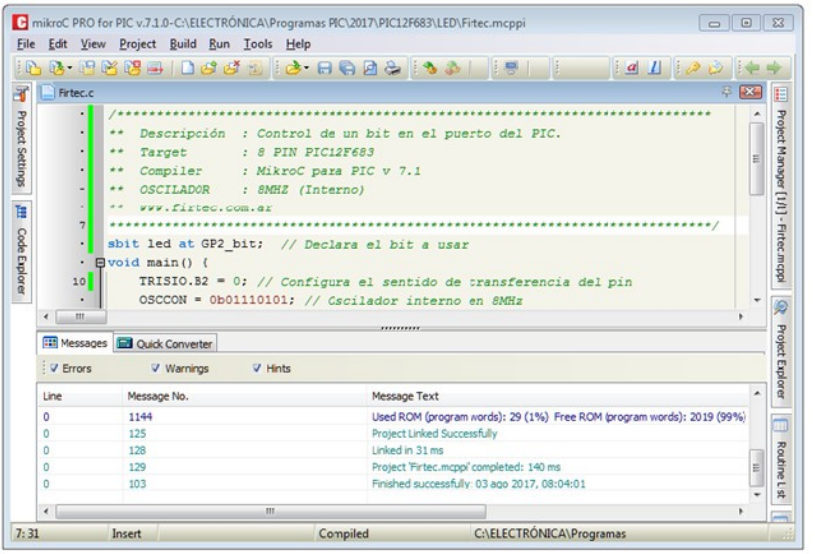
Stack de Memoria o Pila.

El stack es memoria RAM donde el controlador guarda datos temporales de uso propio de la CPU, no depende del programador y es manejado por el controlador, tiene en la serie 18 de pic´s 31 niveles de profundidad lo que mejora notablemente el anidamiento de interrupciones que se verá mas adelante.

Tradicionalmente la pila o stack, sólo se ha utilizado como un espacio de almacenamiento para las direcciones de retorno de subrutinas o rutinas de interrupción, en donde las operaciones para meter y sacar datos de la pila estan escondidos.

En su mayor parte, los usuarios no tienen acceso directo a la información en la pila.
El microcontrolador PIC18 se aparta un poco de esta tradición. Con el núcleo PIC18 nuevo, los usuarios tienen ahora acceso a la pila y se puede modificar el puntero de pila y pila de datos directamente. Habiendo tales niveles de acceso a la pila permite algunas posibilidades de programación única e interesante.

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.



Aspecto del IDE para MikroC PIC.

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 de que dispone.

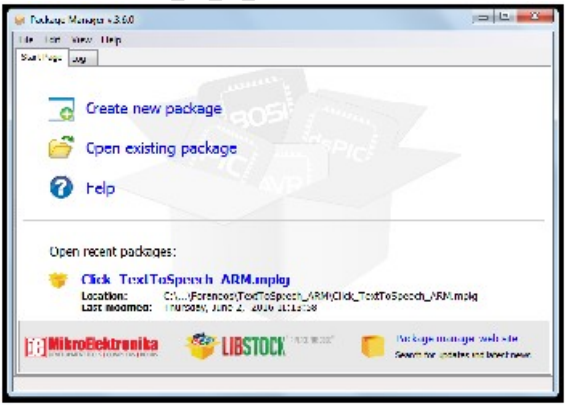
Una particularidad interesante que tiene MikroC es la gran cantidad de bibliotecas y ya listas para usar que se encuentran incorporadas en el propio entorno de trabajo.
Rutinas para el manejo de una pantalla LCD, una comunicación UART, I2C, SPI, incluso bibliotecas para el manejo del ADC, 1-Wire,

entre muchas mas opciones están disponibles, solo debemos marcar la biblioteca y las funciones de control quedan listas para usar. Cada biblioteca cuenta con una ayuda que explica en detalle las funciones disponibles y su funcionamiento.
Las bibliotecas se descargan como paquetes de software y se instalan con un instalador especial, una herramienta que también descargamos de manera libre desde el sitio de Mikroelektronika.

Instalador de paquetes de Mikroelektronika.

El instalador de paquetes es una herramienta que nos permitirá tanto instalar como desinstalar software del entorno de Mikroc.

Para instalar un paquete solo debemos descargarlo y luego con el instalador de paquetes instalarlo, una vez instalado ya lo tendremos listado en el administrador de librerías.
Para desinstalarlo también se usará el administrador de paquetes.
Hay varias herramientas interesante para descargar desde el sitio de Mikroelektronika que funcionan incluso en distintas arquitecturas ya que hay varios compiladores para distintos lenguajes y arquitecturas.



Instalador de paquetes

adiciones sucesivas (a x b = a + a + a + … + a).

En lenguajes como C para multiplicar los números a y b, basta con escribir a*b y el compilador encontrará automáticamente la solución a éste problema y otros similares.

Lenguaje C.

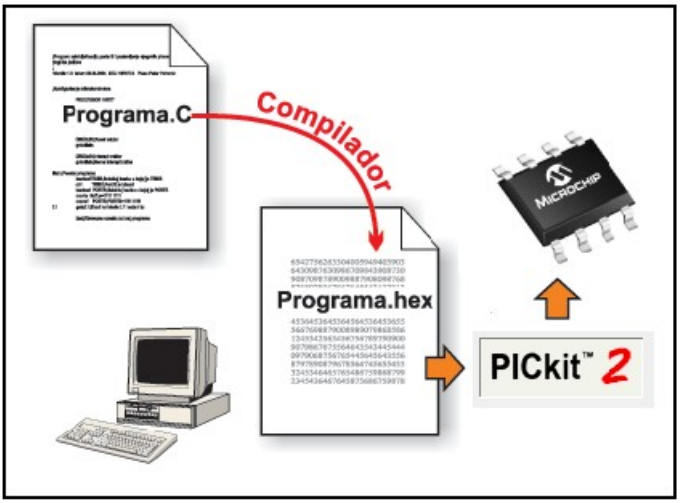
C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, en la actualidad es la principal herramienta de programación para electrónica y el desarrollo de sistemas embebidos con microcontroladores.

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel trabajando directamente con el Hardware.
La primera estandarización del lenguaje C fue en ANSI o ANSI C y uno de los objetivos de diseño del lenguaje C es que sólo sean necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que haga falta un soporte intenso en tiempo de ejecución.

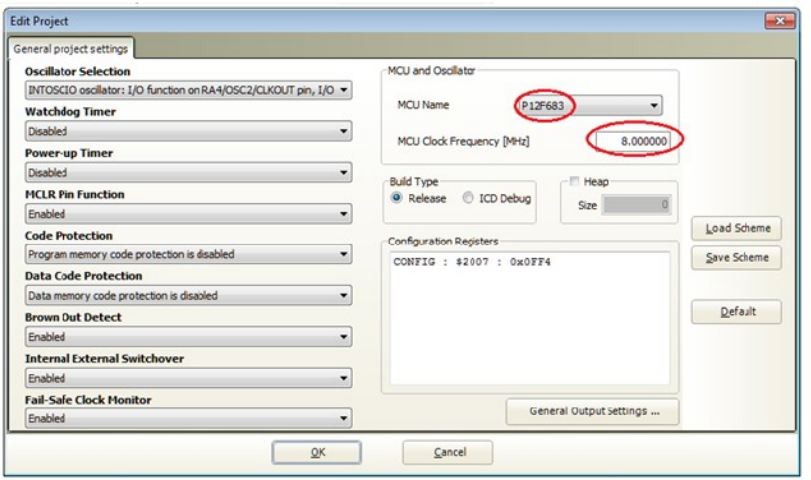
En consecuencia, el lenguaje C está disponible en un amplio abanico de plataformas (seguramente más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente del Hardware.

Activando Pickit desde la barra de herramientas.

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



Proceso de programación a nivel de archivos.



Configuración para el hardware interno del PIC.

MikroC facilita mucho esta configuración ya que en la edición del proyecto podemos definir todos los valores de configuración para el hardware del PIC. Seleccionamos el tipo de PIC que usaremos en el proyecto, su velocidad de reloj, Watchdog, tipo de reloj protección de memoria, etc.

Configuración de puertos.

Los puertos son el punto de entrada para información desde el exterior y también sacar información desde el controlador hacia el mundo exterior.

Si queremos leer datos del mundo exterior el pin del puerto se configura como entrada si lo que buscamos es sacar un dato hacia el exterior el pin se configura como salida. Todos los pines de los puertos se pueden configurar como entradas o salidas *(Excepto los pines involucrados en el USB)*.

00001111 en el registro *ADCON1* dejando los pines en modo analógico.

PCFG3:PCFG0: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

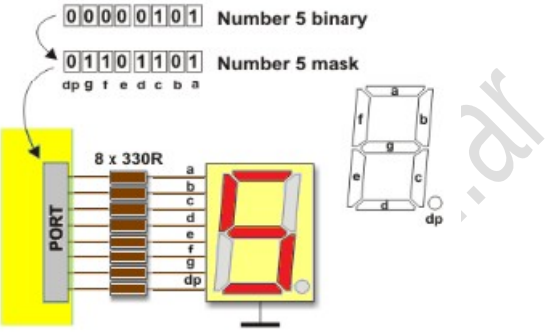
Posibles configuraciones para los pines analógicos.

En la imagen anterior se puede ver parte del registro *ADCON1* que tiene cuatro bits, *PCFG0/1/2/3* y de acuerdo al valor de estos bits será la configuración de pines analógicos o digitales.

Recuerde:

Los pines por defecto son analógicos, si conecta un pulsador para registrar el cambia de estado no funcionará, debe configurar el pin como digital.

Veamos un ejemplo, en este programa vamos a visualizar el estado de un contador a través de un display de siete segmentos tipo cátodo común conectado en el puerto B de un PIC18F4620.



Circuito para conectar un display de siete segmentos cátodo común.

```
/************************************************
**  Descripción  : Contador de un dígito. Display
**  conectado en puerto B.
**  Target       : PIC18F4620 de 40 pines
**  Compiler     : MikroC para PIC v7.1
**  XTAL         : 20MHZ
************************************************/

// Variable en RAM usada para contar
unsigned char contador=0;

const unsigned char Display[16] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x67,
0x77,0x7C,0x39,0x5E,0x79,0x71};

void main() {
```

Podemos declarar un array de números enteros *int numeros[100] {0,1,2,3,..99}* esto declara una cadena de 100 números el índice 0 contiene el número cero.

Si el array tiene 16 elementos, la variable (contador en este caso) en si misma puede contener muchos mas elementos dependiendo de su tipo, sin embargo solo puede direccionar 16 elementos que son los que están declarados dentro del array, si direccionamos mas elementos los resultados son impredecibles, esto suele ser un error frecuente donde los programas muestran un comportamiento errático, son errores en tiempo de ejecución y pueden ser muy molestos a la hora de resolverlos.

En el ejemplo anterior la variable contador es del tipo ***unsigned char*** lo que permite contar 255 elementos pero solo podríamos direccionar los que están declarados en el array.

Tipos de datos en Mikroc.

MikroC maneja distintos tipos de datos o tipos de variables como se puede ver en la siguiente imagen.

Tipo	Tamaño en bytes	Rango
(unsigned) char	1	0 .. 255
signed char	1	- 128 .. 127
(signed) short (int)	1	- 128 .. 127
unsigned short (int)	1	0 .. 255
(signed) int	2	-32768 .. 32767
unsigned (int)	2	0 .. 65535
(signed) long (int)	4	-2147483648 .. 2147483647
unsigned long (int)	4	0 .. 4294967295

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.

do-while.

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

```
do{
                                código 1
                                código 2
                                código n
while(PORTA.RA0!=1 && bandera== 0 );
```

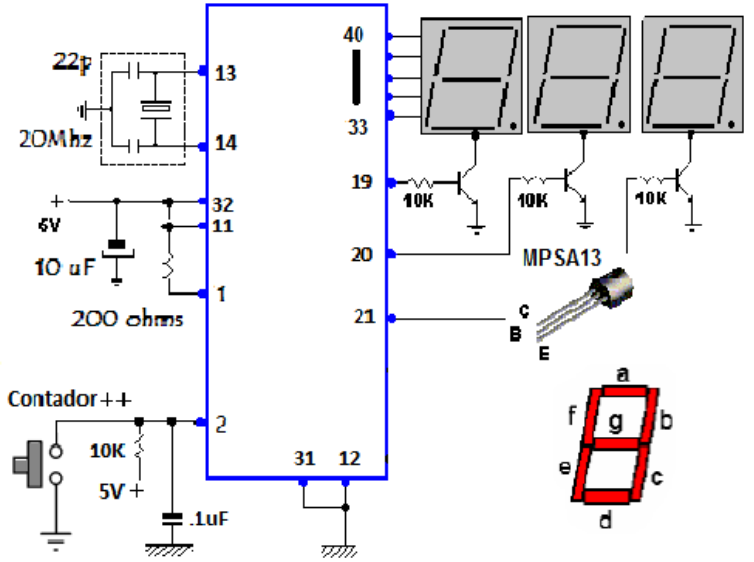
for().

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

```
for(a=0;a<=90;a++){
    // Ejecutar esta línea 100 veces!!!
}
```

switch().

```
    bandera=0; // Un número por vez.

  }
}
```

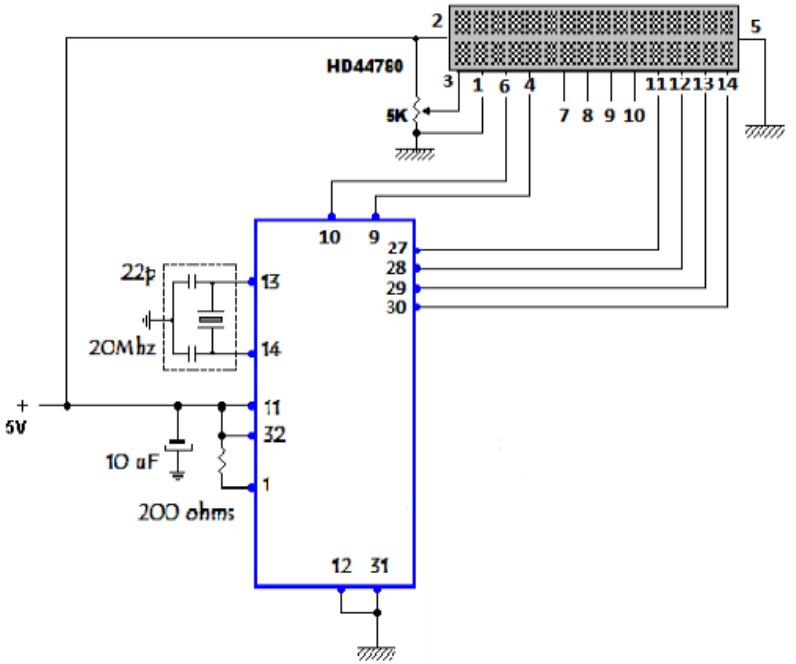


Circuito del contador.

Control del LCD (HD44780).

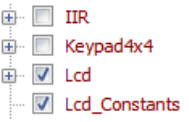
Básicamente podemos imaginar una pantalla LCD como una memoria de 40 caracteres de los cuales solo 16 son visibles (pantallas 16x2).

En nuestro ejemplo vemos como implementar un controlador para la pantalla Hitachi 44780 de dos líneas por 16 caracteres por línea.

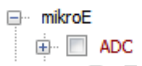


Circuito propuesto para el ejemplo

Para el manejo de la pantalla necesitamos contar con el respectivo driver que MikroC agrega a su biblioteca solo tenemos que marcarlo y todas las funciones del LCD estarán disponibles para su uso.
La siguiente imagen muestra como activar la correspondiente biblioteca.



Se propone un ejemplo simple usando el LCD, leer la temperatura de un sensor analógico LM35 conectado al pin 2 (*RA0*) configurado como analógico. Para usar el conversor analógico necesitamos el driver o biblioteca que maneja el hardware interno del microcontrolador.
Este driver también se encuentra en la biblioteca de MikroC y basta con marcar el casillero correspondiente como se ve en la siguiente imagen.



El driver del LCD tiene muchas funciones para el control total de la pantalla, colocando el cursor sobre el signo "+" se obtiene la lista de funciones y su funcionamiento explicado.
El driver necesita que en nuestro programa se declaren los pines que serán usados por la pantalla, para que todo funcione correctamente lo primero que encontramos es la declaración de estos pines.

```
/************************************************

Descripción  : Mide la temperatura con un sensor
LM35 y muestra los datos en una pantalla LCD 16x2
Target       : PIC18F4620 de 40 pines
Compiler     : MikroC para PIC
XTAL         : 20MHZ

************************************************/
```

```
    mV = mV/10.0; // temperatura, grados Celsius

    FloatToStr(mV,txt);
     // Temperatura se convierte a cadena

    txt[4]=0;

    Lcd_Out(2, 6, txt);
     // LCD muestra la temperatura

    Delay_ms(500); // Espera medio segundo.

  } while(1);

}
```

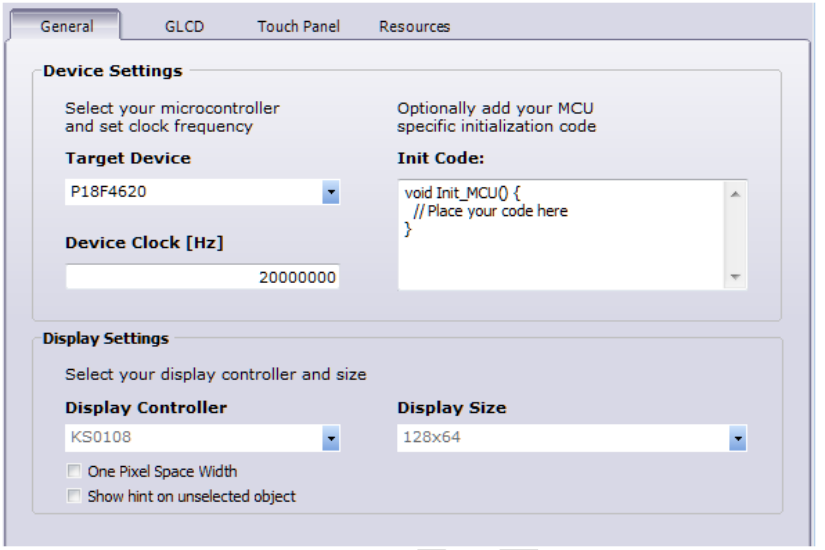
Algunas consideraciones del programa anterior.

- No todos los *LCD Hitachi 44780* compatibles tienen los mismos caracteres almacenados en ROM, puede suceder que intente mostrar el símbolo grados en la función *Lcd_Chr(2,10,223)* el resultado no sea el esperado.
- La función *ADC_Read(0)* lee el canal analógico 0 y guarda la lectura en la variable *temperatura.*
- La lectura del conversor analógico se realiza cíclicamente al ritmo de 500 mS debido a que la función de lectura y tratamiento del dato del conversor está contenido dentro de un *bucle do while*.

KS0108 o compatibles (128x64 pixeles).

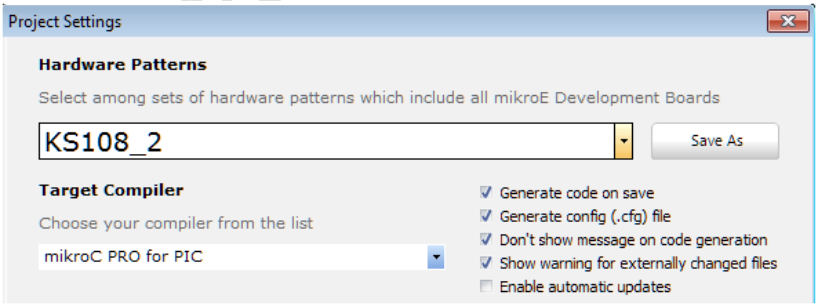
Este GLCD está basado en el controlador Hitachi HD6102 o el clon de este el Samsung KS0108.

Tiene dos controladores, uno controla la parte izquierda y el otro la

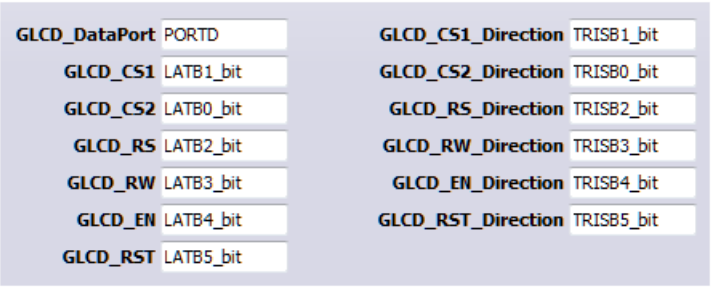


Configuración del hardware para una pantalla KS0108.

En la configuración de Visual GLCD indicamos el tipo de micro y su velocidad, también el controlador que tiene la pantalla gráfica a usar, la siguiente imagen muestra la configuración.



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 más para trabajar con esta pantalla.

Visual GLCD soporta varios tipos de pantallas, el KS108 es simple y económica, las pantallas con *Touch* funcionan igual de bien considerando que estamos trabajando con microcontroladores de 8 bits y velocidades no muy elevadas es por esto que no debemos esperar resultados espectaculares cuando usamos pantallas *Touch* .

En este simple ejemplo se puede ver el aspecto de la pantalla incorporando una imagen de fondo. La imagen debe ser monocromática y ajustada con relación al tamaño de la pantalla.

Interrupciones con MikroC.

Las interrupciones son señales recibidas por el microcontrolador, indicando que debe "interrumpir" el curso de ejecución actual. Una interrupción suspende la ejecución temporaria de un programa, para pasar a ejecutar una subrutina de servicio de interrupción (*ISR*). Las interrupciones surgen de las necesidades que tienen los periféricos de enviar información al controlador, por ejemplo el controlador está ejecutando el programa de una balanza y una línea de interrupción está conectada a un sensor que mide la presión de vapor de una caldera y en determinado momento la presión baja, el sensor detecta el problema y activa la interrupción, el controlador interrumpe su tarea y salta a la *ISR* que trata el evento del vapor.

Los microcontroladores PIC´s tienen fuentes múltiples de interrupción y muchos tienen una prioridad de interrupción que permite que cada fuente de interrupción se le asigne un nivel de alta o baja

activando el bit IPEN (RCON<7>).
La siguiente imagen muestra la estructura del registro de control RCON.

RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(1)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

Estructura del registro de control INTCON.

Cuando se permite la prioridad de las interrupciones, hay dos bits que permiten las interrupciones globalmente. Activando el bit **GIEH (INTCON<7>)** se permiten todas las interrupciones que tengan alta prioridad.

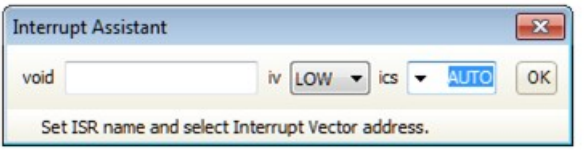
bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16XXX Compatibility mode)

Activando el bit **GIEL (INTCON<6>)** permite todas las interrupciones que tengan baja prioridad.

Cuando la bandera de interrupción esta borrada y la máscara de interrupción este activo para esa interrupción en particular y el bit de las interrupciones globales activo, cuando ocurra una interrupción el programa saltará inmediatamente a la dirección **0x0008 ó 0x0018**, dependiendo del bit de prioridad.

La dirección de retorno se pone en la pila (*stack*) y el Contador de Programa *(PC)* se carga con la dirección de interrupción (*0x08 ó 0x018*). Una vez en la rutina de interrupción (ISR), las fuentes de interrupción se pueden determinar interrogando las banderas de interrupción.
Recordar*:*

- *Las banderas de interrupción se tienen que limpiar por software antes de volver a permitir las interrupciones para evitar interrupciones recurrentes.*
- *En MikroC tenemos un asistente para generar la rutina de interrupción. Las teclas* ***Alt + I*** *despliega un menú mostrado en la siguiente imagen donde completamos el nombre de la rutina de interrupción y su nivel de prioridad con esto automáticamente se genera la ISR en nuestro código. El programador deberá completar la función con la acción que debe realizar la interrupción*



Observe que si declara una ISR de baja prioridad deberá configurar correctamente el registro de prioridades, por defecto el controlador asume que no hay prioridad por lo que todas las ISR siempre apuntan al vector 0x08, vector de alta prioridad.

Para tener en cuenta:

Las ISR son funciones como cualquier otra, pero con algunas particularidades:

- *No devuelven ni aceptan parámetros, siempre serán void.*
- *Solo se puede invocar desde una interrupción.*
- *Primero se activa la máscara individual de cada interrupción y luego el habilitador general de interrupciones.*
- *Las variables globales que utilice se deben declarar como volatile.*
- *Todas las interrupciones apuntan a la misma dirección de memoria, todas las de baja prioridad irán a la dirección 0x018 y todas las de alta prioridad a la dirección 0x008, la única forma de saber cual interrupción se activó es evaluando la bandera de interrupción que cada periférico tiene asignada.*

Veamos un ejemplo de interrupción externa por el pin RB0 (*pin33 INT0*). Esta interrupción generada por hardware puede ser configurada para que se active por flanco de subida o bajada. Un *LED* colocado en *RC2* muestra la interrupción cuando se activa.

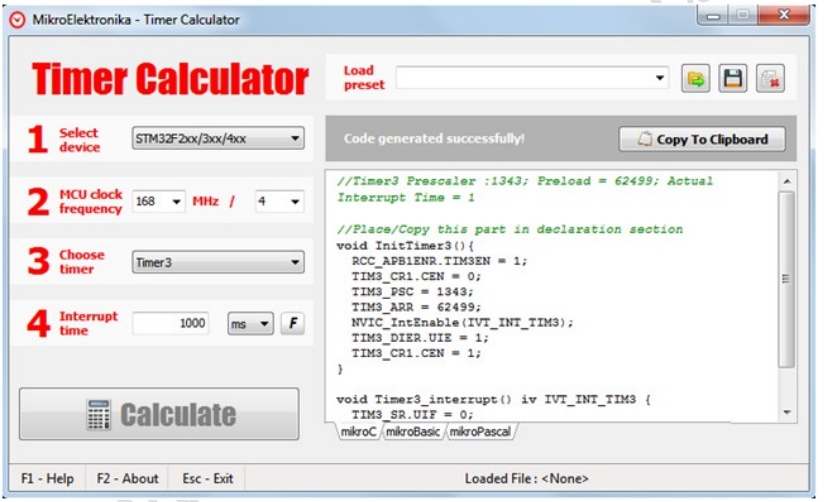
```
/************************************************

Descripción  : Ejemplo de una interrupción externa
por INT0 (pin33)
Target       : PIC18F4620 de 40 pines
Compiler     : MikroC para PIC v7.1
XTAL         : 20MHZ

************************************************/

void interrupt(){   // Servicio de interrupción

  if(INT0F_bit == 1 ) {
// Verifica la bandera de interrupción
```

```
    }
}
```

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.

En la siguiente imagen podemos ver el aspecto de esta herramienta.



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.

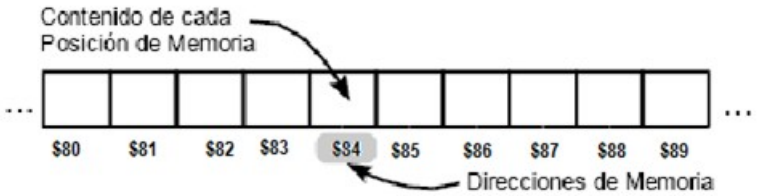
```
    break;
     }
    case 4:{
        PORTD.RD0 = 0;   // Pone a 0 el pin 19
        marca =0;
    break;
  }
 }
}
```

La función mostrar enciende cada dígito en una secuencia definida por la variable *marca* que es modificada en la propia interrupción. La función *Switch()* selecciona el dígito a encender de acuerdo a lo indicado por *marca*.
Observe que este código respecto al contador visto en páginas anteriores, la función mostrar no tiene tiempos muertos como los retardos necesarios para encender cada dígito, el tiempo de encendido lo determina el tiempo generado en el timer, la atención del controlador queda disponible para hacer otras tareas y no estar "*enroscado*" en bucles muertos de retardo como los *delay_ms(x)*.

Punteros con C.

En el famoso libro de Kernighan y Ritchie "*El lenguaje de programación C*", se define un puntero como ***una variable que contiene la di-***

rección de una variable. Tras esta definición clara y precisa, podemos remarcar que un puntero es un tipo especial de variable que almacena el valor de una dirección de memoria. La memoria de los microcontroladores está formada por un conjunto de bytes. Simplificando, cada byte tiene un número asociado, su dirección en esa memoria.



Un puntero es un apuntador a una posición de memoria.

La declaración de un puntero es igual al de una variable a excepción que lleva un asterisco como prefijo.

*unsigned char *pnum;*

También se pueden mezclar declaraciones con variables, como se dijo anteriormente, un puntero también es una variable.

*unsigned char *pnum, num = 30;*

Una convención muy utilizada en lenguaje C es que la variable tipo puntero siempre empiece con la letra p, que simboliza un puntero.

Hasta ahora hemos declarado el puntero, pero no le hemos asignado a quien verá en la memoria, es decir, estará flotando, si tratamos de usarlo puede apuntar a cualquier lugar y no queremos que suceda esta anomalía, sino que apunte a un lugar deseado. Utilizando el

Funciones.

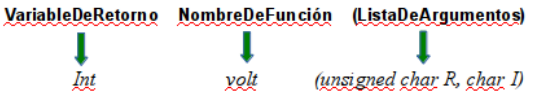
Una función es un trozo de código que se puede invocar muchas veces dentro de un programa, lo que en lenguajes de bajo nivel llamaríamos *subrutina*.

Una razón de crear funciones es la de hacer un programa más corto creando módulos con una utilidad específica. Un programa complejo puede especificar muchas funciones que hacen su resolución más simple, hace la vida del programador de proyectos un poco más sencilla.

Una función tiene un encabezado que declara una función, luego sigue su argumento entre paréntesis y entre llaves la definición de la función. Veamos el siguiente ejemplo de una función que retorna el resultado de la ley de Ohm (V = RI).

```
int volt(unsigned char R, char I)  /* Nombre de la función y sus argumentos de trabajo */
{

int voltaje;                       /* Variable Temporal */

voltaje = R * I;                   /* Ley de Ohm */

return voltaje;                    /* Retorna el resultado */

}
```

La estructura general del encabezado de una función es:



impedancia del cambio del muestreo interno (RSS) afectan directamente al tiempo requerido para cargar este condensador. La impedancia recomendada máxima para las fuentes analógicas es de 2,5kΩ. Cuando la impedancia decrece, el tiempo de adquisición también decrece. Después de que el canal analógico haya sido seleccionado este tiempo de adquisición debe tenerse en cuenta antes de arrancar la conversión.
El cálculo detallado puede observarse en el datasheet del PICs utilizado, a modo de ejemplo vemos que en un 16F87XA el mínimo tiempo de adquisición es de 19.72us el cual debe ser generado por software cada vez que se seleccione un canal.
En cambio para los PIC18Fxx el tiempo mínimo de adquisición se reduce a 2.45us.

El tiempo por bit de la conversión A/D se define como TAD. La conversión A/D necesita un mínimo de 12TAD para la serie 16F o 11TAD para la serie 18F con 10 bits de conversión.
La fuente de reloj A/D de la conversión es seleccionable por software. Hay siete opciones posibles.

- *2 TOSC.*
- *4 TOSC.*
- *8 TOSC.*
- *16 TOSC.*
- *32 TOSC.*
- *64 TOSC.*
- *Oscilador interno RC*

Para el TAD las opciones posibles son:

- *20 x TAD.*
- *16 x TAD.*

- *12 x TAD.*
- *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**

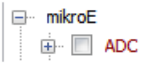
Recordar:

- ***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.

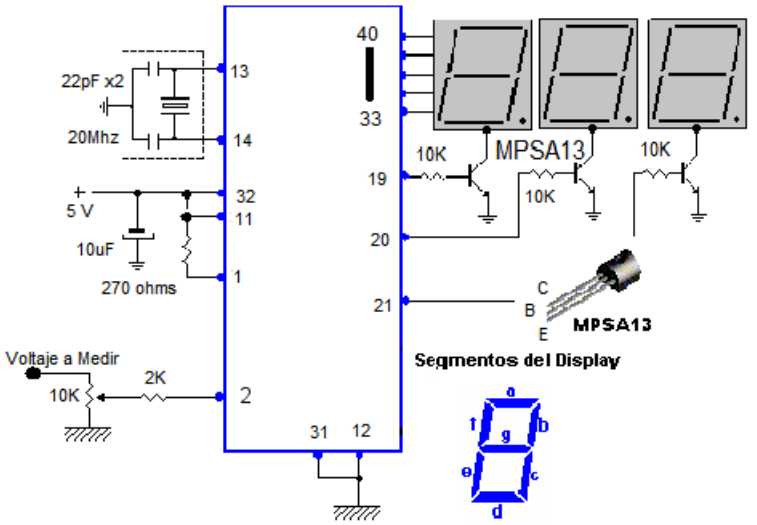
Si uso 96MHz/2=48MHz
*Tad=64*Tosc=64/Fosc=64/48MHz=1.3333us*
Utilizo 96MHz/4=24MHz porque
*Tad=32*Tosc=32/Fosc=32/24MHz=1.3333us*
Utilizo 96MHz/6=16MHz porque
*Tad=16*Tosc=16/Fosc=16/16MHz=1us*
Pero el tiempo de cada ciclo aumenta lo que disminuye el tiempo total de conversión
Si uso un cristal de 20MHz,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

XTAL         : 20MHZ

Autor        : Firtec - www.firtec.com.ar
************************************************/
void Leer_Conversor(void);
// Se declara la función que leerá el conversor A/D

void Mostrar_Conversor(void);
// Función que mostrará los datos

unsigned char contador =0;

unsigned char unidad=0, decena=0, centena=0;

unsigned int conversion=0;


// Tabla para mostrar los dígitos

const unsigned char Digito[16] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,

 0x67,0x77,0x7C,0x39,0x5E,0x79,0x71};
```

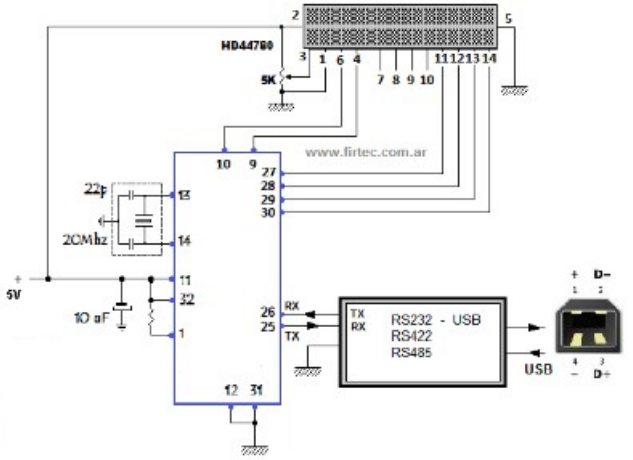


Circuito propuesto para la aplicación.

En el ejemplo anterior se muestra el uso del conversor analógico, el ejemplo lee el canal 0 (pin 2) de un microcontrolador PIC18F4620 y muestra el resultado de la medición en tres dígitos en formato hexadecimal mostrando números que van desde 000 a 3FFF correspondiendo al número mas alto que puede generar el conversor de 10 bits.

El mismo ejemplo anterior controlado por la interrupción del conversor analógico.

```
/************************************************

Descripción  : Voltímetro Hexadecimal [000 a 3FF]
por interrupción
```



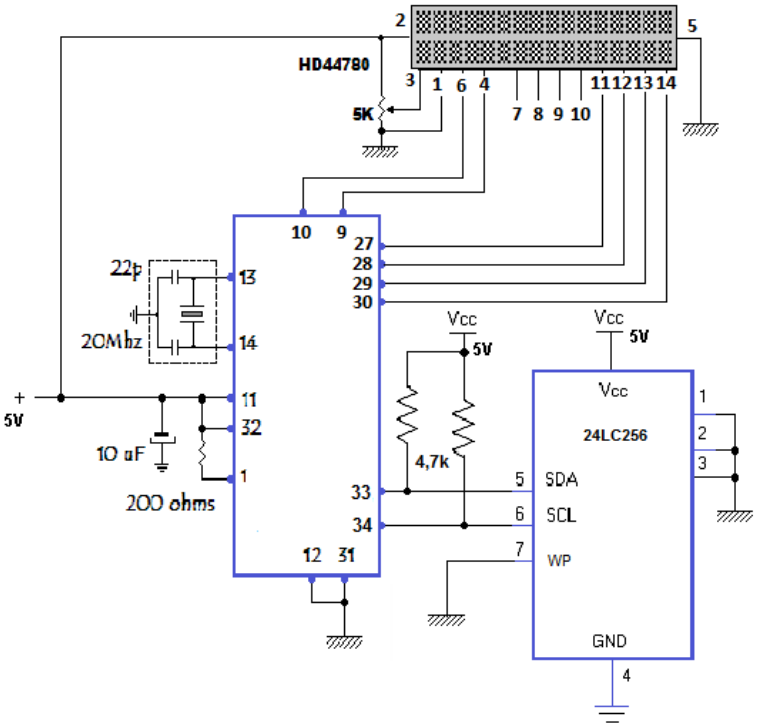
En la imagen anterior podemos ver el circuito propuesto para el ejemplo. Para visualizar los datos recibidos podemos usar cualquier aplicación RS232 como el Hiper Terminal, CuteCom, etc.

El protocolo I2C.

Diseñado por Philips, este sistema de intercambio de información a través de tan solo dos cables permite a circuitos integrados y módulos OEM interactuar entre sí a velocidades relativamente lentas. Emplea comunicación serie, utilizando un conductor para manejar el reloj y otro para intercambiar datos.

Para su uso en MikroC debemos habilitar el driver desde la biblioteca marcando el casillero correspondiente para tener acceso a las funciones del I2C.

Este bus se basa en tres señales:



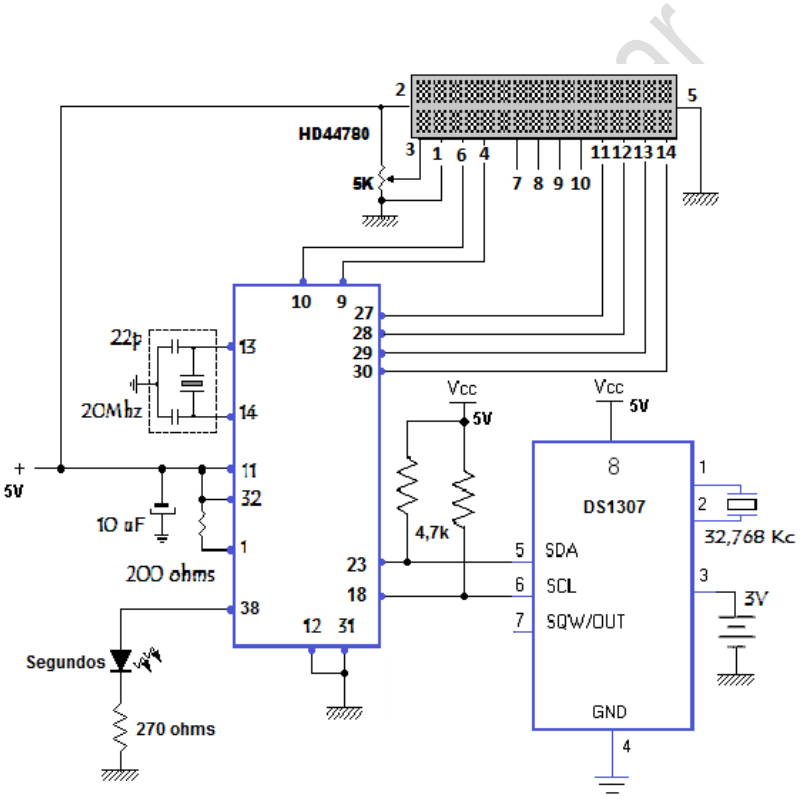
Esquema electrónico para el ejemplo.

RTC DS1307 (Real Time Clock).

Siguiendo con los dispositivos I2C el ***DS1307*** de Dallas Semiconductor (Maxim) es una solución muy interesante cuando necesitamos trabajar con eventos que requieren puntualidad y **exactitud a lo largo del tiempo**. Este pequeño circuito integrado es uno de los más populares **relojes RTC** del mercado por su sencillez de uso y por su confiabilidad a largo plazo. Preparado para ofrecer la hora hasta el año 2100

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 para el ejemplo.

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 que pueden ser adheridas o incorporadas a un producto, un animal o una persona. Contiene la antena para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor RFID. Las etiquetas pasivas no necesitan alimentación eléctrica interna, mientras que las activas sí. Una de las ventajas del uso de radiofrecuencia en lugar de otras tecnologías, es que no se requiere visión directa entre emisor y receptor.

Origen de los RFID.

Los orígenes de la tecnología RFID están un poco confusos, se ha dicho que un dispositivo similar a RFID pudo haber sido una herramienta de espionaje usada por el gobierno soviético a fines de la segunda guerra mundial, también existieron dispositivos con un vago parecido para la identificación de aviones “amigos” durante esta gran guerra.

Otros trabajos tempranos sobre RFID se mencionan desde 1940, sin embargo su verdadero desarrollo y uso ha ocurrido en los últimos veinte años. En el campo de la domótica para el control de acceso a viviendas o control de sistemas de alarmas, para el control y seguimiento de objetos, en veterinaria para identificar animales y también en la actualidad para su uso en personas, esto último con algunos puntos grises respecto al derecho a la privacidad .

Frecuencias en distintos países.

Los distintos países han asignado diferentes partes del espectro radial para RFID, de manera que ninguna tecnología en sí satisface óptimamente todos los requerimientos de los mercados. La industria ha trabajado para estandarizar las principales bandas de Radio.

- *Baja frecuencia (LF), de 125 a 134 kHz.*
- *Alta frecuencia (HF) de 13.56 MHz.*
- *Frecuencia Ultra Alta (UHF), de 860 a 960 MHz.*

La mayor parte de los países han asignado las zonas de 125 o 134kHz del espectro para los sistemas de baja frecuencia. La zona de 13.56 MHz se usa en todo el mundo para los sistemas de alta frecuencia (con algunas excepciones), pero los sistemas de UHF existen solamente desde la segunda mitad de la década de los años 90 y los países no han acordado una sola zona del espectro de la UHF para la RFID.

El ancho de banda en la Unión Europea va desde los 865 hasta los 868 MHz, con los interrogadores con la capacidad de transmitir a máxima potencia (2 vatios ERP) en el centro del ancho de banda (865.6 hasta 867.6 MHz). El ancho de banda para la RFID de UHF en América del Norte, va desde los 902 a los 928 MHz, con lectores capaces de transmitir a máximo poder (1 vatio ERP) para el mayor rango de ese ancho de banda. Australia ha asignado el rango de 920 a 926 MHz para la tecnología RFID UHF. Los canales de transmisión europeos están restringidos a un ancho máximo de banda de 200 kHz, en contraposición a 500 kHz, en Norte América. China ha aprobado anchos de banda en los rangos de 840.25 hasta 844.75 MHz y entre 920.25 hasta 924.75 MHz para las etiquetas y lectores de UHF que se utilizan en ese país. Hasta recientemente Japón no permitía

ningún espectro de UHF para la RFID, pero está estudiando el abrir con este fin la zona de los 960 MHz. Muchos otros dispositivos usan el espectro de la UHF, así que tomará años para que todos los gobiernos acuerden establecer una sola banda de UHF para la RFID.

Cantidad de información almacenada en una etiqueta de RFID.

Esto depende del distribuidor, la aplicación y el tipo de etiqueta, pero generalmente la etiqueta no almacena más que 2 kilobytes (KB), suficiente para almacenar algo de información básica sobre el objeto al que está adosado o vinculado. Las etiquetas simples contienen solamente un número de serie de 96 o 128 bits. Las etiquetas simples son más fáciles de fabricar y son más útiles para aplicaciones en las que la etiqueta será desechada con el empaquetado del producto.

Etiquetas de lectura y lectura/escritura.

Los microchips de las etiquetas de RFID pueden ser de lectura-escritura, de lectura únicamente o de "escribir una vez, leer muchas veces" (WORM). Con los chips de lectura-escritura, se puede agregar información a la etiqueta o escribir sobre la información existente cuando la etiqueta se encuentra en el ámbito de un lector. Generalmente las etiquetas de lectura-escritura poseen un número de serie que no puede ser sobrescrito. Puede emplearse bloques adicionales de datos para almacenar información adicional sobre los artículos vinculados con la etiqueta. Estos bloques de información pueden habitualmente cerrarse para impedir que la información sea sobrescrita. Los microchips de lectura únicamente tienen información que se almacena durante el proceso de fabricación. La información en

este tipo de chip nunca puede ser modificada. Las etiquetas WORM pueden tener un número de serie que se les escribe una sola vez, y esa información no puede ser sobrescrita.

Etiquetas pasiva y etiquetas activas.

Las etiquetas de RFID activas están provistas de un transmisor y una fuente de energía propia (generalmente una batería). La fuente de energía se utiliza para hacer funcionar los circuitos del microchip y para emitir la señal a un lector, funcionan como un teléfono que transmite señales a una estación base. Las etiquetas pasivas no incluyen una batería. Este tipo de etiqueta obtiene energía del lector, que emite ondas electromagnéticas que inducen una corriente en la antena de la etiqueta. Las etiquetas semi-pasivas emplean una batería para hacer funcionar los circuitos del microchip, pero se comunican obteniendo energía del lector. Las etiquetas activas y semi-pasivas son útiles para realizar el seguimiento o rastreo de objetos de alto valor que deben ser escaneados a largas distancias, como ser los vagones de ferrocarril sobre rieles. Las etiquetas activas o semi-pasivas cuestan más que las etiquetas pasivas, lo que significa que no se las puede usar en objetos de bajo costo. Actualmente se está trabajando para bajar el costo de estas tarjetas.

Colisión entre tarjetas.

La colisión de etiquetas ocurre cuando más de un transpondedor retransmite una señal al mismo tiempo, haciendo que el lector se confunda. Las normas estandarizadas de interfaz de protocolos de aire diferentes (y los diferentes sistemas patentados) emplean diferentes técnicas para que las etiquetas respondan al lector una a la vez. Puesto que cada etiqueta puede ser leída en milisegundos, da la

Receptor RFID CR95HF y el bus SPI.

El chip usado en el receptor RFID es el ***CR95HF*** fabricado por STM, para simplificar el desarrollo y considerando el bajo costo, hemos usado una placa ya ensamblada con este chip que se vincula al microcontrolador mediante el bus SPI.

El Bus **SPI** (*Serial Peripheral Interface*) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos, con este protocolo se puede controlar casi cualquier dispositivo electrónico que acepte un flujo de bits serie regulado por un reloj en una comunicación sincrónica.

El bus tiene un pin para el reloj, otro para el dato entrante, dato saliente y un pin para la selección de chip que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Un bus de periféricos serie es la opción más flexible cuando se tiene tipos diferentes de periféricos serie.

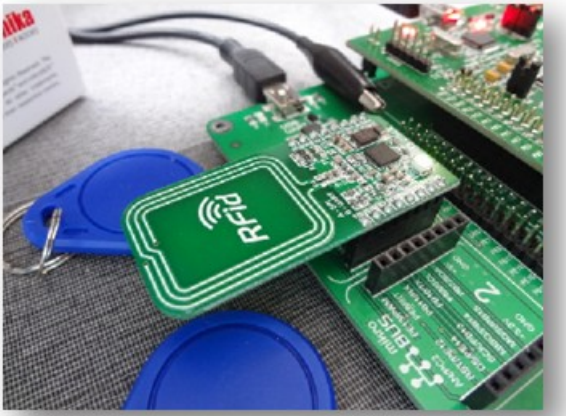
Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube otros cuando el reloj baja.

Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj. Algunos dispositivos tienen dos relojes. Uno para capturar o mostrar los datos y el otro para el dispositivo interno.

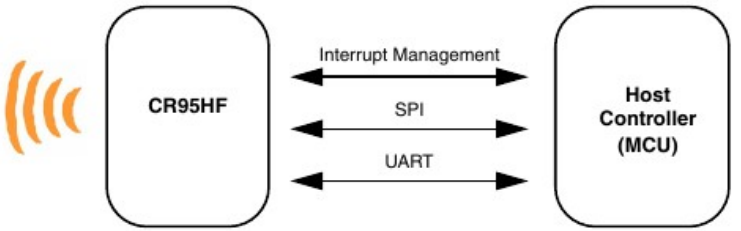
Los pines en el puerto se denominan de la siguiente manera.

- **SCLK** (*Clock*): Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit.
- **MOSI** (***M****aster* ***O****utput* ***S****lave* ***I****nput*): Salida de datos del Master y entrada de datos al Slave.
- **MISO** (***M****aster* ***I****nput* ***S****lave* ***O****utput*): Salida de datos del Slave y entrada al Master.
- **CS**: Para seleccionar un Slave, o para que el Master le diga al Slave que se active.

Para simplificar el armado hemos usado una placa ya ensamblada que distribuye Mikroelektronika a un costo muy bajo por lo que no justifica encarar su armado, sin embargo los diagramas para ensamblar esta placa están disponibles en el sitio de Mikroelektronika. La placa ya montada se puede ver en la siguiente imagen.

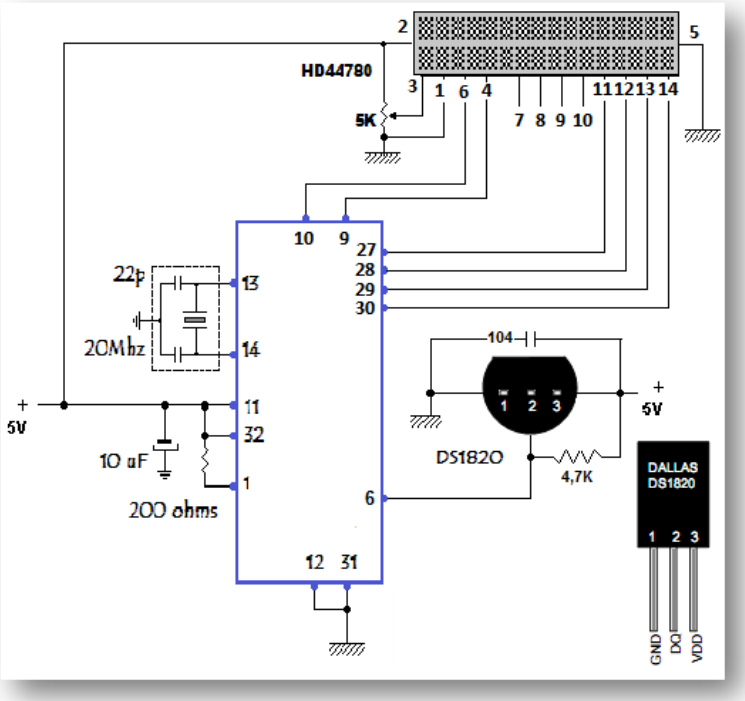


Existen muchos chips disponibles en el mercado para resolver las comunicaciones RFID, el ***CR95HF*** es solo uno de estos chips que que hemos elegido por su simpleza de conexión y gran eficiencia en su funcionamiento, esto lo convierte en una opción a considerar cuando estamos pensando en construir 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.



Flujo de datos entre la placa RFI y el microcontrolador PIC.

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.



Circuito propuesto para el controlador PIC18F4620

Resultados obtenidos al ejecutar el código de ejemplo para el sensor DS18B20.

En ocasiones puede ser necesario necesitar leer el código ROM de los sensores.
En el ejemplo siguiente vemos la forma de leer este código y mostrarlo en un LCD.
Este número de 64 bits puede ser útil para realizar una red de sensores e interrogarlos por su número ROM, podemos incluso mapear la red de sensores de acuerdo a su número ID y ante un fallo poder rápidamente determinar cual sensor no responde.
La lista de los ID en la red se podría guardar en una tabla y direccionarlos secuencialmente de acuerdo a los valores de esa tabla.
El siguiente es un ejemplo de como leer el ID de estos sensores.

```
/************************************************

**  Descripción  : Lectura del código ROM de
                   sensores DS18x20

**  Target       : 40PIN PIC18F4620

**  Compiler     : MikroC para PIC v 7.1

**  XTAL         : 20MHZ

**  NOTA         : Verificado con sensor DS18x20
                   conectado en el pin RA4.

( No olvidar el resistor de 4,7K en el pin 1-Wire)

************************************************/

// LCD module connections
```

```
sbit LCD_RS at LATE1_bit;
sbit LCD_EN at LATE2_bit;
sbit LCD_D4 at LATD4_bit;
sbit LCD_D5 at LATD5_bit;
sbit LCD_D6 at LATD6_bit;
sbit LCD_D7 at LATD7_bit;

sbit LCD_RS_Direction at TRISE1_bit;
sbit LCD_EN_Direction at TRISE2_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;
unsigned temp;
char sernum;
char sernum_hex[8];
unsigned char i;
unsigned char tmp;
```

REGISTER 4-1: RCON: RESET CONTROL REGISTER

R/W-0	R/W-1(1)	U-0	R/W-1	R-1	R-1	R/W-0(2)	R/W-0
IPEN	SBOREN	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX Compatibility mode)

bit 6 **SBOREN:** BOR Software Enable bit(1)
If BOREN1:BOREN0 = 01:
1 = BOR is enabled
0 = BOR is disabled
If BOREN1:BOREN0 = 00, 10 or 11:
Bit is disabled and read as '0'.

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** RESET Instruction Flag bit
1 = The RESET instruction was not executed (set by firmware only)
0 = The RESET instruction was executed causing a device Reset (must be set in software after a Brown-out Reset occurs)

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
1 = Set by power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{PD}$:** Power-Down Detection Flag bit
1 = Set by power-up or by the CLRWDT instruction
0 = Set by execution of the SLEEP instruction

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
1 = A Power-on Reset has not occurred (set by firmware only)
0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
1 = A Brown-out Reset has not occurred (set by firmware only)
0 = A Brown-out Reset occurred (must be set in software after a Brown-out Reset occurs)

Estructura del registro de control RCON.

El bit 3 del registro **RCON** es la bandera que avisa de un desborde del watchdog, el siguiente ejemplo para un PIC18F4620 permite ver claramente el funcionamiento del watchdog.

El pin 2 (RA0) es puesto normalmente a nivel alto mediante una resistencia de 10K, un botón colocado en el mismo lo conecta a nivel bajo cando se oprime.

PROTOCOLO UDP

Su función es proporcionar comunicación entre las aplicaciones de dos equipos. Emplea el protocolo IP para el transporte de los mensajes y, al igual que este, es no orientado a la conexión y no fiable. En el primer caso, debido a que no se establece una conexión previa entre los dos equipos para la transmisión de los mensajes, con lo cual existe la posibilidad de que estos no lleguen ordenados al destino.

En el segundo caso, porque los mensajes pueden llegar dañados o perderse. Tampoco existe confirmación de llegada por parte del receptor, con lo cual no hay manera de saber si alcanzaron

el destino correctamente.
Las aplicaciones que empleen este protocolo deben prever la forma de asegurar que la información que llegue es la correcta.

PROTOCOLO TCP.

Este protocolo, al igual que el UDP, emplea IP para el transporte de los mensajes, pero a diferencia de aquel, está orientado a la conexión y es fiable.
Dijimos anteriormente que el protocolo TCP es fiable. Quizá nos preguntemos cómo es posible esto, considerando que TCP viaja sobre el protocolo IP, que es no fiable. Pues bien, la forma de lograr esto es muy simple, y es que el receptor envíe un mensaje de confirmación al emisor cada vez que reciba un mensaje de él. En caso de que el emisor no reciba esta confirmación pasado un cierto tiempo, volverá a enviar el mismo mensaje.

Recordar que UDP es un protocolo no orientado a la conexión mientras que TCP si.

Conexión y desconexión en TCP.

El proceso para una conexión se realiza en tres pasos. En primer lugar, el emisor manda un mensaje SYN (configurado dentro de la cabecera TCP), y si el receptor está en condiciones de establecer la comunicación, le envía un mensaje SYN + ACK al emisor. Luego, el emisor manda un mensaje ACK al receptor. Recién en ese momento se establece la comunicación.
Por otra parte, el proceso para realizar una desconexión se efectúa en cuatro pasos, en donde cada uno deberá enviar un mensaje FIN y recibir el ACK correspondiente.

Conexión a servidores web.

Cada vez que ingresamos en una página web desde nuestro navegador, estamos accediendo a un servidor web, se trata de programas que se ejecutan en una computadora, o en nuestro caso un programa que se ejecuta dentro de nuestro microcontrolador y está permanentemente a la espera de que algún cliente realice una conexión. Estos clientes son los navegadores web que utilizamos.

El protocolo HTTP.

El protocolo *HTTP* está ubicado en la capa 7, capa de aplicación. Es un protocolo simple, basado en texto, utilizado por los navegadores

importante ya que tendrá que resolver las tareas propias de la aplicación más las tareas de la red, cuando trabajamos con micros de 8 bits y velocidades bajas (40Mhz) la sobrecarga es importante y notoria. (El sistema se pone muy lento).
Es por esto que este enfoque resulta interesante ya que parte de la carga en la gestión de la RED queda bajo la responsabilidad del ENC28J60.

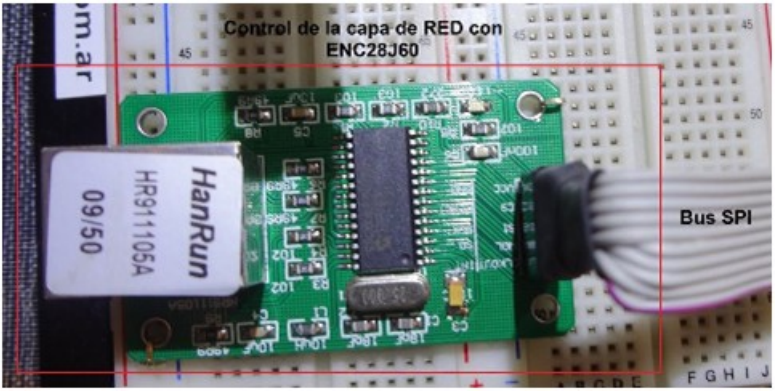
Nota del autor:

Personalmente creo que es demasiado pedir a un micro de 8 bits el que resuelva la gestión de red más un trabajo de control o gestión, siendo más adecuados para estas sobrecargas de procesos micros ya en 32 bits.

ENC28J60.

Este circuito integrado resuelve la capa física y de red de nuestra aplicación. De su hoja de datos sacamos algunas características relevantes.

- *IEEE 802.3 Ethernet Controller.*
- *10/100/1000 Base-T Networks.*
- *Integrated MAC and 10Base-T PHY.*
- *Supports One 10Base-T Port with Automatic.*
- *Supports Full and Half-Duplex modes.*
- *Programmable Automatic Retransmit on Collision.*
- *Programmable Padding and CRC Generation.*
- *Programmable Automatic Rejection of*



Placa usada en el ejemplo.

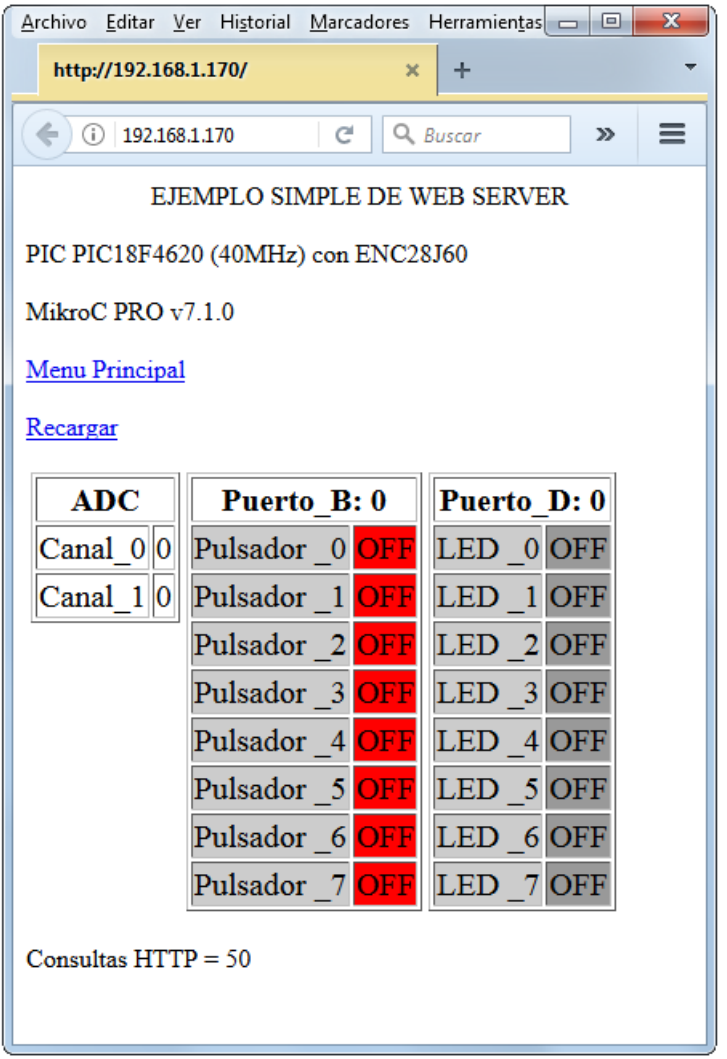
Estas placas son de uso común en el mundo Arduino, se vinculan mediante SPI a nuestro controlador obteniendo una comunicación estable a 10 Base T y se consiguen con facilidad en distintos portales de Internet

Ejemplo de una página web embebida.

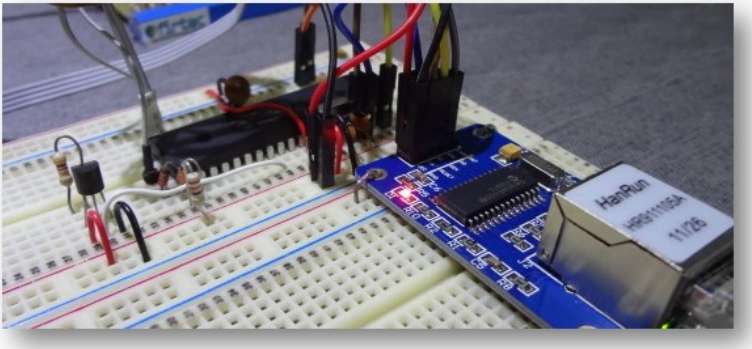
Habiendo programado diversos servidores web con las herramientas propias de Microchip debo decir que trabajar con Mikroc es infinitamente mas sencillo, sin embargo claro (*todo tiene un costo*) es justo remarcar que algunas funcionalidades con las herramientas de Microchip son de un alcance mucho mayor pero a costa de una complejidad considerablemente mayor.

Vamos a crear un pequeño servidor web capaz de controlador una serie de LED's colocados en el puerto D del microcontrolador PIC, también haremos visible el estado de ocho botones colocados en el Puerto B.

En la siguiente imagen podemos ver el aspecto de la página web embebida en la electrónica.



Algunas consideraciones importantes.



En nuestro canal de yotube (*firadmin*) puede ver un vídeo de este proyecto funcionando.

Impresoras Térmicas.

La impresora térmica se basa en una serie de agujas calientes que van recorriendo el papel sensible a la temperatura, que al contacto se vuelve de color negro.

Por su bajo costo, son muy usadas en los cajeros automáticos y supermercados.

La velocidad de impresión se mide en milímetros por segundo, refiriéndose a los milímetros del rollo de papel que salen de la impresora.

El papel tiene lado, si lo coloca de forma errada no imprime.

Con estas impresoras podemos por ejemplo imprimir tickets o reportes de algo que sea relevante conservar en papel. El uso de estas impresoras es muy sencillo, la que usamos como ejemplo de prueba es la que comercializa *Adafruit* muy comentadas en el mundo Arduino.

Básicamente son un receptor serial, y para hacerlas funcionar no tenemos más que enviar caracteres por un puerto serial.

Impresora CSN-A2-T usada en las pruebas.

Algunos puntos a tener presentes con estas impresoras.
La impresora solo recibe ASCII por lo que todo lo enviado deberá ser convertido a caracteres, los datos son enviados a 19200 baudios 8 bits sin paridad. Una vez que tenemos la impresora conectada podemos hacer un impresión de prueba, para esto conectamos la energía a la impresora y al mismo tiempo se oprime el único botón disponible en la impresora por un par de segundos. Esto imprime un patrón de prueba informando también el voltaje de alimentación y velocidad del puerto serial.
El modelo que usamos en el ejemplo es un CSN-A2-T y en la impresora está indicado que funciona de 5V a 9V.

Capítulo V

Historia de la Arquitectura ARM.

ARM es una empresa presente en el mercado desde 1990 y que ha logrado en pocos años colocar productos de consumo global diseñando una arquitectura que lidera a escala mundial en 32 bits.
Concebida originalmente por *Acorn Computers* para uso en computadoras, los primeros productos basados en ARM fueron los *Acorn Archimedes* lanzados en 1987.
La relativa simplicidad de los procesadores ARM los ha convertido en la tecnología dominante en el mercado de la electrónica móvil integrada, microprocesadores y microcontroladores pequeños, de bajo consumo y relativamente bajo costo. En la actualidad alrededor del 98% de los teléfonos móviles vendidos cada año utilizan al menos un procesador ARM.
Desde 2009, los procesadores ARM son aproximadamente el 90% de todos los procesadores RISC de 32 bits que se utilizan en la electrónica de consumo, incluyendo PDA, Tablets, Teléfonos inteligente,videoconsolas, calculadoras, reproductores digitales de música y medios (fotos, vídeos, etc.), y periféricos de computadoras como discos duros y routers.
La arquitectura ARM es licenciable. Las empresas que son titulares de licencias ARM actuales o anteriores incluyen a empresas como *Alcatel-Lucent, Apple Inc., AppliedMicro, Atmel, Broadcom, Cirrus Logic, Digital Equipment Corporation, Ember, Energy Micro, Freescale, Intel, LG,Marvell Technology Group, Microsemi, Microsoft, NEC, Nintendo, Nokia , Nuvoton, Nvidia, Sony, NXP (antes Philips), Oki, ON Semiconductor, Psion, Qualcomm, Samsung, Sharp, STMicroelec-*

básico del ARM. El diseño del ARM se ha convertido en uno de los más usados del mundo desde discos duros hasta juguetes. Hoy en día, cerca del 80% de los procesadores de 32 bits del mundo poseen este chip en su núcleo.Este curso se basa en el procesador **STM32-F407VG** de 32 bits fabricado por *STMicroelectronics* empresa que licencia los núcleos de ARM.

Que es Cortex M4.

Es un microcontrolador que pertenece a la arquitectura **ARM7** de última generación y de los más poderosos dentro de la gama de núcleos escalares con un set de instrucciones ortogonal que dan a este controlador gran flexibilidad y eficiencia para ejecutar código en C. Cortex M4 es una arquitectura RISC (*Reduced Instruction Set Computer*) con un conjunto de instrucciones de 32 bits. Las prestaciones de este controlador permite ya la ejecución de RTOS, Android, Linux, etc.

Características heredadas de RISC.

La arquitectura ARM incorporó algunas características del diseño RISC de Berkeley, aunque no todas. Las que se mantuvieron son:

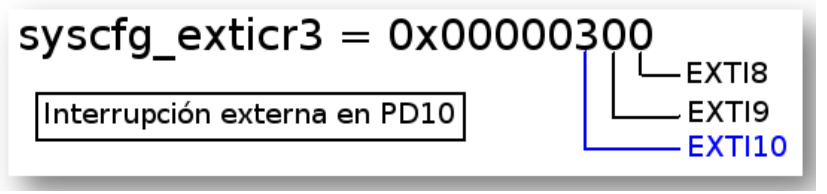
- *Arquitectura de carga y almacenamiento(load-store)*. Las instrucciones que acceden a memoria están separadas de las instrucciones que procesan los datos, ya que en este último caso los datos necesariamente están en registros.
- *Instrucciones de longitud fija de 32 bits*. Campos de instrucciones uniforme y de longitud fija para simplificar la decodificación de las instrucciones.
- *Formatos de instrucción de 3 direcciones*. Consta de “f” bits para el código de operación, “n” bits para especificar la dirección del 1er. operando, “n” bits para especificar la

bre los bits del registro de 32 bits **SYSCFG_EXTICRx**, en este registro vamos a seleccionar el pin, hay cuatro grupos posibles de selección.

- *EXTICR1 Pines de 0 a 3.*
- *EXTICR2 Pines de 4 a 7.*
- *EXTICR3 Pines de 8 a 11.*
- *EXTICR4 Pines de 12 a 15.*

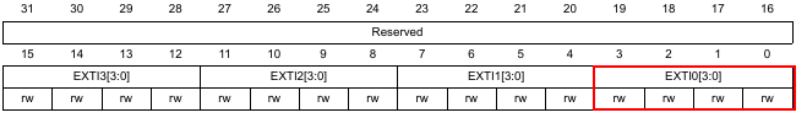
EXTICR1 está relacionado con el bloque EXTIO[3:0] que tiene cuatro bits para definir cual pin será activado. (SYSCFG_EXTICR1 = 0x00000000), para trabajar con PF0 la configuración sería SYSCFG_EXTICR1 = 0x00000101.
Imaginemos que vamos a usar el pin PD10 como entrada de interrupción externa, este pin está en la línea EXTI10 y el registro que permite acceder a esta línea es SYSCFG_EXTICR3 (configura pines de 8 a 11). Recuerde que los registros son siempre de 32 bits.

Observe la siguiente imagen que muestra como activar este bit, luego se define el flanco activo de la interrupción.



Observe en la siguiente imagen como es la organización de bits en el registro SYSCFG_EXTICR1 que es el registro que necesitamos ,para activar PA0, también tenemos el SYSCFG_EXTICR2, SYSCFG_EXTICR3 y SYSCFG_EXTICR4 dependiendo sobre cual pin vamos a trabajar.

Organización de bits en el registro SYSCFG_EXTICR1



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

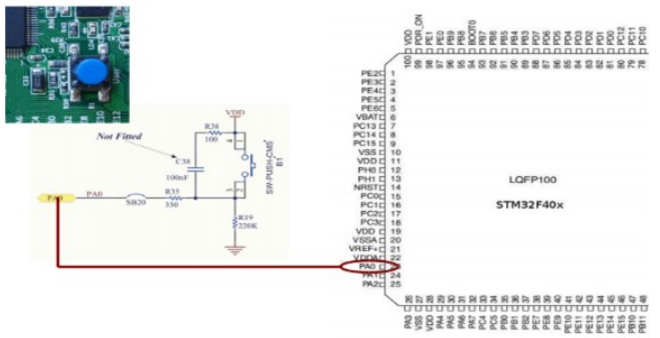
Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)
These bits are written by software to select the source input for the EXTIx external interrupt.
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1000: PI[x] pin

Las variables usadas en las interrupciones deben ser del tipo **volatile** para evitar que el compilador las incluya en los algoritmos propios de optimización. Hay compiladores que en su afán de optimizar las variables, y no pudiendo predecir que la variable se usará en una rutina que se accede en tiempo de ejecución (una interrupción) alteran esta variable.

Las variables usadas en una interrupción que no se declaran del tipo **volatil** pueden derivar en un comportamiento errático de la propia función de interrupción.

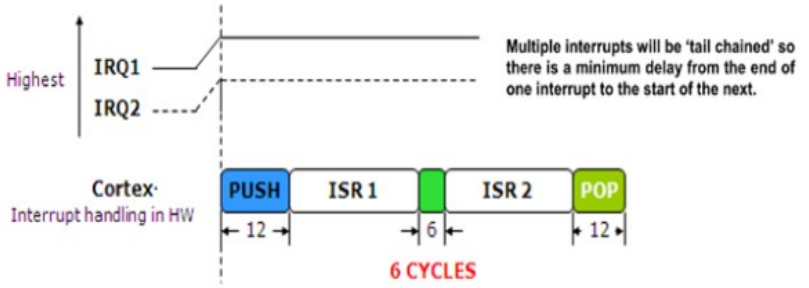
También debemos indicar si la interrupción será por flanco de subida o bajada. El registro EXTI_RTSR y EXTI_FTSR configuran el modo que la interrupción se comportará, la estructura de este registro se puede ver en la siguiente imagen.




```
/
**************************************************D
escripción  : Uso de una interrupción externa por
PA0.
Target       : STM32F407VG
ToolChain    : MiKroC para ARM
**************************************************/

// Consultar el manual de referencia de STM RM0090
// para consulta de los registros usados en el
// ejemplo.



 #define led    GPIOD_ODR.B15  // Pin salida

  void ExtInt() iv IVT_INT_EXTI0 ics ICS_AUTO {

   EXTI_PR.B0 = 1;
// Borra la bandera de interrupción

    led = ~ led;
// Cambia estado del led.

}
```

```
void main() {

  GPIO_Digital_Output(&GPIOD_BASE,
_GPIO_PINMASK_15); // PD15 como salida

  GPIO_Digital_Input(&GPIOA_BASE, _GPIO_PINMASK_0);
// PA0 como entrada

  SYSCFGEN_bit = 1; // RCC APB2 reloj de periféri-
cos activo

  SYSCFG_EXTICR1 = 0x00000000;
// Mapa de pines para PA0

  EXTI_FTSR = 0x00000001; // Flanco de bajado PA0

  EXTI_IMR |= 0x00000001;
// Set de la máscara para PA0

  NVIC_IntEnable(IVT_INT_EXTI0);
// Habilita interrupción EXTI0

  led = 1;   // Led inicia encendido
  while(1) { // Espera por la Interrupción }
}

//***** Fin de archivo - FIRTEC ARGENTINA *******
```

Una característica sobresaliente del eficiente trabajo que realiza el procesador Cortex con las interrupciones es la baja latencia entre el momento que la interrupción es detectada y el procesamiento de la misma.



En los procesadores Cortex prácticamente no existen retardos en atender interrupciones.

Recuerde: *Cuando se trabaja con interrupciones externas, primero se selecciona el grupo de pines y luego el pin específico.*

Temporizador del sistema (SysTick).

Es un temporizador presente en el núcleo del controlador, es independiente del fabricante e integrado al núcleo ARM está disponible en todos los fabricantes.

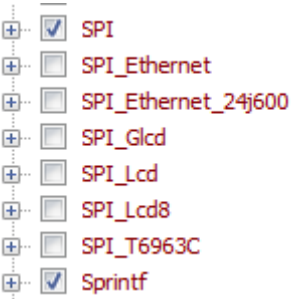
El funcionamiento es bastante simple, básicamente es un contador progresivo y conociendo la velocidad de CPU es simple calcular tiempos para los distintos estados de cuenta.

Importante:

El valor de cuenta para el SysTick no puede ser mayor a un valor de 24 bits 0xFFFFFF (Decimal 16777215).

En el siguiente ejemplo vamos a emplear dos variables en

```
  SPI2_Write(cmd);
  SPI2_Write(dataLen);
  while (dataLen == 0){
    CS = 1;
    break;
  }
  for(i=0; i<dataLen; i++){
    SPI2_Write(sdata[i]);
  }
  CS = 1;
}
/************************************************
* Esta función calibra el CR95HF con los valores
* indicados en la hoja de datos del chip.
************************************************/
void Calibrar_CR95HF() {
unsigned char x = 0;
do{
  sdata[0] = 0x03;
```



Programa ejemplo para el sintetizador de voz.

La librería requiere que informemos en que pines se encuentra conectada la placa TextToSpeech.
En nuestro caso hemos elegido la siguiente disposición de pines.

```
sbit TTS_RST at GPIOE_ODR.B4;

sbit TTS_CS at GPIOE_ODR.B5;

sbit TTS_MUTE at GPIOC_ODR.B5;

sbit TTS_RDY at GPIOD_IDR.B2;
```

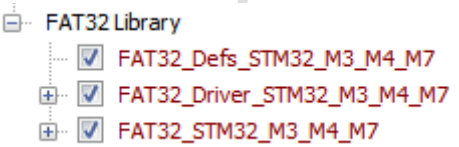
Una función se encarga de la configuración tanto de los pines como del puerto SPI3.

```
void hardware_init(){

    GPIO_Digital_Output( &GPIOE_ODR,
_GPIO_PINMASK_4 );

    GPIO_Digital_Output( &GPIOE_ODR,
_GPIO_PINMASK_5 );
```

DATO_0	PC8
DATO_1	PC9
DATO_2	PC10
DATO_3	PC11
CMD	PD2
CK	PC12

Conexionado

En el ejemplo para el uso de memorias SD usaremos la configuración de pines que se aprecia en la imagen anterior.
Para compilar el ejemplo se debe tener agregada al MikroC las correspondientes bibliotecas que publican las funciones necesarias para el manejo de la memoria SD.



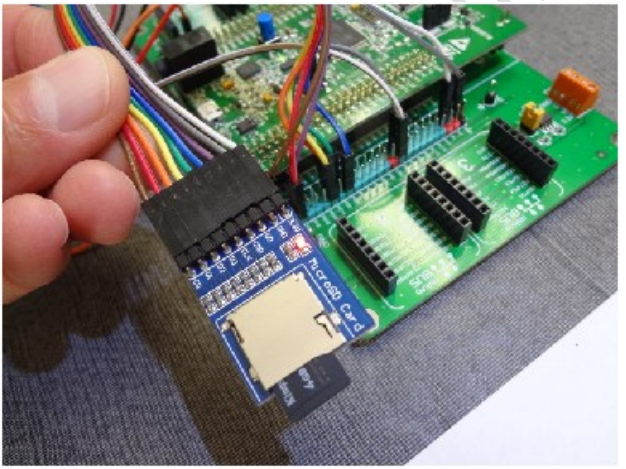
Librerías necesarias para el puerto SDIO.

El conexionado que hemos usado en la memoria es el siguiente.

El nombre del archivo en la línea.

fat32_file = FAT32_Open("Prueba.txt", FILE_APPEND)

La variable FAT32() es la función para crear el archivo, tiene dos argumentos, da nombre al archivo e indica en que forma se lo accede, se recomienda leer la ayuda de la biblioteca para conocer todas las posibilidades de la función. (*Crea el archivo si no existe, agrega datos al final de los existentes, etc*)



Memoria conectada al puerto SDIO.

enviar datos sueltos y esporádicos (mismo que hacemos con RS-232 y microcontroladores) este tipo de socket es ideal para mover datos con nuestra electrónica.

Wi-Fi con ESP8266.

ESP8266 es una plataforma abierta para la "*Internet de las Cosas*"que fue desarrollada en China.

Cuando apareció el módulo Wi-Fi ESP8266, el mundo de los hobbistas se entusiasmó bastante ante el surgimiento de un dispositivo capaz de entregar conectividad inalámbrica a plataformas de hardware a un bajo costo, entre los entusiastas se encontraba un grupo de desarrolladores chinos de plataformas abiertas de hardware, quienes se basaron en el ESP8266 para lanzar un kit llamado *NodeMCU* para el desarrollo de prototipos compatible con *Arduino*, que se programa en *Lua.* NodeMCU ha sido bastante bien recibido por la gente dedicada a desarrollar proyectos Hazlo-Tu-Mismo, pues permite el desarrollo de sistemas como por ejemplo para el monitoreo de temperaturas y humedad ambiental en habitaciones a un costo muy bajo.

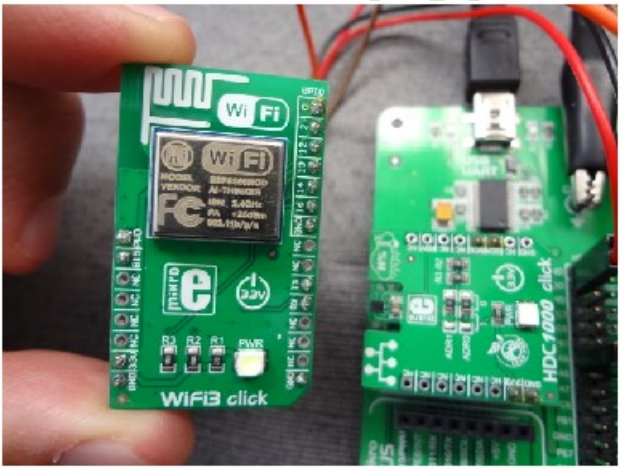
Sin embargo el uso de este firmware no solo sirve para Arduino, también podemos usarlo en otras arquitecturas como por ejemplo ARM.

Para los comandos AT la comunicación se realza mediante una conexión UART, y cualquier programa terminal sirve sin embargo el programa que usemos debe enviar un final de línea en cada transmisión.

Para actualizar el firmware se debe usar un programa especial que permite el acceso a la Flash o memoria de programa.

NOTA:
Este procedimiento puede inutilizar de manera permanente el modulo Wi-Fi si no se realiza correctamente.

Es importante conocer el origen del firmware que se está cargando en el módulo, hay muchas versiones modificadas en Internet y no todas funcionan correctamente, además será necesario cargar los parámetros de acceso a nuestra red Wi-Fi lo que en definitiva es abrir una puerta al mundo en nuestra propia red con los consabidos riesgos de seguridad.
En la imagen siguiente vemos la placa usada en el ejemplo propuesto.



Siempre conviene descargar las versiones oficiales desde el sitio de ***Espressif***, fabricante del chip.
Descargaremos los firmware denominados como "*based on ESP8266_NONOS_SDK_Vx.x.x*". La denominación *NONOS* significa

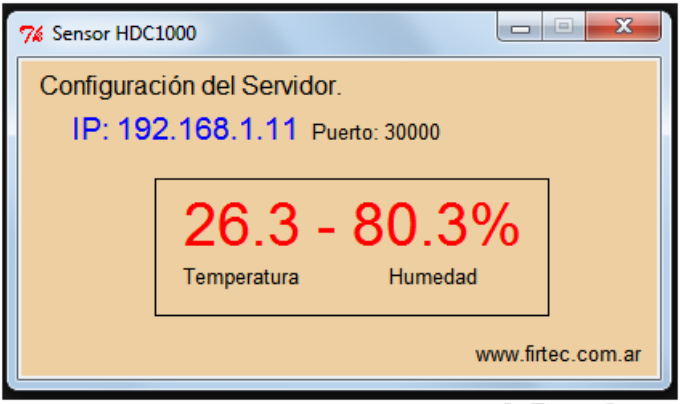
```
        Enviar_String("30000\r\n");

        Delay_ms(500);

        Enviar_String("AT+CIPMODE=1\r\n");
// Activa " TX transparente"

        Delay_ms(500);

        Enviar_String("AT+CIPSEND\r\n");
// Confirmo el modo

}
```

La función *Configurar_ESP8266()* es la encargada de preparar el enlace Wi-Fi para una conexión exitosa. Observe la forma de enviar los comandos AT, debe respetar este formato de lo contrario no funcionara. En rojo se han marcado los detalles de conexión que deben coincidir con los que tenemos en nuestro socket servidor.

Esto es válido dentro de nuestra red (Intranet), si buscamos conectarnos desde cualquier parte del mundo (Internet), entonces debemos hacer pública nuestra dirección IP y puerto, esto ya no es responsabilidad se los sockets sino de la configuración de nuestro router y red.

Midiendo Temperatura y Humedad por Wi-Fi.

Vamos a usar nuevamente el sensor HDC1000, 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.



Gran parte del código Python es igual al del trabajo anterior, sin embargo aquí se agrega la posibilidad de recibir y leer datos.

El código del servidor Python es el siguiente.

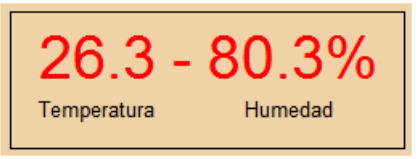
```
# -*- coding: utf-8 -*-

#!/usr/bin/env python

import socket

import sys

import errno

import time

from Tkinter import *

from tkMessageBox import showinfo

bandera = 0
```

```
while(1) {
  char caracter;
  Leer_Sensor();
  sprintf(floatStr,"%2.1f - %2.1f%% ",
          temper  ature,humidity);
  Enviar_String(floatStr);
  sprintf(floatStr,"Temp:%2.1f ",temperature);
  Lcd_Out(2,1,floatStr);
  sprintf(floatStr,"Hum:%2.1f%%",humidity);
  Lcd_Out(2,11,floatStr);
  Delay_ms(500);
  }
```

Las funciones marcadas en rojo son las encargadas de subir los datos a la red. El resto son funciones para el manejo de la pantalla LCD.

Observe la línea del *sprintf()* coloreada, está convirtiendo a caracteres los datos del sensor contenidos en *temperature* y *humidity* con un formato de dos enteros mas un punto mas un decimal y en el caso de la humedad se agrega el signo porcentual, pero también se agrega un guión medio separando los dos datos numéricos.

Básicamente se ensambla una cadena de caracteres que desplegada en la ventana Python de manera apropiada llevaría a pensar que los datos llegan en momentos distintos.



En este ejemplo por ser solo un trabajo didáctico no nos hemos tomado el tiempo de recorrer la cadena y seleccionar los datos de acuerdo a su posición en la ventana.
El siguiente es el código completo el socket cliente para el sensor HDC1000.

```
/*********************************************
Descripción  : Envío de datos con un modulo
WiFi ESP8266. Se conecta a la red enviando los
datos de temperatura y humedad de un sensor
I2C Texas HDC1000. Los datos también se mues-
tran en un LCD 20x4.

Target       : STM32F407VG
ToolChain    : MiKroC 5
www.firtec.com.ar
*********************************************/

 void Configurar_ESP8266(void);

// Pines asignados el LCD
```

OPT3001 en funcionamiento

En el siguiente ejemplo podemos ver como implementar el uso de este sensor.

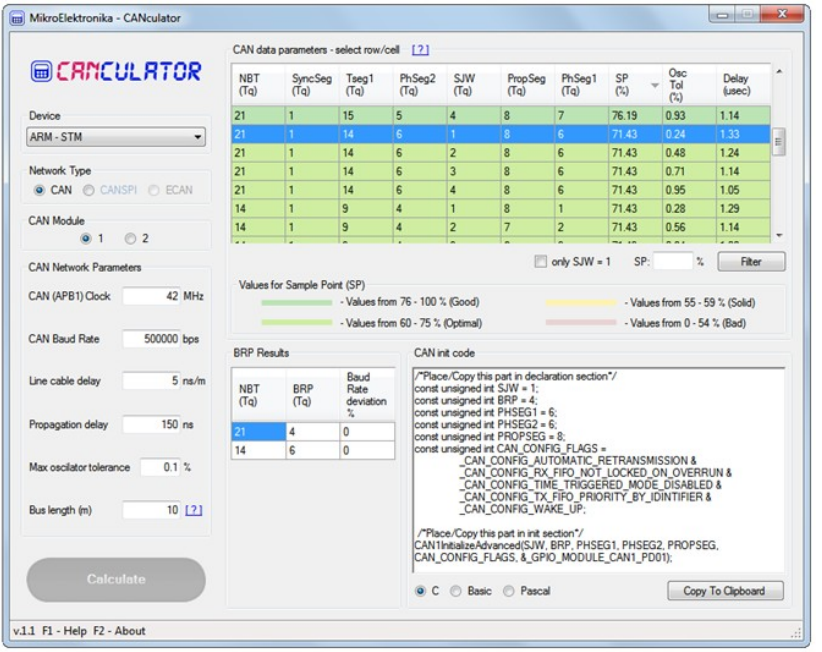
```
/************************************************
Descripción  : Medición de luz ambiente en el rango
de visión del ojo humano con el sensor OPT3001

Target       : STM32F407VG
ToolChain    : MiKroC para ARM
www.firtec.com.ar
************************************************/

    sbit LCD_RS at GPIOE_ODR.B4;

    sbit LCD_EN at GPIOE_ODR.B6;

    sbit LCD_D4 at GPIOC_ODR.B12;

    sbit LCD_D5 at GPIOC_ODR.B13;

    sbit LCD_D6 at GPIOC_ODR.B14;
```

```
                    strcpy(txt_viejo,floatStr);
// Actualiza nuevo valor

          }

   Delay_ms(500);

   }
}
```

El sensor se ha conectado en I2C1 pines PB7 y PB8.

CAN BUS (Controller Area Network).

CAN es un protocolo serie desarrollado por la empresa alemana BOSH originalmente para la industria automotriz, usa el método de transmisión broadcast, es decir, un elemento envía un mensaje a través del bus a todos los componentes, y estos se encargan de saber si la información del mensaje le es útil o no. Si el mensaje fuese de interés para algún nodo, este lo almacena y procesa, si no, simplemente la deshecha. Con la siguiente imagen trataré de aclarar un poco el mecanismo del proceso.



Herramienta CAN Calculator.

Para generar el código slo debemos informar la velocidad del bus donde está conectado el CAN y que velocidad en baudios vamos asignar a las comunicaciones, la velocidad depende de la distancia a cubrir. Esto se refleja en los valores de las variables escritas en las declaraciones globales.

En la actualidad el CAN es una forma de comunicación muy popular y muchos protocolos están basados en CAN por ejemplo *CANopen* para la industria, *CANaerospace* para la aviación, *SAE* para vehículos pesados o *NMA2000* para la industria marina entre otros.

El paso siguiente es ver el ZigBee en funcionamiento, para esto tenemos el siguiente ejemplo que envía los datos de temperatura y humedad de un sensor HDC1000 por ZigBee.

Conectados por ZigBee.

El siguiente ejemplo tiene un coordinador y un cliente, el coordinador es el encargado de mostrar los datos en una pantalla LCD 20x4, el resultado obtenido es el que se aprecia en la siguiente imagen.

El hardware usado en el proyecto es un shield para STM Dscovery con puertos MikroBUS que simplifican mucho el conexionado de una placa *BEE Clik* fabricada por Mikroelektronika.
En realidad lo que nos interesa de esta placa es el chip *MRF24J40MA* de microchip y su electrónica asociada, cualquier modulo que implemente esta electrónica es funcionará en este proyecto. En la siguiente imagen se puede apreciar el aspecto del montaje electrónico. Note que la placa que contiene el chip *MRF24J40MA* fabricado directamente por Microchip (*PCB de color rojo*) y que incluso tiene la antena para el enlace radial, está montado sobre la placa fabricada por Mikroelektronika (*PCB de color verde*) que provee el soporte para el MikroBus.

```
  data_TX_normal_FIFO[18] = DATA_TX[5];

  data_TX_normal_FIFO[19] = DATA_TX[6];

  data_TX_normal_FIFO[20] = DATA_TX[7];

  data_TX_normal_FIFO[21] = DATA_TX[8];

  data_TX_normal_FIFO[22] = DATA_TX[9];

  for(i = 0; i < (HEADER_LENGHT + DATA_LENGHT
+ 2); i++) {

    write_ZIGBEE_long(address_TX_normal_FIFO +
i, data_TX_normal_FIFO[i]); // Escribe el
frame en FIFO

  }

  set_not_ACK();

  set_not_encrypt();

  start_transmit();

}
```

Pantallas Táctiles y TFT.

Que es una pantalla *TFT* (*Transistor de Películas Finas*).

Es un tipo especial de transistor de efecto campo que se fabrica depositando finas películas de un semiconductor activo así como una capa de material dieléctrico y contactos metálicos sobre un sustrato de soporte. Un sustrato muy común es el cristal. Una de las principales aplicaciones de los TFT son las pantallas de cristal líquido. Esto

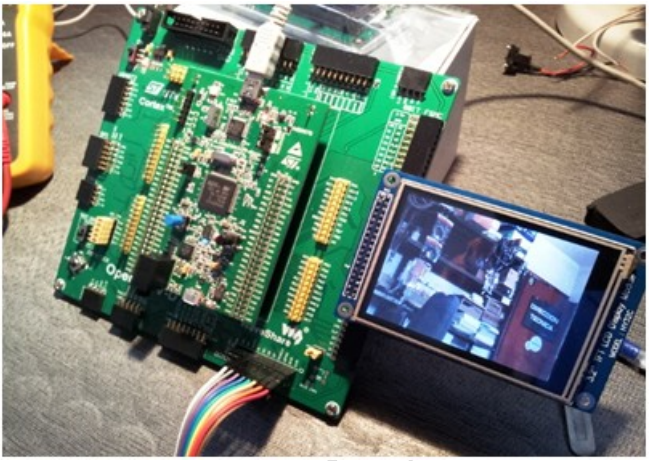
Pantalla TFT HY32C – HY32D.

Estas dos pantallas son parecidas salvo la doble línea de pines que tiene la HY32D y algunos detalles en su electrónica interna.

El controlador del LCD es el ***SSD1289*** y el del Touch Screen es el ***XPT2046*** que es compatible con el ***ADS7843***. La interfaz es paralela en 16 bits, siendo el control del Touch Screen por ***SPI***.
Esta pantalla tiene una resolución de 320*240 pixeles con 65536 colores. Su costo la convierte en una pantalla muy accesible y junto con varias del mismo tipo, son muy usadas en distintos proyectos y equipos electrónicos.
Trabajar con pantallas gráficas suele ser un poco engorroso, sobre todo a la hora del diseño de botones y elementos gráficos como los que estamos acostumbrados a ver en sistemas informáticos o teléfonos de nueva generación.

Si el trabajo se hace sin el soporte de un software específico para el diseño de interfaces gráficas, todo se debe hacer desde cero, indicar donde van las sombras, en que coordenadas, dibujar el botón de una forma cuando esta en reposo y dibujarlo de otra forma cuando

do la imagen captada con una cámara *OV7670* conectado al puerto de vídeo del Cortex M4.



Vídeo en pantalla con una cámara *OV7670* .

Introducción a Visual TFT.

Como se comentó anteriormente, el diseño de interfaces gráficas para electrónica es un trabajo que requiere tiempo y mucha dedicación.

Como seguramente el lector a escuchado mas de una vez, "*el tiempo es dinero*" y cuando debemos invertir gran cantidad de horas en un desarrollo muchas veces nos encontramos que el dinero recibido por este trabajo no está en relación con las horas invertidas.

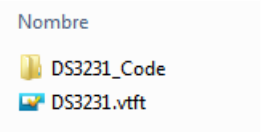
Visual TFT acorta enormemente el tiempo necesario para el desarrollo de una interfaz de usuario.
Tiene un entorno de trabajo muy similar a cualquier entorno de programación informático, una paleta de componentes ya pre-

Pantalla funcionando.

Trabajando con Visual TFT.

Teniendo instalado el MikroC descargamos e instalamos el Visual TFT.

Para ver su funcionamiento vamos a desarrollar *una interfaz* que nos permita ver de manera gráfica la hora de un reloj calendario DS3231. Una vez que el proyecto se ha definido, se crea un archivo *vtft*, este es el punto de entrada al proyecto en su parte gráfica.



Se creará de forma automática una carpeta *xx_Code* que contiene el código que vamos a compilar con MikroC.

Cuando se genera el código desde *Visal TFT* automáticamente se

```
void EveButton1OnClick() {

GPIOD_ODR.B14 = ~ GPIOD_ODR.B14;

}
```

En el evento *OnClick* de cada botón se genera la orden para el cambio de estado del LED. Dependiendo de cual botón se oprime será la acción sobre el LED.

De acuerdo a lo visto, como sería el código para el control de un enlace ZigBee que envíe información de un sensor LP-S25HB y la muestre en pantalla como se aprecia en la siguiente imagen.

Los ejemplos propuestos en este libro se pueden descargar desde el siguiente link: