

# **DISEÑO E IMPLEMENTACIÓN DE SISTEMAS EMBEBIDOS CON PIC®**

## **TOMO I INTRODUCCIÓN Y APLICACIONES BÁSICAS**

Andrés Bruno Saravia  
Fernando Tagliaferri  
Sebastián Gregori Fiadino  
Alejandro Anibal Airoldi

Airoldi, Alejandro Anibal

Diseño e implementación de sistemas embebidos con PIC: introducción y aplicaciones básicas . - 1a ed. - Buenos Aires: mcelectronics, 2013.

v. 1, 360 p. ; 23x18 cm

ISBN 978-987-26021-6-1

1. Informática. 2. Software. I. Título.

CDD 004

Fecha de catalogación: Octubre de 2013

© mcelectronics

Hecho el depósito que marca la ley 11.723

Todos los derechos reservados.

Ninguna parte de este libro, incluido el diseño de la portada, puede reproducirse, almacenarse o transmitirse de ninguna forma ni por ningún medio, sea este eléctrico, químico, mecánico, óptico, de grabación o de fotocopia, sin la previa autorización escrita por parte de mcelectronics. La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual.

La editorial no se pronuncia ni expresa implícitamente respecto a la exactitud de la información contenida en este libro, razón por la cual no puede asumir ningún tipo de responsabilidad en caso de error u omisión.

Las marcas mencionadas son propiedad exclusiva de sus registradores legales.

A nuestras familias y amigos por el apoyo incondicional.



# DISEÑO E IMPLEMENTACIÓN DE SISTEMAS EMBEBIDOS CON PIC®

## SERVICIO DE AYUDA AL LECTOR

Por favor no dude en escribirnos por consultas, sugerencias o para solicitar información adicional:  
[soporte@mcelectronics.com.ar](mailto:support@mcelectronics.com.ar)

## PROGRAMAS Y RECURSOS DE MICROCHIP

Para poder compilar los programas propuestos necesita el entorno de desarrollo MPLAB X, el compilador XC8 para PIC18, el compilador XC30 para los dsPIC y el XC32 para PIC32. Todas las herramientas se pueden obtener en forma gratuita desde la web de Microchip:

[www.microchip.com/mplabx](http://www.microchip.com/mplabx)

## CLASES ON-LINE EN VIVO

No dude en tomar una de nuestras clases on-line para complementar la información de este libro. Puede optar por clases grupales o bien una clase individual a través de Internet con un ingeniero de aplicación.

[www.mcelectronics.com.ar/online](http://www.mcelectronics.com.ar/online)

## COMPLEMENTOS

Microchip Application Libraries [www.microchip.com/mal](http://www.microchip.com/mal)

MCP2200 Configuration Utility [www.microchip.com/mcp2200](http://www.microchip.com/mcp2200)

LABVIEW [www.ni.com/labview](http://www.ni.com/labview)

MATLAB [www.mathworks.com/products/matlab](http://www.mathworks.com/products/matlab)

**En el DVD puede encontrar los compiladores y el código fuente de todos los programas. Recomendamos visitar periódicamente la web de Microchip para descargar las últimas versiones.**



# SUMARIO

<b>MPLAB® X GUIA VISUAL</b>	<b>11</b>
NUEVOS COMPILEDORES DE MICROCHIP	13
CREANDO UN PROYECTO EN MPLAB	14
<b>PROGRAMANDO EN LENGUAJE C</b>	<b>25</b>
CARACTERISTICAS DEL LENGUAJE	27
ARCHIVOS DE CABECERA	29
OPERADORES ARITMÉTICOS	31
DECLARACIÓN DE VARIABLES	33
FUNCION printf()	35
ESTRUCTURAS DE CONTROL Y COMPARACIÓN	37
OPERACIONES A NIVEL DE BITS (BITWISE)	46
CADENAS (STRINGS)	48
FUNCIONES	50
ÁMBITO DE LAS VARIABLES	52
<b>PRÁCTICAS XC8</b>	<b>59</b>
APRENDIENDO A UTILIZAR XC8 DE FORMA PRÁCTICA	61
PROGRAMA 1: LED-PULSADOR	63
PROGRAMA 2: PULSADOR TOUCH	65
PROGRAMA 3: CONTADOR 7 SEGMENTOS	67
PROGRAMA 4: MENSAJE EN UN DISPLAY LCD	76
PROGRAMA 5: CONTADOR CON DISPLAY LCD	80
PROGRAMA 6: CONTADOR CON TIMER 0	83
PROGRAMA 7: CONTADOR CON TIMER 1	87
PROGRAMA 8: RELOJ DE TIEMPO REAL CON TIMER 1	92
PROGRAMA 9: TERMÓMETRO CON PIC18F46K20	100
PROGRAMA 10: USANDO EL MÓDULO PWM	107
PROGRAMA 11: TRABAJANDO CON LA USART	112
CONVERSOR USART - USB MCP 2200	116

<b>ARDUINO CON PIC</b>	<b>121</b>
CARACTERÍSTICAS DE ARDUINO	123
PLATAFORMA CHIP KIT UNO32	124
EL BOOTLOADER	130
ARQUITECTURA DE PIC32	132
EL MPIDE	137
ESTRUCTURA DE UN PROGRAMA ARDUINO	140
EJERCICIOS CON ARDUINO	165
<b>CONTROL INTELIGENTE DE LEDS</b>	<b>185</b>
ILUMINACION LED	187
CONVERTIDOR NO SINCRÓNICO	198
LEDS DE ALTA INTENSIDAD	204
BOMBA DE CARGA	206
CONVERTIDOR BUCK	207
CONVERTIDOR BOOST	213
ILUMINACION INTELIGENTE	217
MICROCHIP MCP165X	220
LINTERNA A LED	224
<b>APLICACIONES GSM Y GPS</b>	<b>233</b>
ESQUEMA GENERAL	236
LAYOUT DE COMPONENTES	239
EL MICROCONTROLADOR	241
MÓDULO GSM Y GPS SIM908	244
CONEXIONES DEL MÓDULO GSM	245
CONSIDERACIONES IMPORTANTES	247
CÁLCULO DE LA LINEA DE TRANSMISIÓN	249
COMANDOS AT BÁSICOS	252
MÓDULO GPS	254
TRAMA NMEA DEL MÓDULO GPS	257
AMPLIFICADOR DE BAJO RUIDO	262
ACELERÓMETRO	265
ESQUEMÁTICO	267
EJEMPLOS Y APLICACIONES	271
REGULACIONES	283

<b>WI-FI HECHO SIMPLE</b>	<b>289</b>
ACERCA DE WI-FI	291
SEGURIDAD Y AUTENTICACION	293
MODULOS WI-FI DE MICROCHIP	295
ENLACE DE DATOS REMOTO	308
COMO SUBIR LOS DATOS	312
LEER DESDE UN SERVIDOR FTP	314
ESQUEMATICO	315
<b>COMUNICACIONES BLUETOOTH</b>	<b>319</b>
BLUETOOTH	321
DISPOSITIVOS DE MICROCHIP	325
CREANDO UN DISPOSITIVO SPP	329
CONSIDERACIONES DE HARDWARE	331
CONFIGURACION SOBRE UART	332
MODOS DE OPERACION	334
ESQUEMA DE CONEXIÓN	344
PERFIL HID	346
<b>BIBLIOGRAFÍA</b>	<b>349</b>



# **MPLAB®X**

## **GUIA VISUAL**



## NUEVOS COMPILADORES DE MICROCHIP

Recientemente Microchip presentó una nueva línea de compiladores para toda su gama de microcontroladores. El compilador XC8 es una versión mejorada HI-TECH por Microchip para que esté totalmente integrado a su plataforma del MPLAB, y tiene todos los recursos y formato que el resto de los compiladores de la familia de Microchip (MPLAB C18 y PICC).

El compilador se caracteriza por ser ANSI C, y si bien no es poderoso en cuanto a funciones embebidas, como el PCW de CCS; el XC8, le permite una programación totalmente transparente, ofreciendo al programador estándar de computadoras poder migrar fácilmente a programar microcontroladores PIC, o al programador Assembler, poder seguir trabajando como lo hacia pero de forma mucho más simple.

Denominación de los nuevos compiladores y las familias que abarcan:

- **XC8 para PIC10F,PIC12F,PIC16F,PIC16F1X y PIC18F**
- **XC16 para PIC24F,PIC24H,dsPIC30F,dsPIC33**
- **XC32 para PIC32**

Microchip ofrece sus compiladores en 3 modos:

- **Versión Standar:** dura 45 días y tiene una perfomance media, la relación de compresión es aceptable (paga licencia luego de los 45 días)
- **Versión PRO:** dura 45 días y tiene una altísima perfomance con la mayor relación de compresión de código (paga licencia luego de los 45 días)
- **Versión Lite:** duración ilimitada, sin restricciones de ningún tipo, sin embargo su relación de compresión es la mitad que la versión PRO, es decir que el mismo código compilado en la versión Lite ocupa, en la memoria de programa, el doble de lo que ocupa la versión PRO.

Otra característica importante es que al ser de **Microchip**, la actualización es constante, y la documentación es muy completa. El compilador de C **XC8** se caracteriza por su sencillez, es muy facil migrar del lenguaje Assembler al Lenguaje C. El programador puede sentir una gran comodidad al programar código encontrando la simplificación de muchas estructuras que antes debían realizarse por medio de extensas rutinas Assembler, y en ningún momento siente la falta de control sobre el hardware ya que puede acceder a cualquier BIT y registro SFR por medio de su etiqueta como figura en el DATA SHEET.

Son estas las razones que nos llevan a recomendar éste compilador y no otro, a la hora de querer migrar a un lenguaje de alto nivel.

## CREANDO UN PROYECTO EN MPLAB

El Lenguaje de Programación C, es un lenguaje que reúne todas las características que presentan los lenguajes de alto nivel y los de bajo nivel, uno puede tener control de su hardware a nivel BIT y al mismo tiempo también puede realizar cualquier tipo de operación matemática como si estuviera delante de una calculadora científica. Esto hace que hoy en día se lo considere un lenguaje de nivel intermedio. El compilador **XC8** de **Microchip** (que en adelante simplemente lo llamaremos **XC8**), utiliza todas las etiquetas originales de los DATA SHEET, para hacer referencia a los BITS y los SFRs de los MCU. De esta forma no debe uno estar recordando formas especiales para configurar los puertos I/O o para enviar datos a los mismos. Tampoco tiene implementadas funciones para controlar su hardware, lo cual también simplifica lo que debe recordar el programador. Simplemente, el programador debe tener presente las palabras claves del lenguaje C y conocer el funcionamiento del HARDWARE del MCU.

El compilador trae una extensa librería de funciones ANSI C, adaptadas a la realidad del MCU, que le permite realizar cualquier conversión de datos que requiera, o procesamiento de funciones matemáticas. Es como sentarse a programar delante de una computadora con el compilador Borland C.

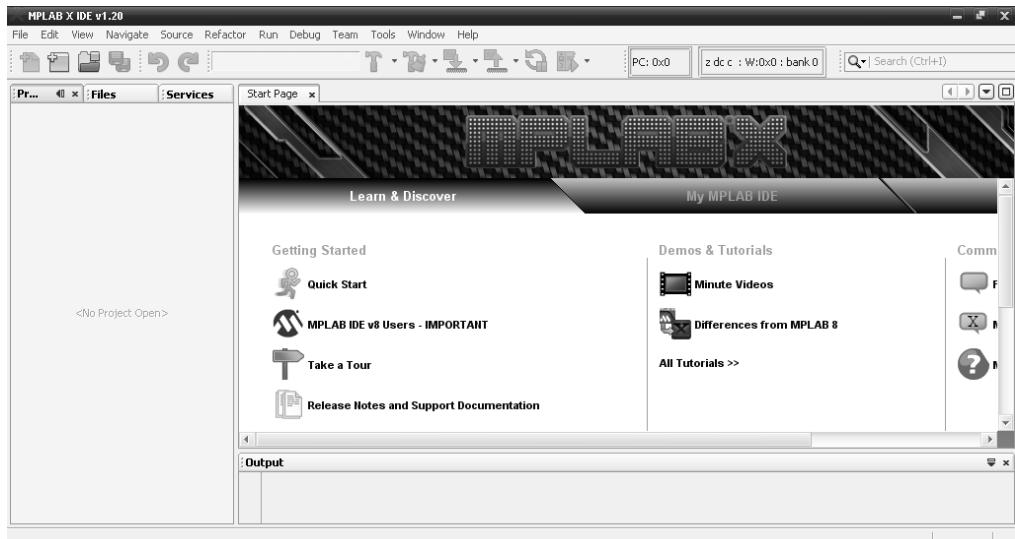
Por ello es tan sencillo para los programadores ANSI C utilizar el compilador **XC8**.

Vamos a comenzar por tanto a crear nuestro primer programa en Lenguaje C. Lo primero que haremos será ejecutar el MPLABX, ya que el compilador **XC8** está integrado al mismo.

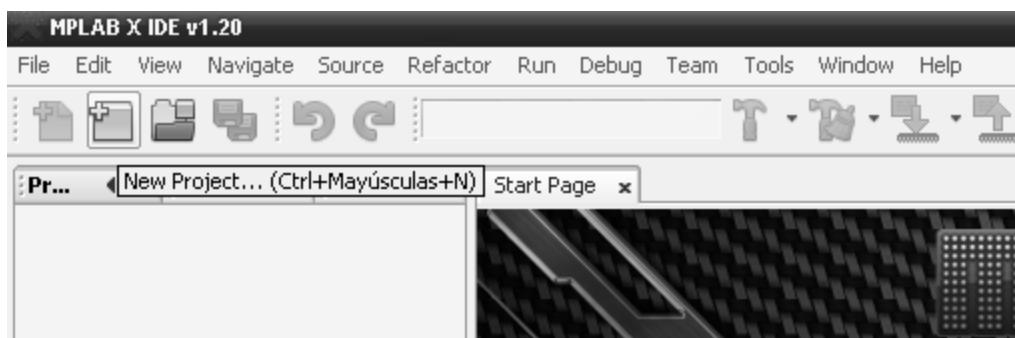
Lo primero que usted debe tener instalado es el MPLABX versión 1..20 en adelante.

Recomendamos esta versión ya que detecta automáticamente los compiladores instalados. Para instalar el MPLABX usted debe tener instalada la máquina virtual de java versión 6.24 en adelante, la cual se consigue en el sitio oficial de java (**si usted tiene Windows XP SP3 o WINDOWS 7, no necesita actualizar el java jre, ya que esta incluido en el paquete**).

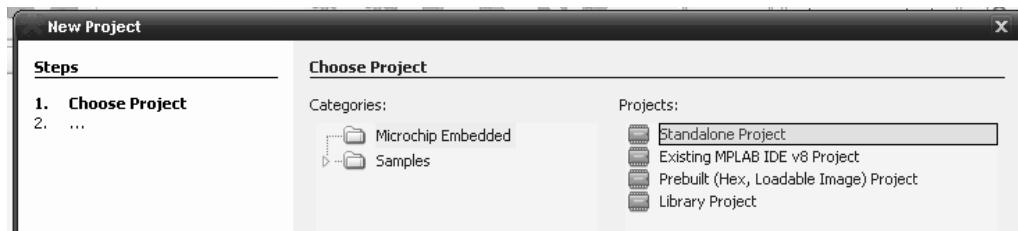
Una vez ejecutado el MPLABX aparecerá la siguiente pantalla:



Vamos a crear nuestro primer programa, para ello con el Mouse pulsaremos el botón de **New Project**:



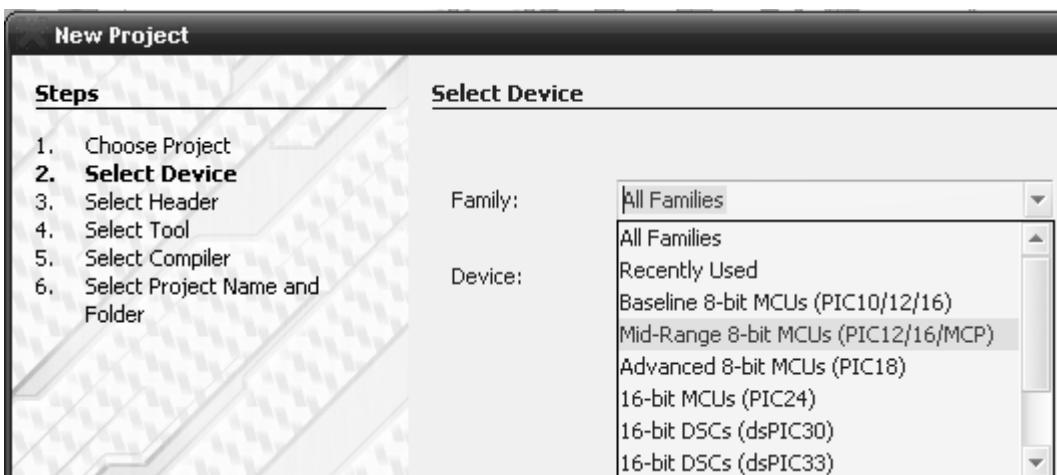
Se desplegará una nueva ventana mediante la cual seleccionamos el tipo de proyecto que queremos realizar:



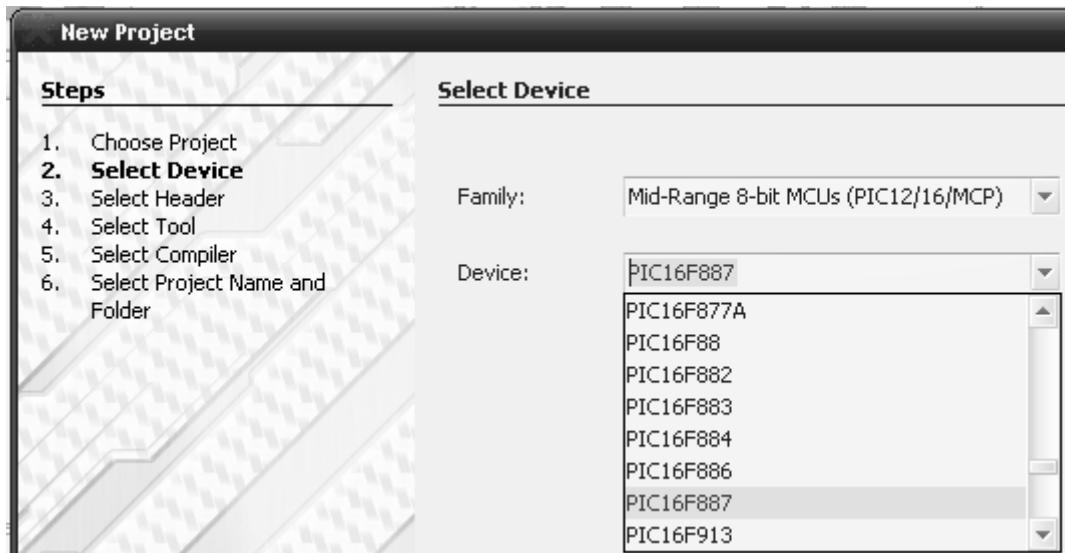
Que para nuestro caso es del tipo **Standalone Project** ("Proyecto independiente"). A continuación pulsamos el botón **NEXT** para pasar al siguiente paso, donde seleccionamos la familia de microcontrolador y su modelo:



Para hacerlo hacemos clic sobre el menú desplegable **Family** y seleccionamos el ítem : **Mid-Range 8-bit MCU (PIC12/16/MCP)** donde se concentran los microcontroladores de la familia PIC10F, PIC12F, PIC16F y PIC16F1; ya que nuestro primer proyecto lo realizaremos para PIC16F887:



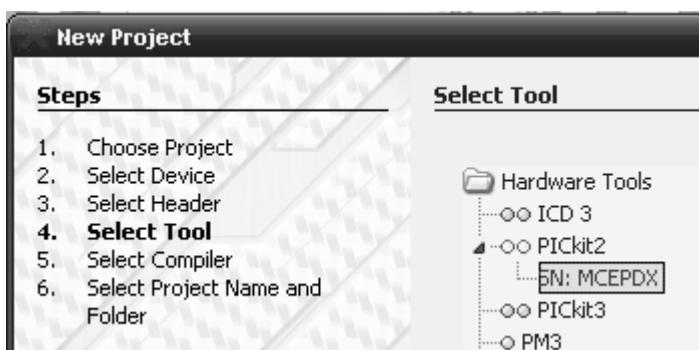
Seleccionamos luego el modelo de microcontrolador haciendo clic sobre el menú desplegable **Device** y buscamos, moviendo el cursor, el **PIC16F887**:



Luego pulsamos el botón siguiente para pasar al tercer paso. Este paso es opcional y depende del modelo de microcontrolador, ya que en algunos debe seleccionarse un Header especial (PIC Línea media Mejorada conocidos como PIC16F1XXX), y es por ello que en la mayoría de los microcontroladores, dicho paso es pasado por alto de forma automática y el wizard pasa al paso 4.

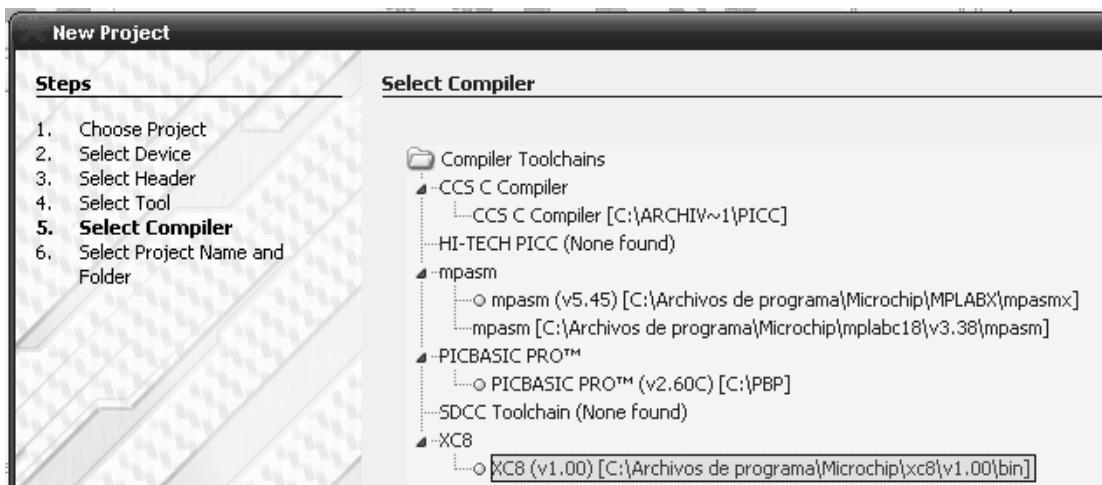
En este se selecciona la herramienta de programación, la cual debe estar conectada a la máquina antes de ingresar al mismo, para que el entorno la detecte automáticamente.

En nuestro caso usaremos el programador **PICKIT2**, ampliamente difundido en Argentina, tanto en su versión original, como en su versión local fabricada por **mcelectronics** bajo Licencia de Microchip con el nombre **PDX**. Para seleccionar el programador hacemos clic con el Mouse sobre el nombre que se detecta a continuación del título del mismo, una vez que este ha sido detectado por Windows:



Si observa podrá ver que los programadores reconocidos por el MPLABX son el PICKIT2 en adelante y han quedado fuera los viejos programadores como el PICSTAR PLUS y el ICD2. El PICKIT2 aparece con luces amarillas, lo cual significa que su soporte tanto en modo programador como debugger se encuentra en modo Beta.

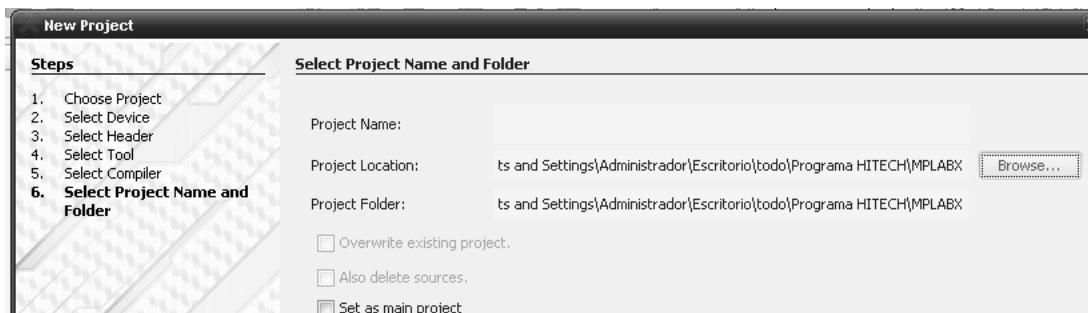
Hacemos entonces clic sobre el botón **NEXT** y pasamos al quinto paso donde seleccionaremos el compilador.



Observe que nos aparecen todos los compiladores instalados que pueden programar a estos microcontroladores. Dependiendo los que usted tenga instalados esta lista variará.

Nosotros seleccionamos con el Mouse el XC8. La versión MPLABX 1.20 automáticamente detecta los compiladores, sin embargo las versiones anteriores, debía ser instalado a mano, y el proceso era engoroso. Por ello le recomendamos que usted instale esta nueva versión de MPLABX.

En el siguiente paso crearemos una carpeta donde pondremos nuestro primer proyecto:



Como siempre es más cómodo trabajar sobre el escritorio, generaremos una carpeta general que contendrá todos nuestros proyectos realizados en MPLABX.

Para ello haremos clic sobre el Botón **Browse...** y se desplegará una ventana que nos permitirá ubicar el lugar donde colocar nuestra carpeta del proyecto y que llamaremos **ProyectosMPLABX**:



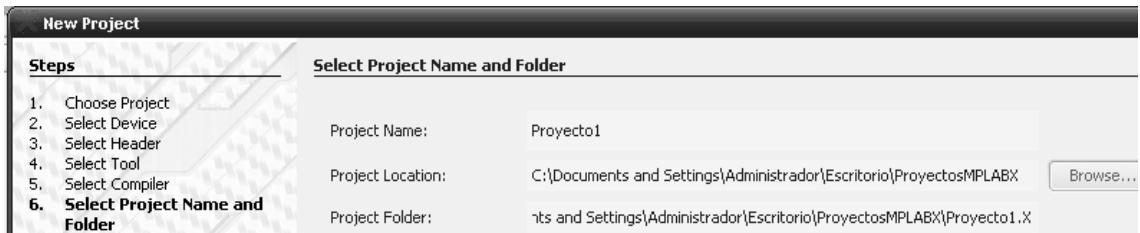
Seleccionamos el **Escritorio** y luego hacemos clic sobre el ícono de la creación de **Nueva Carpeta**:



Y le ponemos el nombre a la misma haciendo clic con el Mouse debajo de la carpeta creada, con lo cual debería quedarnos de la siguiente forma:

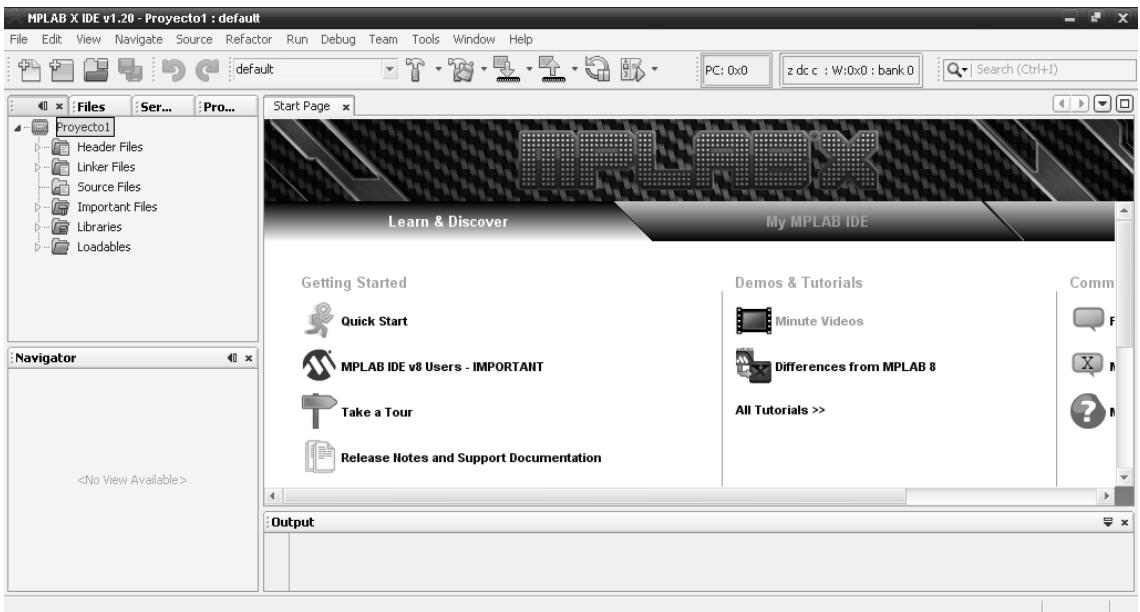


Luego hacemos clic sobre el botón **Abrir** y colocamos el nombre de nuestro primer proyecto sobre la ventana **Project Name**:



Observe que en la medida que uno escribe el nombre del proyecto, en la ventana **Project Folder**: el entorno crea una carpeta, dentro de la carpeta de proyectos, de forma automática y que lleva el nombre de nuestro proyecto con el agregado de **.X**, lo cual lo destaca como proyecto realizado en MPLABX.

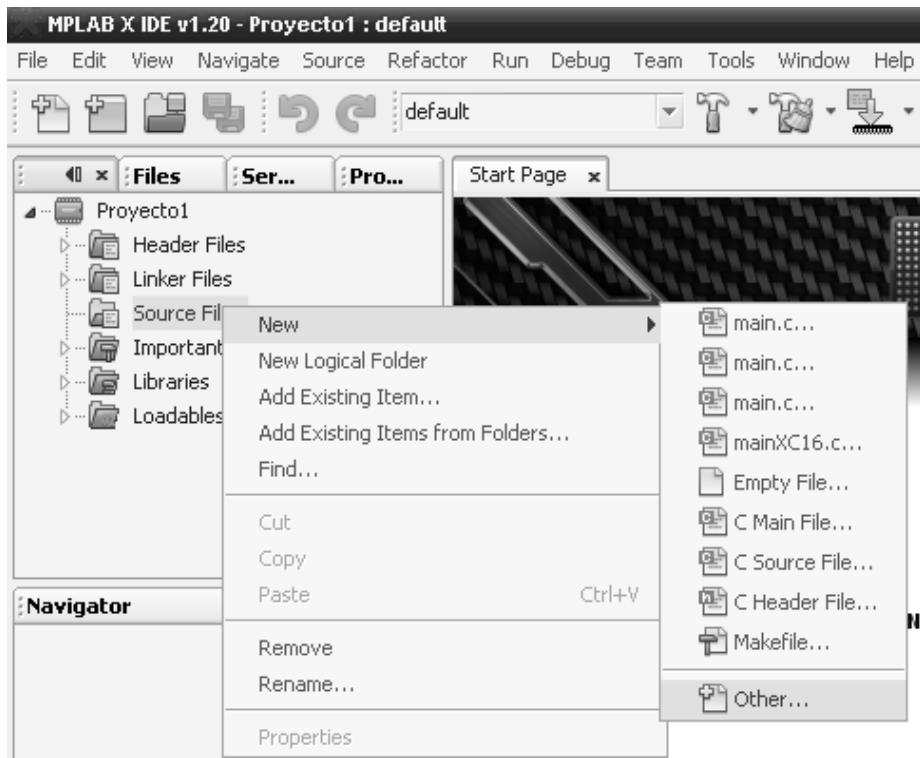
Para terminar, simplemente hacemos clic en el Botón **Finis**, lo cual cierra el wizard y despliega ya nuestro proyecto sobre la ventana principal:



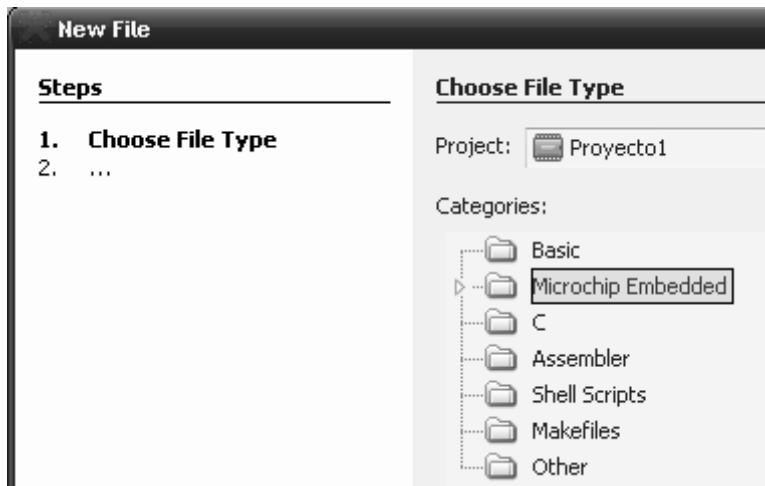
Nuestro Proyecto1 ya está creado y solo debemos crearle el código.

Observe el nombre del proyecto identificado con el chip, debajo del cual se organiza el proyecto en carpetas, como lo hacía el MPLAB viejo.

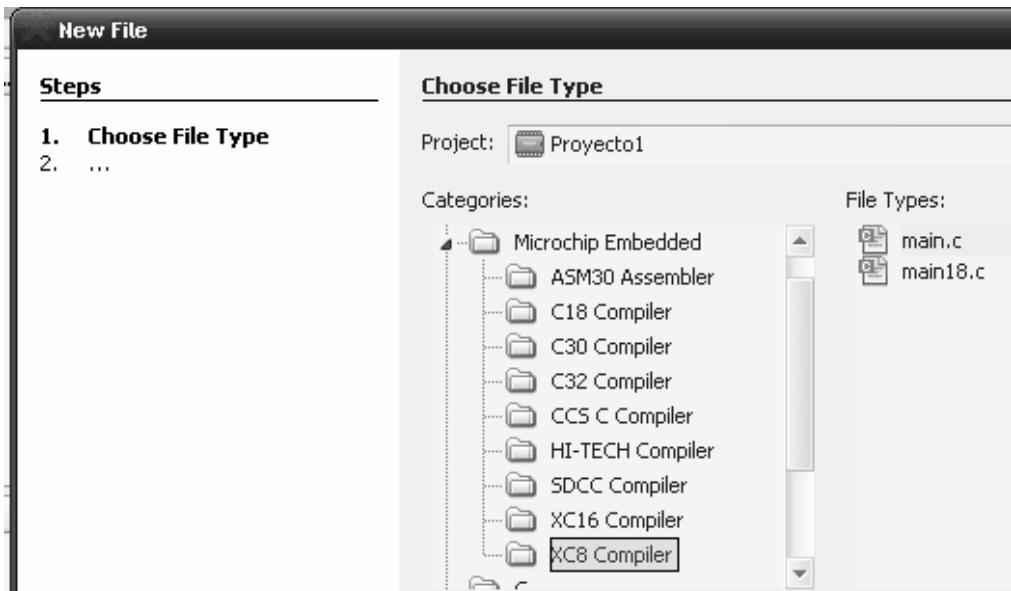
Agregaremos ahora nuestro código el cuál adjuntaremos a la carpeta **Source Files**. Para esto son muy útiles las plantillas que trae el entorno para los diferentes compiladores embebidos. Por tanto para seleccionar una de estas plantillas haremos clic con el botón derecho del Mouse sobre **Source Files** y se desplegará una ventana:



Y seleccionaremos el ítem **New** y luego dentro de este, seleccionamos **Other...** lo cual nos desplegará una nueva ventana, dentro de la cual seleccionaremos **Microchip Embedded**



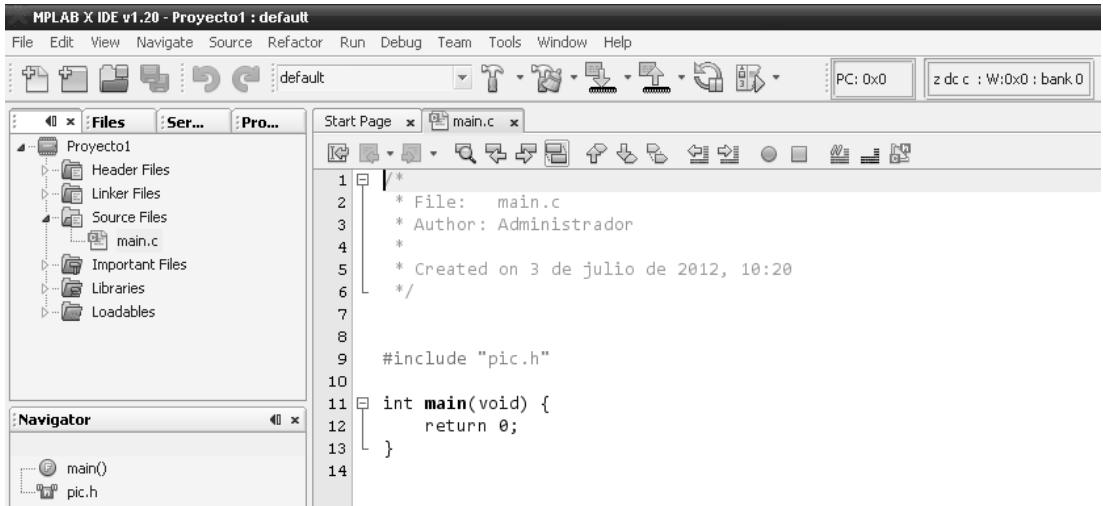
Luego haremos doble clic sobre el mismo lo cual nos desplegará el árbol de subdirectorios y seleccionaremos el del **XC8**, dentro del cual seleccionamos la plantilla **main.c**



Y hacemos clic en **Next**, lo que despliega una nueva ventana donde pondremos el nombre del archivo fuente:



Y finalmente pulsamos **Finish**, lo cual cierra la ventana y volvemos a la principal, donde nos aparece nuestro programa agregado a la carpeta de **Source Files** y desplegado en la ventana de trabajo:



Para que esta plantilla se adapte perfectamente a la normativa del compilador XC8 solo modificaremos la línea 9 donde aparece la siguiente “directiva”:

```
#include "pic.h"
```

Por la que corresponde:

```
#include <xc.h>
```

Y a continuación escribiremos nuestro primer código en XC8 nativo:

```

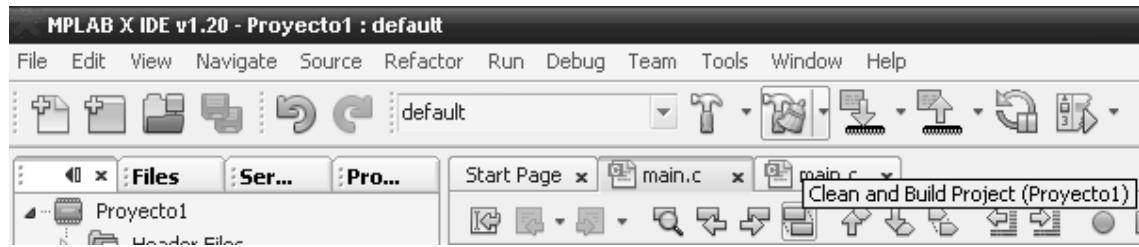
#include <xc.h>
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_INTRC_NOCLKOUT & WDTE_OFF & PWRTE_ON & LVP_OFF &
CPD_OFF & BOREN_ON);
int main(void) {

    ANSEL=0;
    ANSELH=0;
    PORTB=0;
    TRISB=0x00;
    while(1)
    {
        RBO=!RBO;
        __delay_ms(500);
    }

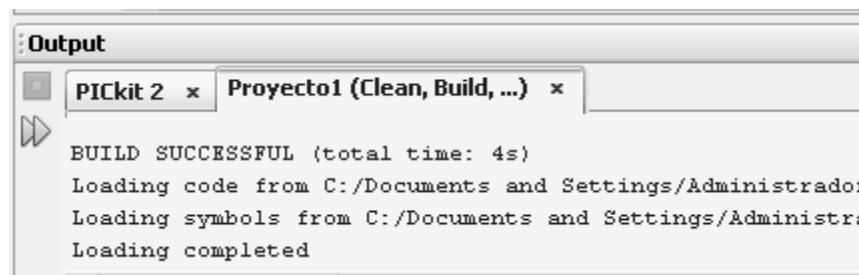
    return 0;
}

```

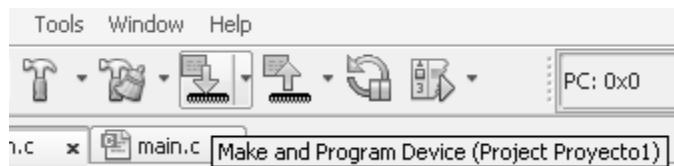
Este código simplemente provocará que un LED conectado al PIN RB0 titile de forma continua con un intervalo de 500 ms. Una vez escrito este programa usted deberá compilarlo pulsando el botón **Clean and Build Project** (identificado con el ícono de un martillo y una pala)



Y si no cometió errores al escribir el código, deberá aparecerle el mensaje de **BUILD SUCCESSFUL** en la ventana inferior del entorno y que se denomina **Output**:



Para pasar este código al microcontrolador simplemente haga clic con el Mouse sobre el botón **Make and Program Device**:



# **PROGRAMANDO EN LENGUAJE C**



## PROGRAMANDO EN LENGUAJE C

Es particularmente importante comprender los diferentes tipos de datos y las reglas que rigen la operación del Lenguaje C. La idea directriz de C es la definición de procedimientos (funciones), que en principio devuelven un valor. Lo que para nosotros es -conceptualmente- el programa principal, también es en C una función (la más externa).

El Lenguaje C está diseñado con vistas a la compatibilidad. En este sentido, todas las definiciones que puedan hacerse no serán concretas, pues son adaptables de acuerdo con la implementación. Un entero, por ejemplo, es una entidad con ciertas características generales, pero su implementación diferirá en distintos microcontroladores.

El lenguaje C maneja los datos en forma de variables y constantes. Las variables, simbolizadas mediante alfanuméricos (cuyas reglas de construcción veremos más adelante), presentan características que será muy importante considerar:

**Tipo de dato:** cada variable (también las constantes) está caracterizada por el tipo de dato que representa.

**Visibilidad:** en un programa C, cada variable tiene un rango de visibilidad (procedimientos en los que es reconocida), que depende de cómo se la haya declarado.

**Existencia:** relacionado con la anterior característica, es posible que el contenido de una variable perdure, o que se pierda, por ejemplo, al terminarse un procedimiento.

### Set de caracteres

C emplea dos sets (conjuntos) de caracteres:

- El primero de ellos incluye todos los caracteres que tienen algún significado para el compilador.
- El segundo incluye todos los caracteres representables.

C acepta sólo ciertos caracteres como significativos. Sin embargo, otros caracteres pueden formar parte de expresiones literales (constantes literales, nombres de archivo, etc.) que no serán analizadas por C.

## CARACTERES INTERPRETADOS POR C

Los caracteres a los que C asigna especial significado se pueden clasificar en alfanuméricos y signos especiales. Los caracteres alfanuméricos incluyen las letras (alfabeto inglés, de A a Z), mayúsculas y minúsculas, los dígitos, y el guión bajo (underscore: '\_').

En todos los casos, **las mayúsculas son consideradas distintas de las minúsculas**. Toda cadena alfanumérica con significación en C está compuesta exclusivamente por estos caracteres.

Los signos especiales son los listados en la siguiente figura. Ellos se emplean como delimitadores, operadores, o signos especiales.

Mayúsculas:	A - Z	Signo más	+
Minúsculas:	a - z	Signo menos	-
Dígitos:	0 - 9	Paréntesis izquierdo	(
Guion bajo:	_	Paréntesis derecho	)
Coma	,	Corchete izquierdo	[
Punto	.	Corchete derecho	]
Punto y coma	;	Llave izquierda	{
Dos puntos	:	Llave derecha	}
Signo de interrogación	?	Signo Mayor	>
Signo de admiración	!	Signo Menor	<
Comilla simple	'	Signo igual	=
Comilla doble	"	Asterisco	*
Barra vertical		Ampersand	&
Barra	/	Porcientos	%
Barra invertida	\	Caret	^
Tilde	~		

## **ARCHIVOS DE CABECERA**

Los archivos de cabecera son archivos cuya extensión es .h, (ejemplo stdio.h), y en principio uno incluye en su programa aquellos archivos necesario. Un archivo de cabecera contiene declaraciones de variables y constantes, prototipos de funciones, macros, etc.

El lenguaje C ofrece una cantidad de importante de estos archivos para que uno pueda escribir los programas y hacer uso de diversas funciones que permitan, por ejemplo, ingresar datos por la USART, utilizar funciones matemáticas, utilizar funciones para manipular cadenas, funciones para convertir datos y muchos etc.

El programa más sencillo de escribir en C necesita la inclusión de este archivo, ya que aquí se encuentran las funciones básicas de entrada/ salida, (en el futuro E/S), (stdio significa “Standar Input Output”):

**#include <stdio.h>**

También es esencial incluir un archivo fundamental conocido como archivo de cabecera del procesador:

**#include <xc.h>**

Este archivo se encarga de incluir el archivo de definiciones de etiquetas de los registros de funciones especiales (los que se encargan de manejar el hardware del microcontrolador) según el procesador que definimos al crear el proyecto.

En estos archivos no se encuentran implementadas las funciones, sino solo sus cabeceras, es decir se definen los “prototipos” de dichas funciones.

Los archivos de cabecera se incluyen de la siguiente forma:

**#include <stdio.h>**

Se utilizan los símbolos < > cuando el archivo de cabecera se encuentra en el directorio por defecto del lenguaje C instalado, el directorio por defecto para XC8 es:

**C:\Archivos de programa\Microchip\XC8\1.0\include**

Ahora si usted creo el archivo .h, (uno puede crear sus propios archivos de cabecera) lo incluye de la siguiente forma:

**#include “miarchivo.h”**

## Comentarios

En C tradicional los comentarios se colocan entre `/*` y `*/`, este generalmente es usado para comentar un bloque o varias líneas de código, pero también se puede usar para comentarios `//` que permiten comentarios en una sola línea.

```
/* Esto es un comentario  
Usado para comentar varias  
Lineas de código como en este  
Ejemplo */
```

`//` y esto también, pero se usa para una sola línea

## Puntos Y Comas / Llaves

Las sentencias ejecutables en C terminan en punto y coma ;  
Las llaves agrupan un conjunto de sentencias ejecutables en una misma función o en alguna estructura de iteración o comparación (ver más adelante).

## Los Identificadores

Los identificadores se utilizan para identificar, (valga la redundancia): variables, constantes, funciones, etc, básicamente se trata de los nombres  
Deben comenzar con una letra y tener una longitud máxima de 32 caracteres.  
Sólo pueden contener letras y números, pero no caracteres especiales, salvo el guión bajo, (underscore).

## Tipos de datos Básicos

El estándar Ansi define un conjunto de tipos básicos y su tamaño mínimo. En distintas implementaciones del lenguaje C estos tamaños puede cambiar, así por ejemplo: el **int** se define en Ansi C como un tipo de dato que almacena enteros en un rango de -32767 hasta +32767. En ciertas implementaciones de C para entornos de 32 bits el tipo **int** posee un rango de -2147483648 a 2147483647. En el caso de los microcontroladores la implementación de los datos depende del tipo de procesador; así para Hitech en MCU PIC 16F la implementación se define como indicamos a continuación:

## Tipos de datos enteros y decimales

Tipo	Tamaño (bits)	Tipo Aritmético
bit	1	Entero sin signo
signed char	8	Entero con signo
unsigned char	8	Entero sin signo
signed short	16	Entero con signo
unsigned short	16	Entero sin signo
signed int	16	Entero con signo
unsigned int	16	Entero sin signo
signed short long	24	Entero con signo
unsigned short long	24	Entero sin signo
signed long	32	Entero con signo
unsigned long	32	Entero sin signo
float	24 or 32	Decimal
double	24 or 32	Decimal
long double	igual al doble	Decimal

El compilador XC soporta tipos decimales, también conocidos como variables en punto flotante (pues la posición del punto decimal se puede correr) soporta 24 y 32 bits. El punto flotante se implementa usando el formato IEEE 754 que usa un formato de 32 bits o implementando un formato modificado en 24 bits.

## OPERADORES ARITMÉTICOS

Los operadores aritméticos son los siguientes:

OPERADOR	ACCIÓN
-	Resta
+	Suma
*	Multiplicación
/	división.
% (sólo para enteros)	Resto de la división entera
--	Decremento
++	Incremento.

Los operadores de decremento e incremento equivalen a:

$a=a + 1$  !  $a++$   
 $a=a - 1$  !  $a--$

En caso de presentarse a con el operador delante:

$a= 8;$   
 $b = ++a;$   
b toma el valor de 9.

Pero de plantearse lo siguiente:

$a = 8;$   
 $b = a++;$   
b toma el valor de 8.

O sea que en este último caso, primero ocurre la asignación y luego el incremento en a.

## OPERADORES RELACIONALES

Los operadores relacionales se usan normalmente en las estructuras comparativas usadas en los programas:

OPERADOR	ACCIÓN
>	Mayor que
$\geq$	Mayor igual que
<	Menor que
$\leq$	Menor igual que
$=$	Igual que
$\neq$	Distinto que

## OPERADORES LÓGICOS:

Los operadores lógicos se usan para relacionar estructuras comparativas creando comparaciones complejas. Los operadores usados por el lenguaje son los siguientes:

OPERADOR	ACCIÓN
$\&\&$	And
$\ $	Or
!	Not

En C, cualquier valor distinto de 0 es VERDADERO. FALSO es 0 (cero).

## **DECLARACIÓN DE VARIABLES:**

En C siempre se deben declarar las variables.

La declaración consiste en un tipo de dato, seguido por el nombre de la variable y el punto y coma:

```
int a;  
int b,c,d;  
int a = 10;
```

Los tres casos son definiciones correctas de variables, en el último además de declarar la variable se le asigna un valor inicial. En caso de existir una expresión con variables de diferentes tipos, el resultado obtenido es del tipo de operando de mayor precisión.

Todos los char se convierten a int.

Todos los float se convierten a double.

Hay que tener en cuenta que el tipo char es en realidad un int de menor precisión.

## **ARREGLOS (ARRAYS):**

Un array es una colección de elementos de un mismo tipo, que se referencian usando un nombre de variable común. En C, el array ocupa posiciones de memoria contiguas. La dirección más baja corresponde al primer elemento y la más alta al último. Para acceder a un elemento específico se usan índices.

### **Arrays unidimensionales:**

#### ***Forma general de declararlos:***

```
tipo nombre-variable[tamaño];
```

### **Ejemplo:**

```
int numeros[10];
```

Es un arreglo de 10 elemento enteros, donde el primero es numeros[0] y el último numeros[9].

Guardemos en este array los números dígitos, (0 al 9):

```
for(i=0; i<=9; i++)
    numeros[i] = i;
```

Cuando se declara un array, también se le puede asignar valores iniciales.

### **Ejemplos:**

```
int numeros[10] = {1,2,3,4,5,6,7,8,9,10};
char cadena[6] = "holo";
char cadena[5] = {'h', 'o', 'l', 'a', '\0'};
int matriz[4][2] = {1,1,2,4,3,9,4,16};
```

## **CONVERSIÓN DE TIPO DE DATOS (CAST):**

A veces es útil, o necesario, realizar conversiones explícitas para obligar que una expresión sea de un cierto tipo. La forma general de esta conversión en C es:

```
(tipo) expresión;
```

siendo **tipo**, el tipo de datos al que se convertirá la expresión.

**NOTA:** Esta conversión de tipos, (también llamada CAST), permite **convertir expresiones** no variables, esto es, si tengo una variable x de tipo int y le aplico **(float)x** lo que se convierte es el resultado, en caso que yo asigne esta operación a otra variable, pero **no se convierte la variable x a float**.

Supongamos que hacemos un programa que divide 10 por 3, uno sabe que el resultado será flotante: 3.333, y como 10 y 3 son enteros uno escribiría:

```
int a=10, b=3;
float r;
r=a/b;
printf("El resultado es %f", r);
```

pero se encontraría que el resultado no es el deseado, esto ocurre porque en C la división entre enteros da como resultado un entero y en la realidad no siempre es así, (sólo en el caso que b sea divisor de a). Pero cambiando el cálculo de la división por:

```
r=(float)a/b;
```

así se garantiza que el resultado será flotante.

## LA FUNCIONES PRINTF()

Es una función que están incluidas en el archivo de cabecera stdio.h y se utiliza para sacar información por la USART, y los datos son vistos por el software Hyperterminal de Windows. El número de parámetro pasados puede variar, dependiendo de la cantidad de variables a mostrar. El primer parámetro indica, por un lado, los caracteres que se enviarán por la USART y además los formatos que definen como se verán los argumentos, el resto de los parámetros son las variables enviar.

### Ejemplo:

```
int a=100, b=50;  
printf("%i es mayor que %i", a, b);
```

se enviará por la USART:

“100 es mayor que 50” (sin las comillas).

Los formatos más utilizados con printf() son:

CODIGO	FORMATO
%c	Un solo carácter
%d	Decimal (un entero)
%i	Un entero
%f	Punto decimal flotante
%e	Notación científica
%o	Octal
%x	Hexadecimal
%u	Entero sin signo
%s	Cadena de caracteres
%%	Imprime un signo %
%p	Dirección de un puntero

Los formatos pueden tener modificadores para especificar el ancho del campo, el número de lugares decimales y el indicador de alineación a la izquierda.

Ejemplos:

**%05d**, un entero de 5 dígitos de ancho; rellenará con ceros. Alineado a la derecha.

**%10.4f**, un real de 10 dígitos de ancho, con 4 decimales. Alineado a la derecha.

**%-10.2f**, un real de 10 dígitos de ancho, con 2 decimales. Alineado a la izquierda.

En la función printf() también se pueden encontrar “caracteres de escape” que permiten intercalar algún carácter especial en la cadena.

**Ejemplo:**

```
printf("\nHola mundo.\n");
```

Aquí antes de imprimir el texto “Hola mundo”, \n obliga a un salto de línea - retorno de carro, (ENTER) y . Y luego de imprimir el texto, hace otro salto de línea - retorno de carro.

#### CARACTERES DE ESCAPE:

CÓDIGO	DESCRIPCIÓN
\n	Salto de línea – retorno de carro (ENTER)
\t	Tabulado horizontal
\v	Tabulado vertical
\r	Retorno de carro.
\b	Backspace.
\f	Alimentación de página.
\'	Imprime una comilla simple.
\"	Imprime una comilla doble.
\\\	Imprime una barra invertida, (\).
\xnnn	Notación hexadecimal
\nnn	Notación octal

**Importante:** Antes de poder usar la sentencia **printf**, se tiene que haber configurado la USART cargando los registros de control que permiten su operación.

```

#include <xc.h>
#include <stdio.h>
#include "uart.h"

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & LVP_OFF & CP_OFF & CPD_OFF &
BOREN_ON);

void main(void)
{
    unsigned char input;
    init_comms(); //setea la USART con usart.h

    // Enviamos un mensaje

    while(1)
    {
        printf("\n Hola Mundo!!!\n");

    }
}

```

### **ESTRUCTURAS DE CONTROL Y COMPARACIÓN:**

Estas estructuras le permiten al programa tomar una decisión a partir de la evaluación de una **condición**, si dicha condición luego de ser evaluada se cumple, se ejecutará todas las sentencias que se encuentren debajo de la sentencia **if**, sino se cumplen se ejecutarán las que se encuentren debajo del **else**.

#### **Sentencia if (*forma general*):**

```

if (condición)
    sentencia1;
else
    sentencia2;

```

Si la condición es verdadera se ejecuta la sentencia1, de lo contrario la sentencia2.

### **Ejemplo:**

Este es un programa que enciende un led si se ha pulsado un pulsador. El Led se encuentra colocado en el puerto RB0 y el pulsador en el puerto RA0. El programa ha sido implementado para un PIC16F1939:

```
#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & MCLRE_ON & CP_OFF & CPD_OFF
& BOREN_ON);
__CONFIG(WRT_OFF & VCAPEN_OFF & PLLEN_OFF & STVREN_OFF & LVP_OFF);

void main(void)                                //función principal
{
    ANSELB=0;                                  //abrimos la función
    ANSELA=0;                                  //desactivamos los puertos analógicos
    TRISA=255;                                 //y los configuramos como digitales
    TRISB=0;                                  //configuramos todo el PORTA como entrada
    while(1)                                   //PORTB como salida
    {
        if (RA0==1)                            //bucle iterativo infinito
            LATB0=1;                          //abrimos el bucle
        else
            LATB0=0;                          //testeamos si se pulso RA0
        }                                     //encendemos el BIT 0 del PORTB
    }                                         //apagamos el BIT 0 del PORTB
                                            //cerramos bucle iterativo
                                            //cerramos la función principal
```

La condición en un **if** siempre debe ir entre paréntesis.

Si alguna de las ramas tiene más de una sentencia estas deben ir encerradas entre { }.

```

if (b)
{
    r = a/b;
    printf("\nEl resultado es: %f", r);
}
else
    printf("\nNo se puede dividir por cero");

```

No siempre se aplica la forma completa **if-else**, en ocasiones solo se usa la forma simple del **if** y si esta se cumple, se colocan debajo las operaciones que se desean realizar. En el siguiente segmento de código vemos esta forma:

```

//contador binario para PIC16F877A
#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & CP_OFF & CPD_OFF & BOREN_ON);

//declaramos nuestras variables
unsigned char contador=0;                                //Creamos la variable contador

//Programa principal
void main(void)                                         //Función principal
{
    ADCON1=6;                                            //Abrimos la función
    TRISB=0;                                             //desactivamos puertos analógicos
    TRISA=255;                                           //PORTB como salida
    while(1)                                              //PORTA como entrada
    {
        if(RA0==1)                                         //bucle iterativo infinito
        {
            contador=contador+1;                          //abrimos el bucle
            PORTB=contador;                            //se pulso el pulsador???
            __delay_ms(2);                           //desactivamos puertos analógicos
            while(RA0==1);                           //PORTB como salida
            __delay_ms(2);                           //PORTA como entrada
        }
        PORTB=contador;                                //mostramos la cuenta
    }                                                       //cerramos bucle iterativo
}                                                       //cerramos la función principal

```

También se pueden escalar las sentencias **if** para crear estructuras de toma de decisión más complejas:

```
if (condición)
    Sentencia1;
else if (condición)
    Sentencia2;
else if (condición)
    Sentencia3;
else
    Sentencia4
```

Y anidar como lo mostramos en el siguiente ejemplo de código:

```
if (TMR1IF==1)
{
    Segundo++;
    if (Segundo==60)
    {
        Segundo=0;
        Minuto++;
        if(Minuto==60)
        {
            Minuto=0;
            Hora++;
            if(Hora==24)
                Hora=0;
        }
    }
    TMR1IF=0;
    TMR1L=0b00000000;
    TMR1H=0b10000000;
}
```

## **SENTENCIA SWITCH:**

La sentencia switch permite evaluar diferentes valores para una misma variable:  
Su forma general es:

```
switch(variable)
{
    case valor1:
        sentencias;
        break;
    case valor2:
        sentencias;
        break;
    case valor3:
        sentencias;
        break;
    .
    .
    .
    default:
        sentencias;
}
```

El switch evalua cada caso, cuando coincide uno de ellos con el contenido de la variable, ejecuta las sentencias del caso y termina el switch. En caso de no encontrar ningún case que corresponda, en igualdad, con el contenido de la variable, ejecuta las sentencias de la cláusula default, si esta ha sido especificada, sino termina el switch.

El siguiente programa pide el ingreso de un valor y luego ofrece un menú de opciones para elegir que operación se desea realizar: averiguar el cuadrado del número, el cubo y si es par o no. El programa ha sido realizado para PIC16F877A.

```

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "uart.h"
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & CP_OFF & CPD_OFF & BOREN_ON);

void main(void)
{
int opcion, valor, res;
char buf[80];
init_comms();           //setea la USART con usart.h
printf("Introduzca un valor entero mayor que 0:\n");
printf("\n\n");
gets(buf);
valor= atoi(buf);
printf("*** MENU DE OPCIONES ***\n\n");
printf("1 - Averiguar el cuadrado:\n");
printf("2 - Averiguar el cubo:\n");
printf("3 - Averiguar si es par o no:\n");
printf("\nIngrese opción: ");
printf("\n\n");
gets(buf);
opcion= atoi(buf);
switch(opcion)
{
    case 1:
        re = valor*valor;
        printf("El cuadrado de %i es %i\n", valor, res);
        break;
    case 2:
        re = valor*valor*valor;
        printf("El cubo de %i es %i\n", valor, res);
        break;
    case 3:
        res = valor % 2;
        if (res)
            printf("El número %i es impar\n", valor);
        else
            printf("El número %i es par\n", valor);
        break;
    default:
        printf("Opción erronea");
}
}

```

## BUCLES ITERATIVOS:

Son estructuras que permiten ejecutar una cantidad de instrucciones un número de veces o mientas se cumpla una condición determinada. Existen 3 formas para crear estos bucles:

### El bucle for

Esta instrucción nos permite crear un bucle iterativo el cual se repetirá una cantidad de veces, la cual esta determinada dentro de ciertas condiciones

*Forma general:*

```
for(inicialización; condición; incremento)
    sentencia;
```

```
for(inicialización; condición; incremento)
{
    sentencias;
}
```

El siguiente ejemplo muestra los primeros 100 números enteros:

```
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "uart.h"
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & CP_OFF & CPD_OFF & BOREN_ON);

void main(void)
{
int i;
init_comms(); //setea la USART con usart.h
while(1)
{
    for(i=1; i<=100; i++)
        printf("%d", i);
}
```

También puede funcionar al revés, (los primeros 100 enteros mostrados de 100 a 1);

```
for(i=100; i>=1; i--)
printf("%d", i);
```

Se pueden evaluar más de una variable:

```
int i, j;
for(i=1, j=100; i<=100, j>0; i++, j - )
printf("i = %d, j= %d\n", i, j);
```

El siguiente bucle no termina nunca, (bucle infinito):

```
for( ; ; )
```

Si la cantidad de sentencias, (equivalencias, operaciones aritméticas, llamadas a funciones, etc.), pertenecientes al bucle son más de una, estas deben ir entre { }.

### **El bucle while**

El bucle while es otra forma de bucle, la sentencia se ejecutará mientras la condición se cumpla. Si la condición se cumple, entonces se ejecuta la sentencia, caso contrario, no se ejecutará

#### **while (condición)**

```
    sentencia;
```

si son varias las sentencias que deben ejecutarse, estas deben colocarse entre llaves usando la siguien

#### **while (condición)**

```
{
    sentencias;
}
```

**Ejemplo:** En el siguiente ejemplo aplicamos el while para crear un bucle iterativo permanente (while(1)) y luego para crear un enclavamiento para el pulsador.

Mientras no se suelte, el sistema quedará en esa instrucción (while(RA0==1)). El código crea un pulsador con retención, invirtiendo (toggleando) el estado del PORT RBO, el cual tiene conectado un LED. El código ha sido realizado para un PIC16F1939.

```
#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & MCLRE_ON & CP_OFF & CPD_OFF
& BOREN_ON);
__CONFIG(WRT_OFF & VCAPEN_OFF & PLLEN_OFF & STVREN_OFF & LVP_OFF);
void main(void) //función principal
{
    //abrimos la función
    ANSELB=0;      //todo el PORTB como digital
    ANSELA=0;      //todo el PORTA como digital
    TRISA=255; //todo el PORTA como entrada
    TRISB=0;      //todo el PORTB como salida
    LATB=0;        //inicializamos el LATB completo
    while(1) //bucle iterativo infinito
    {
        //abrimos el bucle
        if (RA0==1) //esta pulsado el pulsador???
        {
            //si esta pulsado;
            LATB0=!LATB0; //toggleamos RBO
            __delay_ms(2); //anti-rebote de entrada
            while(RA0==1); //soltó el pulsador???
            __delay_ms(2); //antirebote de salida
        }
    } //cerramos bucle iterativo
} //cerramos la función principal
```

### El bucle do – while

```
do
{
    sentencia;
}
while (condición);
```

La diferencia con el while es que en do – while por lo menos el flujo del programa entra una vez al bucle y luego, (al llegar a la cláusula while), decide si continúa iterando.

## **OPERACIONES A NIVEL DE BITS, (BITWISE):**

El lenguaje C proporciona la posibilidad de manipular los bits de un byte, realizando operaciones lógicas y de desplazamiento.

### **Operadores a nivel de bits:**

OPERADOR	ACCIÓN
&	And entre bits
	Or entre bits
^	Xor entre bits, (or exclusivo).
~	Not , (si es 1 pasa a ser 0 y viceversa)
<<	Desplazamiento a izquierda
>>	Desplazamiento a derecha
var << n	Se desplaza n bits a la izquierda.
var >> n	Se desplaza n bits a la derecha.

Los Operadores lógicos trabajan de la siguiente manera:

```
1 & 1 = 1  
1 | 0 = 1  
0 | 1 = 1  
1 | 1 = 1  
1 ^ 0 = 1  
0 ^ 1 = 1  
~1 = 0  
~0 = 1
```

De forma general conviene tener siempre presente estos resultados

```
X & 1 = X  
X & 0 = 0  
X | 1 = 1  
X | 0 = X  
X ^ 1 = ~X  
X ^ 0 = X
```

**Ejemplo:** En el siguiente ejemplo usamos el operador de desplazamiento para desplazar un BIT por el PORTB. Este desplazamiento puede ser observado si tenemos 8 LEDS conectados al PORTB. El programa esta realizado para un PIC16F887A:

```

#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

unsigned char contador=0;

void main(void)
{
    ADCON1=6;      //todos los puertos como digitales
    TRISB=0b00000000;//todo el PORTB como salida
    TRISA=0b11111111;//todo el PORTA como entrada
    PORTA=0;        //inicializamos el PORTA
    PORTB=0;        //inicializamos el PORTB

    while(1)//Bucle repetitivo eterno
    {
        RB0=1; // abrimos el bucle
        __delay_ms(500); //esperamos 500ms
        for(i=0,i<8,i++)//abrimos un bucle para rotar 8 veces
        {
            PORTB=PORTB<<1; //desplazamos el PORTB una vez
            __delay_ms(500);// esperamos 500ms
        }
    }
}      //cerramos la función principal

```

## CADENAS (STRINGS):

Se conoce como “cadena de caracteres o strings” al conjunto de caracteres ASCII que forman un número, una palabra, o un conjunto de ellas.

En C no existe un tipo de datos específico para declarar cadenas, en su lugar la idea de cadena surge de un array de caracteres que siempre termina con el carácter nulo, (\0) y cuya posición debe contemplarse al dimensionar el array.

Para guardar una cadena de 10 caracteres:

```
char cadena[11];
```

Cuando se introduce una constante de cadena, (encerrada entre dobles comillas), no es necesario terminar con el carácter nulo, ya que el C lo crea automáticamente. Lo mismo sucede cuando se introduce una cadena desde el teclado, utilizando la función gets(), incluida en stdio.h, que genera el carácter nulo con el retorno decarro, (enter).

Ejemplo: Programa que muestra una cadena introducida desde el teclado:

```
#include<xc.h>
#include<stdio.h>
#include <stdlib.h>
#include "uart.h"

#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

char cadena[100];

void main(void)
{
    init_comms();//setea la USART con usart.h

    While(1)
    {
        printf("Ingrese cadena, de hasta 100 caracteres\n");
        gets(cadena);
        printf("Usted ingresó: %s", cadena);
    }
}
```

## FUNCIONES DE CADENAS

Estas funciones estan incluidas en las librerías fundamentales del compilador y son usadas para el tratamiento de los strings. Para poder usarlas, se debe incluir el archivo de cabecera **string.h**, el cual se encuentra dentro de la carpeta **include**. Se toma por convención que los strings terminan con el carácter nulo para estas funciones.

### **strcpy():**

Se utiliza para copiar sobre una cadena:

```
strcpy(cadena, "Hola");
```

Guarda la constante “Hola” en la variable cadena.

**ATENCIÓN:** En C no se puede realizar entre cadenas la asignación cadena = “Hola” ya que recordemos que son arrays de caracteres.

### **strlen():**

Devuelve la cantidad de caracteres que posee la cadena, sin contar el carácter nulo.

```
a = strlen(cadena);
```

### **strcat():**

Concatena dos cadenas.

```
strcat(cadena1, cadena2);
```

Resultado: cadena1 es la suma de cadena1 + cadena2

### **strcmp():**

Compara los contenidos de dos cadenas.

```
strcmp(string1, cstring2);
```

Si son iguales devuelve 0.

Si cadena1 es mayor que cadena2: devuelve un valor mayor a 0.

Si cadena1 es menor que cadena2: devuelve un valor menor a 0.

## FUNCIONES:

Las funciones son porciones de código que facilitan la claridad de desarrollo del programa.  
Todas las funciones retornan un valor y pueden recibir parámetros.

La estructura general de un función en C es la siguiente:

```
Tipo_de_retorno nombre_función (tipo param1,...,)  
{  
    sentencias  
    return(valor_de_retorno);  
}
```

Los posibles tipos de retorno son los tipos de datos ya vistos: (int, float, void, char,etc).  
Para crear una función en C, primero hay que declarar el prototipo de la misma antes de la función main() y luego de la llave final del programa se define la función.

### Ejemplo:

Programa con una función que recibe 2 parámetros enteros y retorna la suma de los mismos:

```
#include<xc.h>  
#include<stdio.h>  
#include <stdlib.h>  
#include "uart.h"  
  
#define _XTAL_FREQ 4000000  
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );  
  
int suma(int x, int y); //prototipo de la función  
  
void main(void)  
{  
    int a, b;  
    unsigned char buf[80];  
    init_comms(); //setea la USART con usart.h  
    while(1)  
    {  
        printf("Ingrese valor de a: ");  
        buf=gets();  
        a= atoi(buf);  
        printf("\nIngrese valor de b: ");  
        buf=gets();  
        b= atoi(buf);
```

```

        printf("\nLa suma de a y b es: %i", suma(a,b));
    }
}

//Ahora viene la definición de la función
int suma(int x, int y)
{
    return x+y;
}

```

Se retorna de una función cuando se llega a la sentencia `return` o cuando se encuentra la llave de cierre de la función.

Cuando lo que se desea escribir es un procedimiento que, por ejemplo, realice un dibujo o muestre un texto por pantalla o cargue una arreglo, o sea, que no devuelva ningún valor se escribe como tipo de retorno `void`, (tipo vacío). El siguiente programa consta de una función que se encarga de cargar un arreglo de caracteres:

```

#include<xc.h>
#include<stdio.h>
#include <stdlib.h>
#include "uart.h"

#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

void carga(void);
char v[10];
void main(void)
{
    int i;
    init_comms(); //setea la USART con usart.h
    carga(); //llamo a la función que carga el arreglo
    for(i=0; i<10; i++)
        printf("%c, ", v[i]);
}

//definición de la función
void carga(void)
{
    int i;
    for(i=0; i<10; i++)
        v[i] = getc();
}

```

En este caso la función se comporta como un procedimiento, por eso carece de la sentencia return, que estaría de más pues el retorno es void.

## **ÁMBITO DE LAS VARIABLES:**

### **Variable global:**

Conocida por todas las funciones. Se puede utilizar en cualquier punto del programa. Se declara fuera del main.

### **Variable local:**

Se declara apenas abrir una llave en el código, cuando la llave se cierra esta variable desaparece.

### **Variable declarada en los parámetros formales de una función:**

Tiene el mismo comportamiento de las variables locales.

## **PASO DE PARÁMETROS:**

### **Paso por valor:**

Cuando se pasa un parámetro por valor a una función, (ver ejemplo de la función que suma), la función hace copias de las variables y utiliza las copias para hacer las operaciones. **No se alteran los valores originales**, ya que cualquier cambio ocurre sobre las copias que desaparecen al terminar la función.

### **Paso por referencia:**

Cuando el objetivo de la función es **modificar el contenido de la variable pasada como parámetro**, debe conocer la dirección de memoria de la misma. Necesita que se le anteponga a la variable el operador &, puesto que se le está pasando la dirección de memoria de la variable, ya que el objetivo es guardar allí un valor ingresado por teclado.

El siguiente programa tiene una función que intercambia los valores de dos variables de tipo char.

```

#include<xc.h>
#include<stdio.h>
#include <stdlib.h>
#include "uart.h"

#define _XTAL_FREQ 4000000
__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

void cambia(char* x, char* y); //prototipo

void main(void)
{
    char a, b;
    init_comms();//setea la USART con usart.h
    a='@';
    b='#';
    printf("\n**** Antes de la función ****\n");
    printf("Contenido de a = %c\n", a);
    printf("Contenido de b = %c\n", b);
    cambia(&a,&b); (*)
    printf("\n**** Después de la función ****\n");
    printf("Contenido de a = %c\n", a);
    printf("Contenido de b = %c\n", b);
    getc();
}

void cambia(char* x, char*y) (**)
{
    char aux;
    aux=*x;
    *x=*y;
    *y=aux;
}

```

En la línea (\*) se llama a la función cambia() pasándole las direcciones de memoria de las variables, puesto que precisamente el objetivo es modificar los contenidos.

La función en (\*\*), **recibe los parámetros como punteros**, puesto que son los únicos capaces de entender direcciones de memoria como tales. Dentro de la función tenemos entonces x e y que son punteros que apuntan a la variable a y a la variable b; utilizando luego el operador \* sobre los punteros hacemos el intercambio.

**ATENCIÓN:** Los arrays, (entiéndase también cadenas), siempre se pasan por referencia y no hace falta anteponerle el símbolo &, pues como habíamos dicho el nombre de un array es un puntero al primer elemento del mismo.

## **APENDICE:**

En este apéndice encontrará el texto de los archivos de librería de las funciones para el manejo de la USART.

### **Archivo USART.h: Librería para USART.C**

```
#ifndef _SERIAL_H_
#define _SERIAL_H_

#define BAUD 9600
#define FOSC 4000000L
#define NINE 0 /*poner 1 si usa 9 BITS */

#define DIVIDER ((int)(FOSC/(16UL * BAUD) -1))
#define HIGH_SPEED 1

#if NINE == 1
#define NINE_BITS 0x40
#else
#define NINE_BITS 0
#endif

#if HIGH_SPEED == 1
#define SPEED 0x4
#else
#define SPEED 0
#endif
```

```

//definición de PINES de la USART según el MCU

#if defined(_16F87) || defined(_16F88)
    #define RX_PIN TRISB2
    #define TX_PIN TRISB5
#else
    #define RX_PIN TRISC7
    #define TX_PIN TRISC6
#endif

/* Inicialización de la USART */
#define init_comms()\
    RX_PIN = 1;      \
    TX_PIN = 1;      \
    SPBRG = DIVIDER; \
    RCSTA = (NINE_BITS|0x90); \
    TXSTA = (SPEED|NINE_BITS|0x20)

void putch(unsigned char);
unsigned char getch(void);
unsigned char getche(void);

#endif

```

#### Archivo USART.C: Librería para manejar la USART

```

#include <htc.h>
#include <stdio.h>
#include "uart.h"

void
putch(unsigned char byte)
{
    /* envía un byte */
    while(!TXIF)      /* finalizó la transmisión?? */
        continue;
    TXREG = byte;
}

```

```
unsigned char
getch() {
    /* recibe un byte */
    while(!RCIF)      /* recibió un dato?? */
        continue;
    return RCREG;
}

unsigned char
getche(void)
{
    unsigned char c;
    putch(c = getch());
    return c;
}
```



# **PRÁCTICAS**

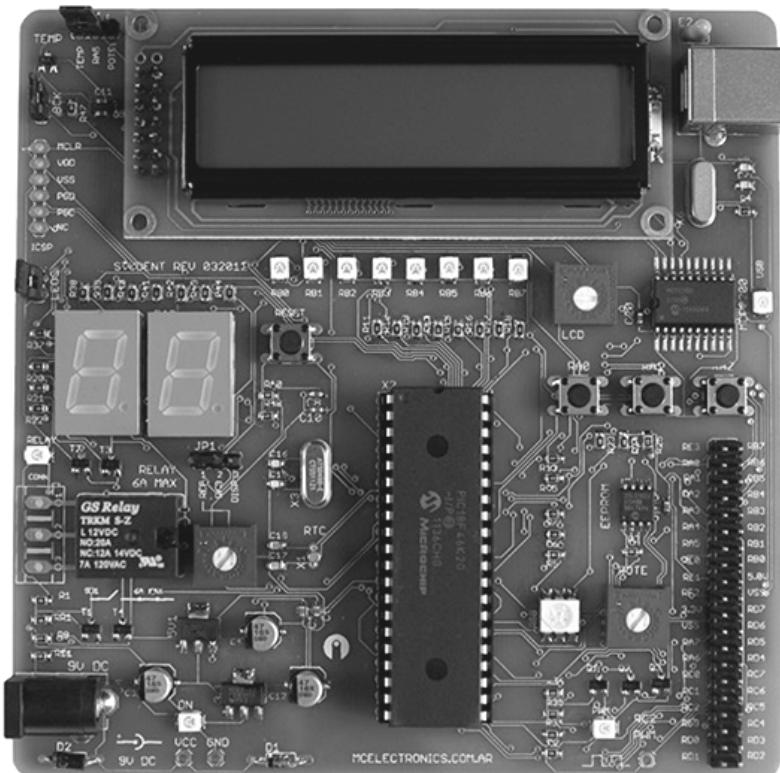
# **XC8**



## APRENDIENDO A USAR EL XC8 DE FORMA PRÁCTICA

Hasta ahora hemos aprendido a realizar un proyecto con MPLABX y los fundamentos de la programación en lenguaje C. Para aplicar los conocimientos adquiridos usaremos una placa de entrenamiento desarrollada por mcelectronics, partner de Microchip en Argentina.

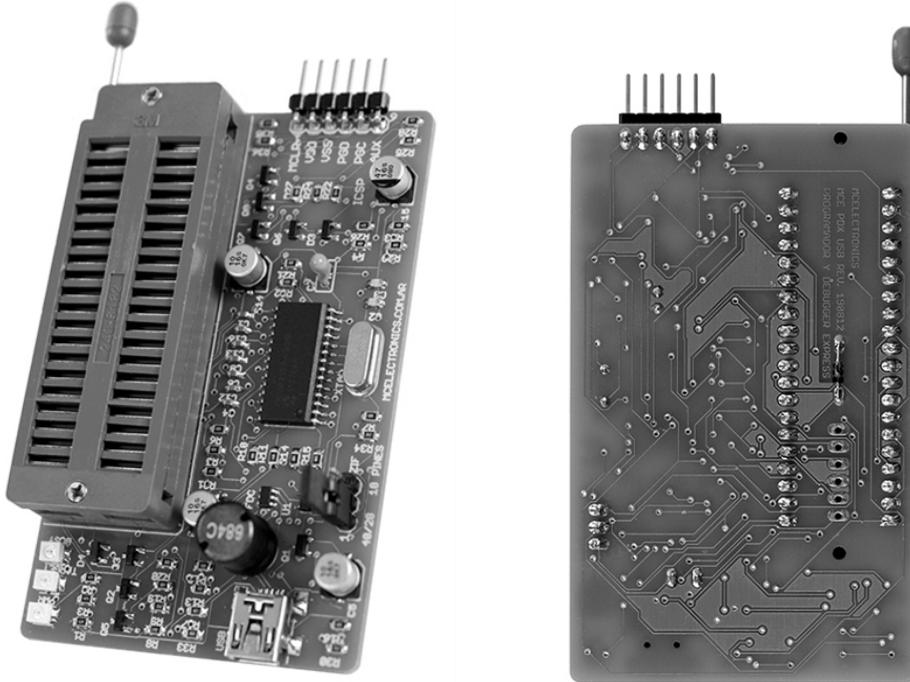
La placa se denomina MCE Starter KIT Student:



La placa contiene todos los elementos necesarios para realizar todas las prácticas básicas para el testeo de la mayoría de los periféricos del MCU:

- Control de puertos I/O
- Manejo de display 7 segmentos de forma multiplexada
- Manejo de display LCD
- Puertos analógicos para leer Tensión y Temperatura
- Control de Timers e Implementación de un RTC
- Lectura de Memoria EEPROM
- Control del PWM
- Envío y recepción de datos por USART

Para programar dicha placa usamos la herramienta de programación PDX desarrollada por la misma firma, la cual es compatible con el PICKIT2 de Microchip. Mediante el PDX se pueden programar una extensa gama de microcontroladores de Microchip desde PIC10F hasta PIC32, vía ICSP.



#### **Programdor MCE PDX compatible con PICKit2 de Microchip.**

Usted puede adaptar los ejemplos a cualquier circuito de aplicaciones que incluso haya construido, al igual que para programar el microcontrolador podrá usar cualquier programador que tenga, compre o construya.

## PROGRAMA 1: LED-PULSADOR

En esta primera práctica aplicada introduciremos el concepto básico de control, a partir de la lectura del estado de 3 pulsadores y el accionamiento de 3 Leds. Los pulsadores en la Placa Starter Kit Student se encuentran colocados en los puertos RA0, RA1 y RA2. Los led se encuentran en los Puertos RB0 al RB7. De estos solo usaremos RB0, RB1 y RB2.

```
#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

void main(void)                                //funcion principal
{
    ANSEL=0;                                     //todos los puertos como digitales
    ANSELH=0;
    TRISB= 0;                                    //todo el PORTB como salidas

    TRISA=255;                                   //todo el PORTA como entradas
    PORTA=0;                                     //inicializamos el PORTA
    PORTB=0;                                     //inicializamos el PORTB

while(1)                                         //Bucle repetitivo eterno
{
    if(RA0==1)                                  //abrimos el bucle
        RB0=1;                                    //esta pulsado RA0??
    else
        RB0=0;                                    //si es verdadero, encendemos RB0
                                                //sino
                                                //apagamos RB0

    if(RA1==1)                                  //esta pulsado RA1??
        RB1=1;                                    // si es verdadero, encendemos RB1
    else
        RB1=0;                                    //sino
                                                //apagamos RB1

    if(RA2==1)                                  //esta pulsado RA2??
        RB2=1;                                    //si es verdadero, encendemos RB2
    else
        RB2=0;                                    //sino
                                                //apagamos RB2
}

}                                                 //cerramos la función principal
```

**#include<xc.h> :**

Esta línea nos permite incluir un archivo de cabecera genérico para que el mismo incluya en el proceso de compilado el archivo de cabecera del microcontrolador que tengamos configurado en el proyecto. Dentro del archivo de cabecera del microcontrolador, el cual se identifica por el nombre del MCU por ejemplo PIC16F887.h. Dichos archivos se encuentran almacenados en la carpeta include del compilador.

Para facilitar la migración entre microcontroladores, existe el archivo xc.h el cual busca el archivo de cabecera del MCU que hayamos seteado en nuestro proyecto.

**#define \_XTAL\_FREQ 4000000:**

Esta línea le indica al compilador cual es la frecuencia de entrada al microcontrolador, con ella el compilador calcula el tiempo necesario para crear los bucles de las macros delays

**\_CONFIG(FOSC\_XT & WDTE\_OFF & CP\_OFF & LVP\_OFF ):**

Esta línea le permite al usuario setear los fusibles de configuración que define las características de funcionamiento del núcleo (core) del microcontrolador. Dichas características

**while(1):**

Creamos un bucle infinito, el cual contiene al programa que se va ejecutar constantemente.

Este programa nos muestra los elementos básicos que encontramos en todo programa de lenguaje C.

## PROGRAMA 2: PULSADOR TOUCH

En esta segunda práctica le vamos a enseñar como leer sin rebotes el estado de 3 pulsadores el cual accionará 3 Leds. Los pulsadores tendrán memoria y la misma cambiará de estado cada vez que se accione el pulsador. Los pulsadores en la Placa Starter Kit Student se encuentran colocados en los puertos RA0, RA1 y RA2. Los led se encuentran en los Puertos RB0 al RB7. De estos solo usaremos RB0, RB1 y RB2.

```
#include<xc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & LVP_OFF );

void main(void)          //funcion principal
{
    ANSEL=0;
    ANSELH=0;
    TRISB=0;
    TRISA=0b11111111;
    PORTA=0;           //inicializamos el PORTA
    PORTB=0;           //inicializamos el PORTB

    while(1)//Bucle repetitivo eterno
    {
        if(RA0==1)      // abrimos el bucle
            //accionado el pulsador??
        {
            RB0=!RB0;   //invierte el estado de RB0
            __delay_ms(2); //antirebote de entrada
            while(RA0==1); //esperamos hasta RA0=0
            __delay_ms(2); //antirrobote de salida
        }

        if(RA1==1)      //accionado el pulsador??
        {
            RB1=!RB1;   //invierte el estado de RB1
            __delay_ms(2); //antirebote de entrada
            while(RA1==1); //esperamos hasta RA1=0
            __delay_ms(2); //antirrobote de salida
        }
    }
}
```

```

        if(RA2==1)          //accionado el pulsador??
    {
        RB2=!RB2;         //invierte el estado de RB0
        __delay_ms(2);    //antirebote de entrada
        while(RA2==1);    //esperamos hasta RA2=0
        __delay_ms(2);    //antirrobo de salida
    }

}

//cerramos la función principal

```

Para evitar los rebotes que se producen en el accionamiento de un pulsador se han colocado las líneas `__delay_ms(2)`. Existe un rebote tanto al accionar el pulsador, como al soltarlo. Para evitar leer los falsos estados que se producen en ese momento de inestabilidad mecánica del pulsador, mantenemos al microcontrolador dentro del `delay`.

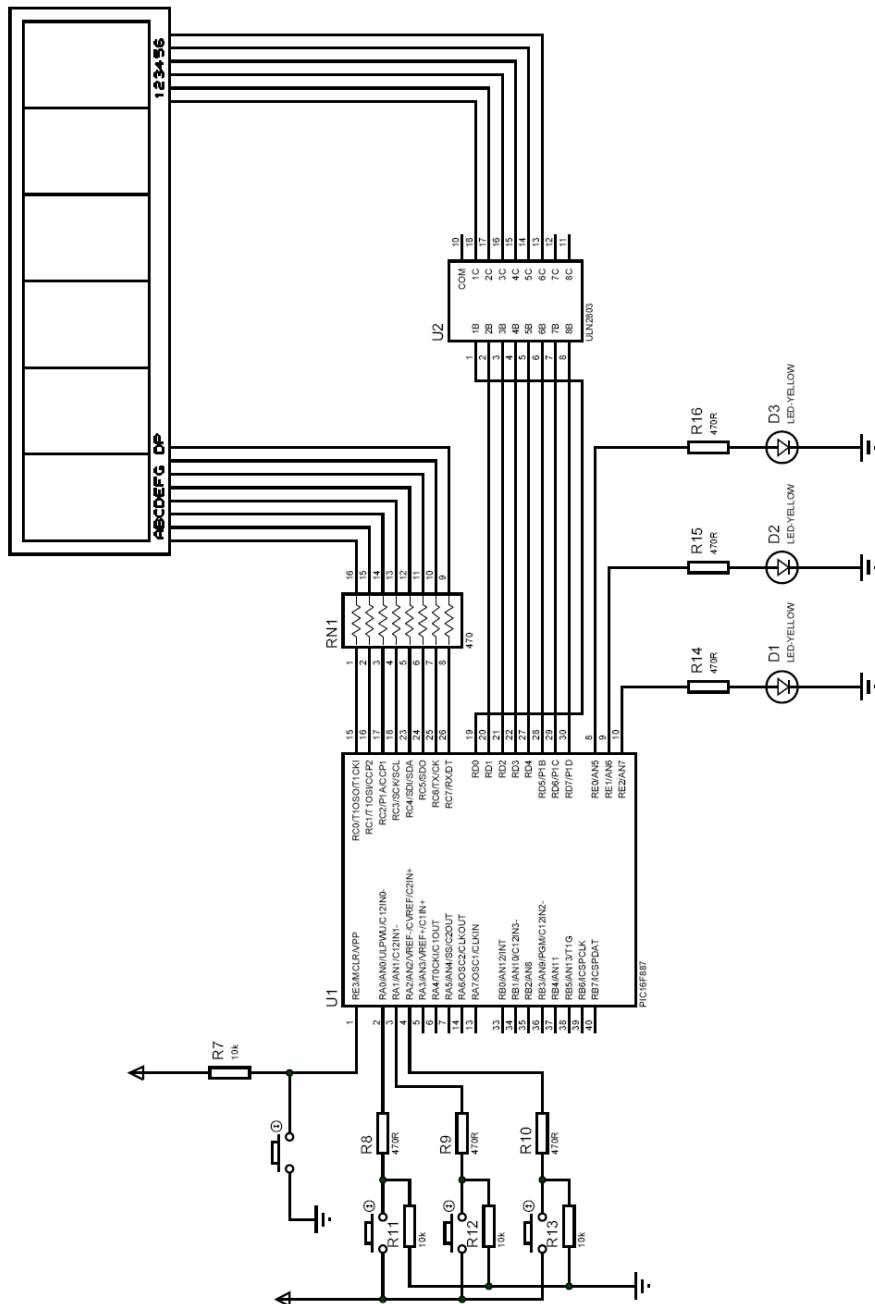
La sentencia `__delay_ms(x)` es en realidad una función embebida dentro del compilador y que se la conoce como macro. El compilador xc tiene varias macros de este tipo. En este caso la función introduce un tiempo, definido por `x` en milisegundos.

### **while(RA2==1);**

En esta línea el programa queda a la espera que el usuario deje de accionar el pulsador. Mientras el pulsador se este accionando, el programa leerá en el puerto de entrada 1 y por tanto al cumplirse la condición del `while`, el programa quedará en este punto, sin hacer nada ya que la línea se cierra con un punto y coma. Esto es lo mismo que haber creado un bucle vacío.

## PROGRAMA 3: CONTADOR 7 SEGMENTOS

En este tercer programa realizaremos un contador de 6 dígitos con cuenta ascendente y descendente. El siguiente es el circuito de aplicación:



Para este circuito decidimos usar una implementación independiente de la placa student en un circuito propio que le diera una mayor capacidad de cuenta. El circuito se realizó en PROTEUS.

Los displays están multiplexados, y son controlados por el PORTD, el cual se encuentra conectado a un ULN2803, mediante el cual se amplia la capacidad de corriente de cada puerto a unos 500ma. Los segmentos, son comunes a todos los displays o dígitos del contador, y se encuentran conectados al PORTB. Con el bit de menor peso, RBO excitamos el segmento a; y con el bit RB7 excitamos el segmento del punto decimal (dp).

El microcontrolador usado es el PIC16F887, y usamos su reloj interno seteado en 4 Mhz. El contador nos permite contar de forma ascendente o descendente, el modo se controla mediante el pulsador BTN3. Por otra parte podemos habilitar o no el estado de cuenta, accionando el pulsador BTN2. Los pulsos a contar son simulados por el pulsador BTN1, y aplicados a RA0.

```
#include<htc.h>
#include<stdio.h>
#define _XTAL_FREQ 4000000

__CONFIG(FOSC_INTRC_NOCLKOUT & WDTE_OFF & CP_OFF & LVP_OFF );

//declaracion de funciones
void LeerTeclado(void);
void MostrarDisplay(void);

//etiquetas del pines
//entradas
#define BTN1 RA0
#define BTN2 RA1
#define BTN3 RA2

//salidas
#define LED_3 RE0
#define LED_2 RE1
#define LED_1 RE2

#define DIGITO1 RD0
#define DIGITO2 RD1
#define DIGITO3 RD2
#define DIGITO4 RD3
#define DIGITO5 RD4
#define DIGITO6 RD5

//constantes globales
const unsigned char Conver7SEG[10]={0b00111111,
```

```

0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111101,0b00000111,0b01
111111,0b01100111};

//variables globales
unsigned long contador=0;
unsigned char buffer[10];
bit ESTADO=0,MODO=0;

//main
void main(void)           //funcion principal
{
    IRCF0=0;               //CONFIGURO OSCILADOR interno a 4MHz
    IRCF1=1;
    IRCF2=1;
    SCS=1;

    ANSEL=0;
    ANSELH=0;

    TRISA=255;

    TRISC=0;
    TRISD=0;
    TRISE=0;

    PORTD=0;
    PORTE=0;
    PORTC=0;           //inicializamos los PORT

    while(1)             //Bucle repetitivo eterno
    {
        // abrimos el bucle
        LeerTeclado();
        MostrarDisplay();
    }

}                         //cerramos la función principal

void LeerTeclado(void)
{
    if(BTN1==1)
    {
        if(ESTADO==1)
        {
            if(MODO==1)
            {

```

```

        contador=contador+1;
        if(contador==1000000)
            contador=0;
    }
    else
    {
        contador=contador-1;
        if(contador>999999)
            contador=999999;
    }

}

do
{
    MostrarDisplay();
}
while(BTN1==1);

}

if(BTN2==1)
{
    ESTADO=!ESTADO;
    do
    {
        MostrarDisplay();
    }
    while(BTN2==1);
}

if(BTN3==1)
{
    MODO=!MODO; //invierte Bandera
    do
    {
        MostrarDisplay();
    }
    while(BTN3==1);
}

}

void MostrarDisplay(void)
{
    if(ESTADO==1)
        LED_1=1;
}

```

```
else
    LED_1=0;

    sprintf(buffer,"%06lu",contador);

    PORTC=Conver7SEG[(buffer[0]-0x30)];
    DIGITO1=1;
    __delay_ms(1);
    DIGITO1=0;
    PORTC=0;

    PORTC=Conver7SEG[(buffer[1]-0x30)];
    DIGITO2=1;
    __delay_ms(1);
    DIGITO2=0;
    PORTC=0;

    PORTC=Conver7SEG[(buffer[2]-0x30)];
    DIGITO3=1;
    __delay_ms(1);
    DIGITO3=0;
    PORTC=0;

    PORTC=Conver7SEG[(buffer[3]-0x30)];
    DIGITO4=1;
    __delay_ms(1);
    DIGITO4=0;
    PORTC=0;

    PORTC=Conver7SEG[(buffer[4]-0x30)];
    DIGITO5=1;
    __delay_ms(1);
    DIGITO5=0;
    PORTC=0;

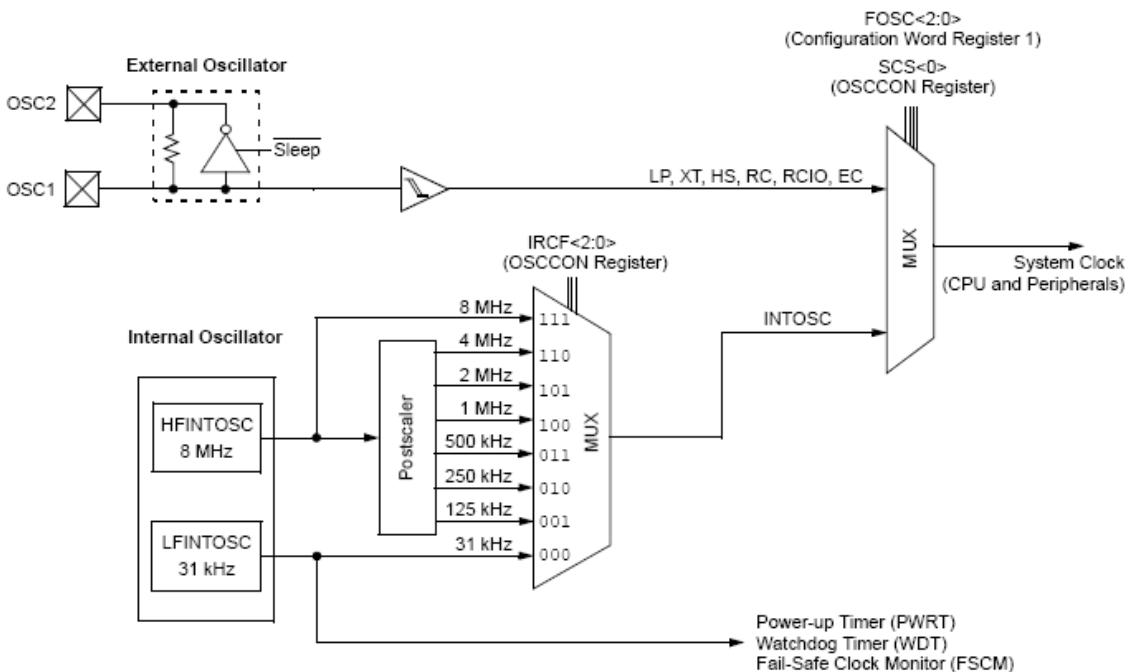
    PORTC=Conver7SEG[(buffer[5]-0x30)];
    DIGITO6=1;
    __delay_ms(1);
    DIGITO6=0;
    PORTC=0;
}
```

## EL PROGRAMA:

Para poder trabajar con el oscilador interno hemos seteado el fusible de configuración interno correspondiente en los bits de configuración:

FOSC\_INTRC\_NOCLKOUT

Sin embargo también se hace necesario que agreguemos en el código el seteo de la frecuencia a la cual vamos a trabajar pues el oscilador interno desemboca en un POSCALER (divisor de frecuencia) que nos permite obtener distintas frecuencias de trabajo:



```
IRCF0=0; //CONFIGURO OSCILADOR interno a 4MHz
IRCF1=1;
IRCF2=1;
SCS=1;
```

El bucle principal se ha simplificado creando 2 funciones, una muestra la información en display y la otra lee si ha ingresado un pulso o si se accionó algún pulsador de funciones. Esto nos permite crear una estructura de mantenimiento del código más dinámica y prolífica.

En adelante, para la mayoría de los ejemplos que vamos a ver hemos repetido el mismo esquema; un bucle simple desde el cual se invocan a las funciones más complejas:

```

while(1)           //Bucle repetitivo eterno
{
    // abrimos el bucle
    LeerTeclado();
    MostrarDisplay();
}

```

La función LeerTeclado no presenta mayores inconvenientes ya que el abc de su operación la vimos en el Programa Nº2, sin embargo aquí existe el agregado del MostrarDisplay, al pulsarse el botón:

```

if(BTN2==1)
{
    ESTADO=!ESTADO;
    do
    {
        MostrarDisplay();
    }
    while(BTN2==1);
}

```

Y mantenemos el MostrarDisplay() mientras se esté pulsando el mismo, ya que el muestreo del display lo realiza el microcontrolador, y por lo tanto no puede no realizar ninguna tarea. Por ello mientras el usuario mantenga el pulsador accionado debemos continuar refreshando la información sobre el display.

La estructura de la entrada RA0, es diferente a las otras ya que es la que debe llevar el estado de cuenta:

```

if(ESTADO==1)
{
    if(MODO==1)
    {
        contador=contador+1;
        if(contador==1000000)
            contador=0;
    }
    else
    {
        contador=contador-1;
        if(contador>999999)
            contador=999999;
    }
}

```

Tanto ESTADO como MODO son dos banderas (FLAGS) en memoria de 1 bit (observe que así han sido declaradas), las cuales cambian de estado al ser accionados los pulsadores correspondientes. En el código para poder incrementar o decrementar al contador debemos testear si ESTADO=1 ya que esta condición habilita el conteo del contador, luego en función de MODO el contador se incrementa o decremente.

Para mostrar el estado de cuenta en el display es donde tenemos el mayor trabajo, ya que la rutina debe primero decodificar la cuenta a 7 segmentos luego realizar el barrido de los display. La cuenta podía realizarse de dos formas, la mas sencilla, pero que insumía 3 mucho más código, consistía en crear 6 contadores para contener cada dígito e incrementarlos en cascada, copiando el viejo modelo de contadores en cascada de la lógica discreta. La otra forma era mas simplificada, y mostraba un lado poco recurrente pero excelente del uso de las funciones estándar de C. nosotros optamos para este libro, esta última técnica, ya que les permitirá aprender como emplear de forma menos convencional las funciones estándar de C.

El contador se implementó como una variable del tipo unsigned long contador=0; para incrementarla o decrementarla sencillamente.

Luego para decodificarla el primer paso es convertir la cuenta en un string o conjunto de caracteres ASCII, los cuales terminarán con un '\0'. Para hacer esta conversión usamos la siguiente función del C estándar:

```
sprintf(buffer,"%06lu",contador);
```

La misma es una variante del printf con la diferencia de que sprintf escribe con formato dentro de una variable tipo string, que en nuestro caso se denomina buffer y al cual le hemos asignado el tamaño de [10] caracteres.

La función lee el contenido del contador y lo almacena en el array buffer con el formato de 6 caracteres sin signo en formato ASCII.

El formato se lo hemos establecido con "%06lu" de esta manera no importa cuantos dígitos tenga la cuenta actual, el especificador de formato, los completará con 0 si estos no alcanzan los 6 dígitos. Esto permite simplificar la rutina de muestreo.

Una vez convertido a string, pasamos a leer cada elemento del buffer, tomando en cuenta que los array tienen almacenado su primer elemento con el índice cero:

```
buffer[0]-0x30
```

Esta forma nos permite, que leido l elemento del buffer, restarle el hexadecimal 0X30 para poder obtener el valor BCD del número almacenado (ya que los números 0 al 9, en ASCII se almacenan como 0x30 a 0x39).

De esta forma obtenemos el número de cada dígito en BCD para luego usarlo como puntero de una tabla de conversión BCD a 7 segmentos realizado con un array de constantes, el cual hemos declarado al principio del código:

```
const unsigned char Conver7SEG[10]={0b00111111,  
0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111101,0b00000111,0b01  
111111,0b01100111};
```

De esta forma obtenemos el dígito convertido a 7 segmentos para luego enviarlo al display:

```
PORTC=Conver7SEG[(buffer[0]-0x30)];
```

Luego para hacer el muestreo activamos el dígito correspondiente, durante un tiempo determinado, y posteriormente lo apagamos, borrando además el puerto para evitar arrastrar visualmente la lectura a los dígitos siguientes:

```
DIGITO1=1;  
__delay_ms(1);  
DIGITO1=0;  
PORTC=0;
```

El tiempo de muestreo (`__delay_ms(1)`) determina el nivel de intensidad lumínica de los segmentos del display, ya que a mayor tiempo, mayor energía eléctrica es convertida en energía fotónica en el display.

Sin embargo si el tiempo de muestreo es demasiado largo, caemos dentro del límite de la persistencia retiniana del ojo (unos 20ms) y tenemos efecto parpadeo. Para evitar dicho fenómeno, el muestreo de todos los dígitos debe realizarse antes de ese tiempo. La práctica demuestra que si se realiza el muestreo 100 veces por segundo, es imperceptible el multiplexado, y el ojo integra toda la muestra.

#### PROGRAMA 4: MENSAJE EN UN DISPLAY LCD

En este nuevo ejemplo vamos a realizar la escritura de un mensaje en un display del tipo LCD inteligente alfanumérico de 2 líneas por 16 caracteres como el que tiene la placa de entrenamiento Starter Kit Student.

Para manejar el LCD se modificó la librería original de microchip, denominada XLCD y que viene para PIC18, la cual originalmente se diseño para el compilador MPLAB C18. Esta librería esta adaptada al compilador XC8 y a la familia PIC16F.

Puede encontrar la librería al instalar el XC8 o en el DVD del libro.

Como puede observar en la primera parte de la librería se deben definir los pines físicos donde se ha conectado el LCD .

En la placa Student, el LCD esta conectado mediante una interfaz de 6 conexiones:

LCD	MCU
RS.....	RE1
RW.....	GND
E.....	RE2
D4.....	RD4
D5.....	RD5
D6.....	RD6
D7.....	RD7

Es por esta razón que hemos configurado la librería de la siguiente forma:

```
/* Interfaz 4 u 8 bits
 * Para 8 bits descomente el #define BIT8
 */
/* #define BIT8 */

/* Si trabaja en 4 bits y usa el nibble bajo comente la siguiente linea
 */
#define UPPER

/* Si no usa el pin R/W para chequear el estado BUSY del LCD comente la
 * siguiente linea y conecte el pin R/W del LCD a GND.
 */
#ifndef BUSY_LCD

/* DATA_PORT define a que Puerto estan conectadas las lineas de dato */
#define DATA_PORT PORTD
#define TRIS_DATA_PORT TRISD
```

```

/* CTRL_PORT define donde estan conectadas las lineas del PORT de control.
 */
#define RW_PIN  PORTEbits.RE0          /* PORT para RW */
#define TRIS_RW TRISEbits.TRISE0      /* TRIS para RW */

#define RS_PIN  PORTEbits.RE1          /* PORT para RS */
#define TRIS_RS TRISEbits.TRISE1      /* TRIS para RS */

#define E_PIN   PORTEbits.RE2          /* PORT para D */
#define TRIS_E  TRISEbits.TRISE2      /* TRIS para E */

```

En nuestros proyecto deberemos colocar dentro de la carpeta del proyecto la librería, la cual será invocada por el programa principal. Nuestro programa simplemente ahora escribirá un mensaje en la primera y segunda línea del LCD.

A continuación listamos el código de nuestro programa:

```

#include <xc.h>
#define _XTAL_FREQ 4000000
#include "lcd_pic16.c"
__CONFIG( FOSC_INTRC_NOCLKOUT & WDTE_OFF & PWRTE_OFF & MCLRE_ON & CP_OFF &
CPD_OFF &
BOREN_OFF & IESO_OFF & FCMEN_OFF);

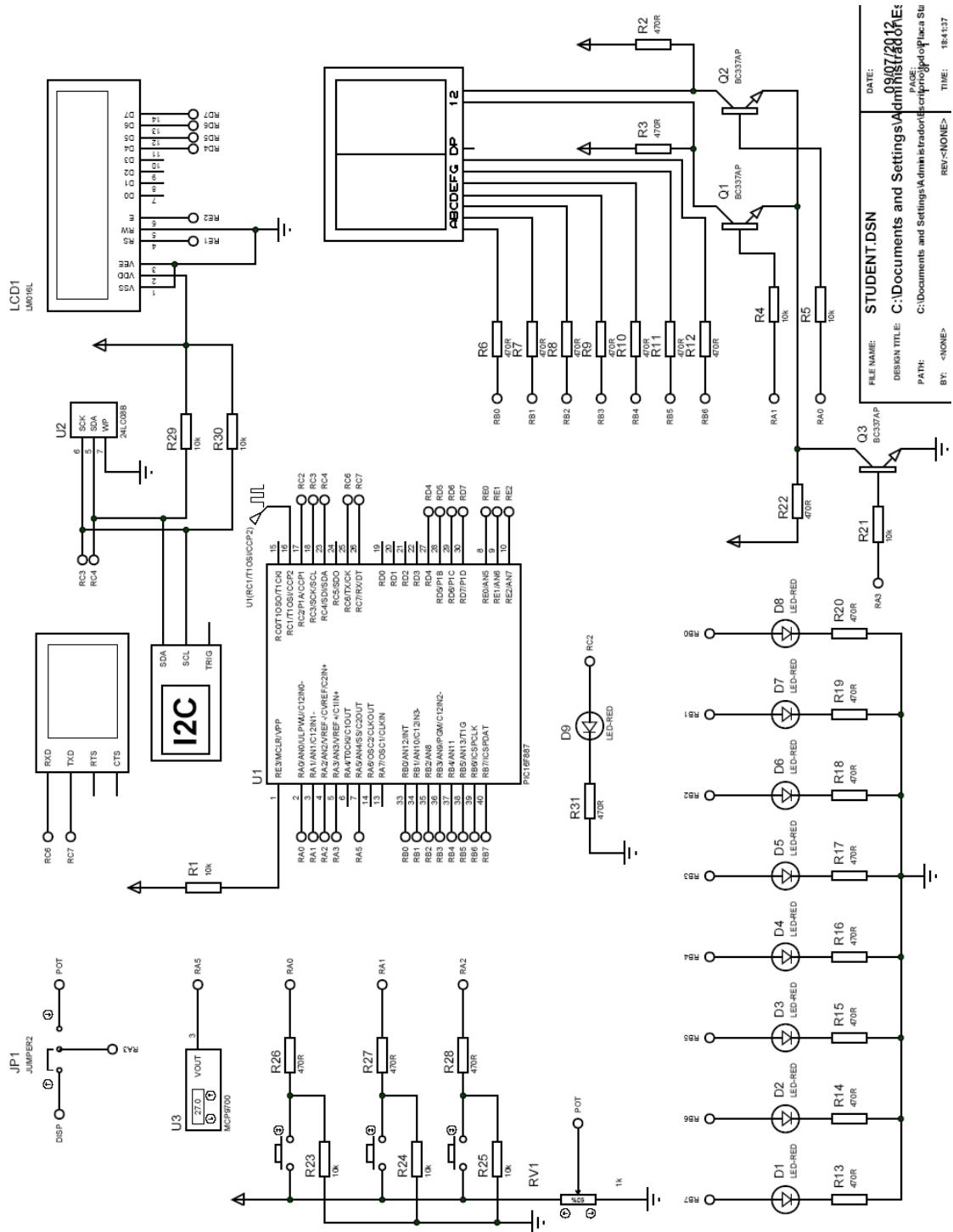
void SendCmdLCD(unsigned char CMD);

void main(void) {
    ANSEL=0;
    ANSELH=0;
    OpenXLCD(FOUR_BIT & LINES_5X7 );
    SendCmdLCD(CURSOR_OFF);
    while(1)
    {
        SetDDRamAddr(0x00);
        putrsXLCD("Microchip");
        SetDDRamAddr(0x40);
        putrsXLCD("Compilador XC8");
    }
}

```

```
void SendCmdLCD(unsigned char CMD)//escribe un comando
{
    while(BusyXLCD());
    WriteCmdXLCD(CMD);
```

En la siguiente figura podemos ver el esquemático de la placa Student de MCElectronics



## PROGRAMA 5: CONTADOR CON DISPLAY LCD

En este nuevo programa veremos como implementar un contador ascendente / descendente usando un LCD inteligente para mostrar el estado de cuenta. Como en el ejemplo anterior usaremos la placa Starter Kit Student. Por lo tanto usaremos los pulsadores ubicados en RA0 y RA1 para manejar el contador.

En la carpeta de nuestro proyecto colocaremos la librería del LCD que usamos en el ejemplo anterior. El código es muy sencillo, lo listamos a continuación:

```
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "lcd_pic16.c"

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & LVP_OFF & CP_OFF & CPD_OFF &
BOREN_ON);

//variables globales
unsigned int contador=0;
unsigned char buffer[20];
bit flag_modo=0;

void init(void)
{
    ANSEL=0;
    ANSELH=0;
    TRISB=0xFF;
    TRISA=0xFF;
    TRISD=0;
    TRISE=0;
    OpenXLCD(FOUR_BIT & LINES_5X7 );
}

void muestra_LCD (void)
{
    SetDDRamAddr(0x03);
    putrsXLCD("modo:");
    if (flag_modo==0)
        putrsXLCD("UP ");
    else
        putrsXLCD("DOWN");
    SetDDRamAddr(0x43);
    sprintf(buffer,"cuenta:%05u",contador);
    putsXLCD(buffer);
}
```

```

void main(void)
{
    init();
    while(1)
    {
        muestra_LCD();
        if (RA0==1)
        {
            if(flag_modo==0)
                contador=contador+1;
            else
                contador=contador-1;
            muestra_LCD();
            __delay_ms(2);
            while(RA0==1);
            __delay_ms(2);
        }
        if (RA1==1)
        {
            flag_modo=!flag_modo;
            muestra_LCD();
            __delay_ms(2);
            while(RA1==1);
            __delay_ms(2);
        }
    }
}

```

En este caso el código del contador se simplifica ya que el MCU no tiene que encargarse del barrido del LCD, ya que de ello se encarga el controlador propio del LCD. Sin embargo para mostrar la información del conteo, debemos traducir a formato ASCII dicha cuenta. Esto lo hacemos mediante la sentencia:

```
sprintf(buffer,"cuenta:%05u",contador);
```

Como lo hicimos en el caso del contador 7 segmentos.

Observe que los mensajes fijos son almacenados en la ROM y desplegados en pantalla mediante la función putrsXLCD(). Mientras que los strings almacenados en la RAM son enviados al LCD mediante la función putsXLCD().

Por otra parte antes de enviar cada mensaje primero posicionamos el cursor del display sobre el lugar que queremos, usando SetDDRamAddr().

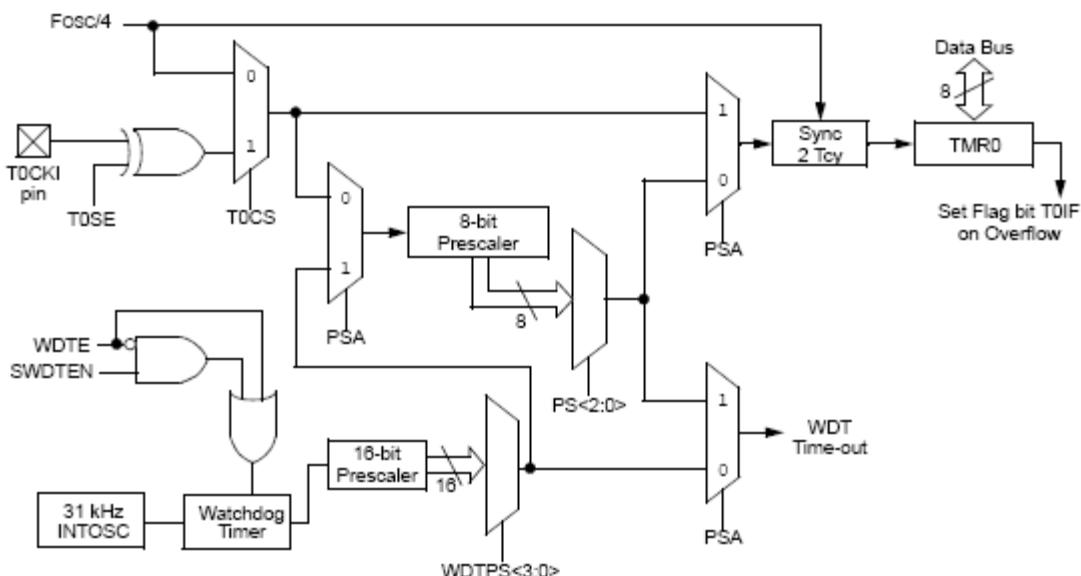
El flag\_modo le indica al programa si el contador, funciona en modo ascendente o descendente. Observe que cada vez que se modifica tanto el estado de cuenta, como el estado del flag\_modo esto se refleja en el LCD pues se dispara el refresco del mismo mediante la rutina muestra\_LCD();

## PROGRAMA 6: CONTADOR CON TIMER 0

En este ejemplo vamos a enseñarle a programar el TIMER 0 conocido en el MCU como TMRO. Los microcontroladores PIC incorporan serie de contadores integrados conocidos como "Timers/Counters", ya que cuando el contador cuenta pulsos provenientes del exterior funcionan como verdaderos contadores de eventos. Sin embargo, el contador también puede contar pulsos internos cuya fuente es el ciclo de máquina interno, es decir Fosc/4, en este caso se denomina Timer, pues en este caso cuenta tiempo.

En los PIC16F solo podemos encontrar hasta 3 Timers, mientras que en los PIC18F podemos encontrar hasta 5 Timers, dependiendo el modelo.

En este ejercicio practicaremos con el Timer 0. Como lo haremos funcionar en la Iaca STUDENT, y esta no tiene destinada ninguna fuente externa para excitar a este contador, lo conectaremos a la fuente interna de clock Fosc/4 y activaremos un divisor de frecuencia interno que trae , en un factor de división 256. De esta forma retrazaremos el estado de cuenta para poder ver la misma en el LCD. En la siguiente figura podemos ver el esquema del TMRO:



El TMRO es un contador de 8 bits. Este contador está controlado por un registro de control denominado OPTION\_REG. Cada bit de dicho registro controla una función del TMRO.

La fuente que excita el TMRO puede ser interna o externa según como se programe el bit TOCS. Cuando TOCS=1, la fuente de clock es externa, caso contrario, es interna, la salida de la unidad interna de Temporización la cual excita el núcleo de la CPU, a la frecuencia Fosc/4.

Cuando la fuente de clock es externa, la señal debe aplicarse al Terminal TOCKI, y en este caso es posible seleccionar el flanco al cual se incrementará el contador, mediante la programación del bit TOSE. Cuando TOSE=0, el contador se incrementa por flanco ascendente, cuando TOSE=1, se incrementa por flanco descendente.

Si la frecuencia de la señal de entrada es muy alta , la misma pede dividirse por un factor N. Para esto existe un Divisor de frecuencia programable , conocido como PRESCALER, ya que el mismo se encuentra a la entrada del TMRO.

Dicho PRESCALER se comparte con el Watch Dog Timer, como consecuencia existe un bit el cual permite asignar el PRESCALER a la salida del Watch Dog Timer, para retardar su tiempo de accionamiento, o asignarlo a la entrada del TMRO. De esta forma si PSA=0, el PRESCALER queda conectado al TMRO, si PSA=1, el mismo queda conectado a la salida del WDT.

Cuando se usa el PRESCALER, el factor de división se programa mediante tres bits denominados PS0, PS1, PS2.

En el programa que se ha elaborado, se configuró el TMRO para que funcione excitado por la fuente interna TOCKI. Se asigno el PRESCALER con un factor de división máximo, 256. La cuenta podrá observarse en el display.

```
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "lcd_pic16.c"

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & LVP_OFF & CP_OFF & CPD_OFF &
BOREN_ON);

//variables globales
char buffer[20];

//funciones prototipo
void init(void);
void muestra_LCD(void);
void Open_IO(void);
void Open_TMRO(void);

//función principal
void main(void)
{
    init(); //Inicializa los puertos I/O, LCD y Timer0
    while(1)
    {
        muestra_LCD();
    }
}
```

```

}

//Configura los puertos I/O
void Open_IO(void)
{
    ANSEL=0;
    ANSELH=0;
    TRISB=0xFF;
    TRISA=0xFF;
    TRISD=0;
    TRISE=0;

}

//Configura el Timer0
void Open_TMR0(void)
{
    TOCS=0;// fuente de excitación del TMR0 Fosc/4
    PS0=1;// Prescaler en 256
    PS1=1;
    PS2=1;
    PSA=0; //TMR0 con prescaler
    TMR0=0;//borramos el Timer
}

void init(void)
{
    Open_IO();
    Open_TMR0();
    OpenXLCD(FOUR_BIT & LINES_5X7 );
}

void muestra_LCD (void)
{
    SetDDRamAddr(0x03);
    putrsXLCD("Contador TMR0");
    SetDDRamAddr(0x43);
    sprintf(buffer,"cuenta:%03u",TMR0);
    putsXLCD(buffer);
}

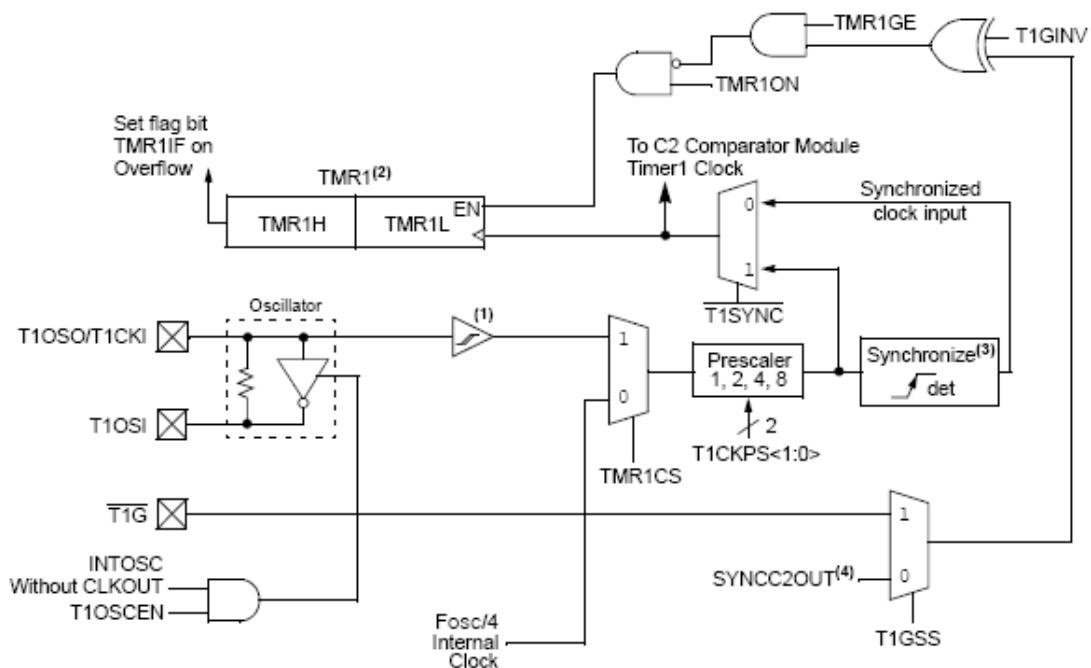
```

La arquitectura del programa es totalmente modular; existen una serie de funciones que configuran y muestran el funcionamiento del TMRO. Así Open\_IO(), configura los puertos I/O; Open\_TMRO(), configura el funcionamiento del TMRO y muestra\_LCD(), lee el contenido del TMRO y lo muestra en el LCD.

Es interesante destacar que apesar de que su estado de cuenta máximo es pequeño, ya que solo puede contar desde 0 a 255 por ser TMRO de 8 bits, este timer es el único que tiene la posibilidad de programar el flanco de excitación de la señal de clock. Esta característica es muy útil en los automatismos industriales.

## PROGRAMA 7: CONTADOR CON TIMER 1

El Timer 1 es un Contador que al igual que lo hace el TMR0, puede contar tanto pulsos internos como externos, y en este último caso, dichos pulsos pueden ser generados externamente o a partir de un oscilador propio que incorpora el Timer. En la siguiente figura podemos ver el diagrama interno del Timer 1:



En los terminales T1OSO y T1OSI se puede colocar un cristal, por ejemplo de la frecuencia de 32768Hz para crear una base de tiempo para un reloj. En este caso se deberá poner en uno el bit T1OSCEN, lo cual habilita una compuerta inversora con su correspondiente resistor de realimentación, mediante la cual se crea un oscilador interno, conocido como oscilador secundario.

El Timer1 también puede funcionar como contador de eventos externos, en este caso los pulsos deben aplicarse al terminal T1OSO/T1CK. El bit de control TMR1CS debe ser puesto en uno. Si por el contrario se quiere hacer trabajar al Timer1 de forma interna, se colocará el bit TMR1CS en cero.

Tanto para la fuente interna como la externa se puede adicionar un divisor de frecuencia o también conocido como PRESCALER (porque se encuentra a la entrada del Timer, si estuviera en la salida, recibiría el nombre de POSTSCALER) programable, el cual puede configurarse para que divida la frecuencia de entrada por 1 (no divide), por 2, por 4 o por 8. Dichos pulsos de entrada pueden sincronizarse con el reloj de la CPU o funciona de forma asincrónica, mediante el seteo del bit T1SYNC.

Para encender el Timer1 existe el bit TMR1ON, por tanto una vez configurado el funcionamiento de este timer coloque este bit en 1 para que el mismo comience a funcionar:

Por último se desea puede habilitar el funcionamiento del timer 1 por medio del pin T1G o compuerta de habilitación o gate. Cuando este pin es puesto en cero el Timer1 esta habilitado para contar. Para habilitar esta forma de “disparar el funcionamiento del Timer1” debe colocarse en 1 en bit T1GSS

El siguiente programa hace funcionar al Timer 1 como contador cuya fuente de excitación es el reloj interno y se puede programar el factor de división.

```
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include "lcd.h"

__CONFIG(FOSC_XT & WDTE_OFF & CP_OFF & PWRTE_ON & LVP_OFF & CP_OFF & CPD_OFF &
BOREN_ON);

//variables globales
char divisor=0;
char buffer[20];

//declaracion de funciones prototipo
void scan_teclado(void);
void init(void);
void muestra_LCD (void);
void setup_timer1(void);
void control_t1(void);

void main(void)
{
    RD5=0;
    RD7=1;
    init(); //configuramos I/O
    setup_timer1(); //Configuramos inicialmente el Timer1
    while(1)
    {
```

```

        scan_teclado();
        muestra_LCD();
        control_t1();
    }

}

void control_t1(void)
{
    switch(divisor)
    {
        case 0:
            T1CKPS0=0;// divisor x1
            T1CKPS1=0;
            break;
        case 1:
            T1CKPS0=1;// divisor x2
            T1CKPS1=0;
            break;
        case 2:
            T1CKPS0=0;// divisor x4
            T1CKPS1=1;
            break;
        case 3:
            T1CKPS0=1;// divisor x8
            T1CKPS1=1;
            break;
    }
}

void setup_timer1(void)
{
    TMR1CS=1;//habilitamos fuente externa
    T1OSCEN=1;//habilitamos el oscilador interno de 32768hz
    T1CKPS0=0;// divisor x1
    T1CKPS1=0;
    nT1SYNC=0;// T1 sincronizado con clock interno
    TMR1GE=0;// disparo externo desactivado
    T1GINV=0;
    TMR1ON=0;//Timer1 apagado
    TMR1H=0;
    TMR1L=0;
}

void scan_teclado(void)
{
    if(RB0==0)

```

```

{
    TMR1ON=!TMR1ON;
    muestra_LCD();
    __delay_ms(2);
    while(RB0==0);
    __delay_ms(2);
}
if(RA4==0)
{
    divisor=divisor+1;
    if(divisor==4)
        divisor=0;
    muestra_LCD();
    __delay_ms(2);
    while(RA4==0);
    __delay_ms(2);
}
}

void init(void)
{
    ANSEL=0;
    ANSELH=0;
    TRISB0=1;
    TRISA=0xFF;
    TRISD=0;
    TRISE=0;
    lcd_init(FOURBIT_MODE);
}

void muestra_LCD (void)
{
    unsigned int Timer1=0; //contiene la cuenta del Timer en 16 bits
    lcd_goto(0x01);
    lcd_puts("TMR1:");
    if(TMR1ON==1)
        lcd_puts("ON ");
    else
        lcd_puts("OFF ");
    switch(divisor)
    {
        case 0:
            lcd_puts("DIV:X1");
            break;
        case 1:
            lcd_puts("DIV:X2");
}

```

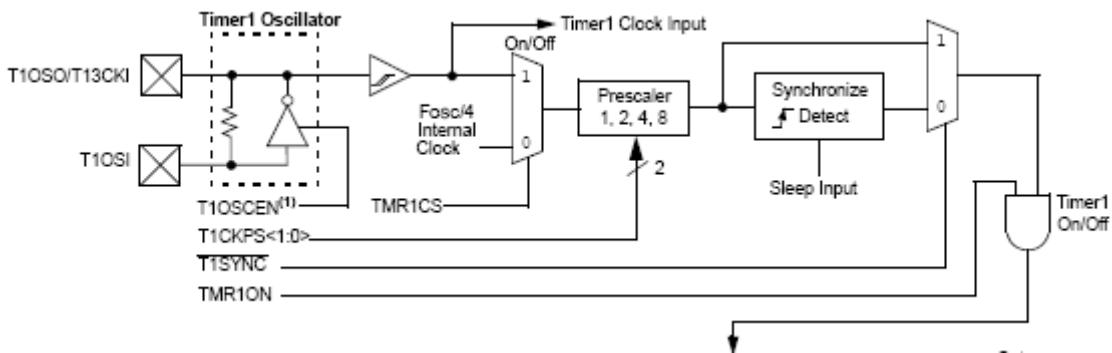
```
        break;
    case 2:
        lcd_puts("DIV:X4");
        break;
    case 3:
        lcd_puts("DIV:X8");
        break;
}
lcd_goto(0x43);

//concatenamos el los 2 registros del Timer en la variable Timer1
Timer1=TMR1H;
Timer1=Timer1<<8;
Timer1=Timer1+TMR1L;
///////////////////////////////
sprintf(buffer,"cuenta:%05u",Timer1);
lcd_puts(buffer);
}
```

## PROGRAMA 8: RELOJ DE TIEMPO REAL CON TIMER 1

Un RTR es un Reloj de Tiempo Real , que en ingles se lo suele abreviar como RTC. En los microcontroladores PIC es posible implementar esta función sobre un Timer especialmente diseñado para ello, el Timer 1(TMR1).

Como lo mencionamos anteriormente el Timer1 esta dotado de una compuerta inversora realimentada negativamente por medio de un resistor interno. Dicha compuerta tiene su entrada y salida accesible entre los terminales T1OSO y T1OSI:



Entre esos terminales se suele colocar un cristal de 32768hz. De esta forma el Timer1 configurado para que cuente pulsos externos y precargado con el valor 32768, se desbordará una vez por segundo, ya que cuenta hasta 65535 por ser de 16 bits. Basados en ese principio se puede configurar la interrupción del programa principal para que el desborde del Timer1 dispare una rutina que actualice una serie de registros que cuenten tiempo (hora, minuto y segundo). Lo que pretendemos en este punto es que aprenda a usar el sistema de interrupciones para realizar un reloj.

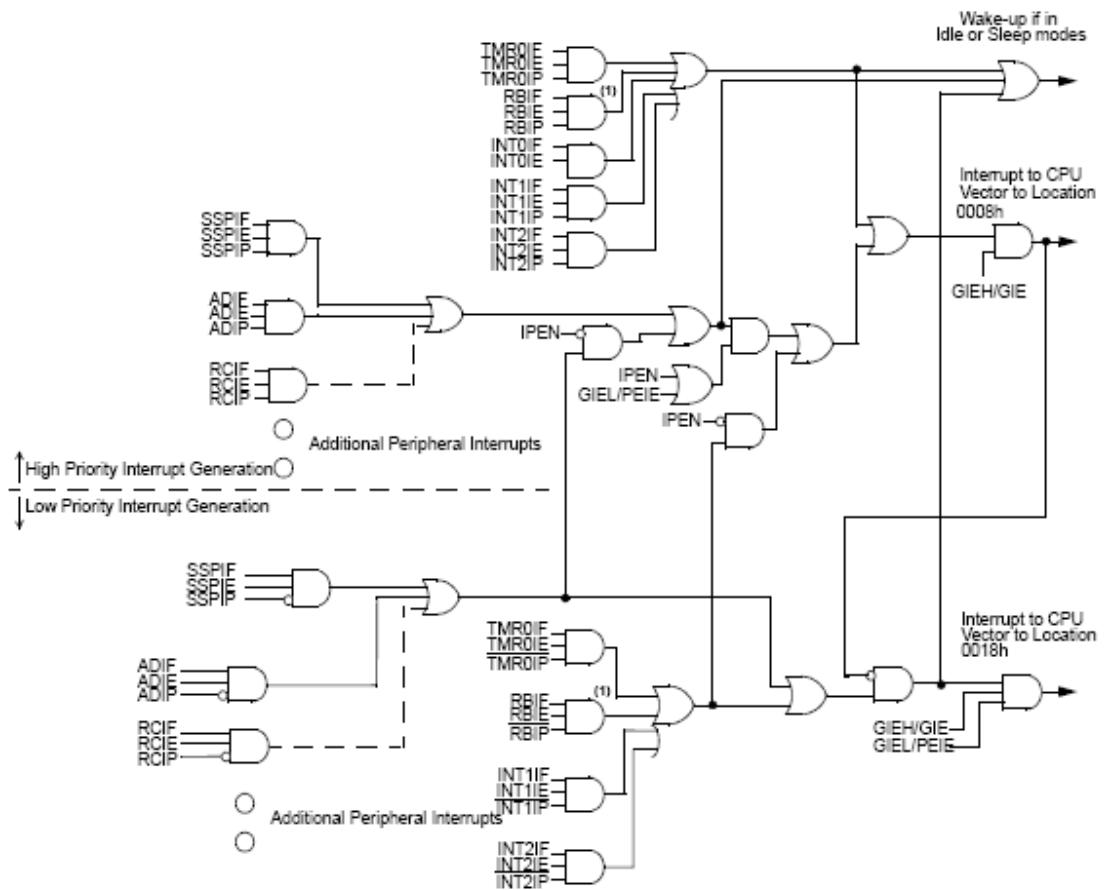
Antes de lanzarnos al código introduciremos el concepto de una interrupción y su manejo en lenguaje C.

### Un breve vistazo a las interrupciones:

Básicamente una interrupción es una rutina que se ejecuta disparada por un evento del hardware. Entiéndase por evento del hardware, por ejemplo al final de una conversión del ADC, el desborde de un Timer, la llegada de un dato a la USART, etc.

Para que dicha interrupción se pueda ejecutar, el elemento o periférico interruptor debe estar habilitado localmente para generar una petición de atención de interrupción. Además debe estar habilitado el Sistema Global de Interrupciones.

Desde el punto de vista lógico el sistema de interrupciones es un circuito combinacional de compuertas lógicas del tipo AND y OR como se muestra esquemáticamente en la siguiente figura:



En los microcontroladores PIC18F el sistema de interrupciones puede ser configurado como sistema de interrupciones compatible con PIC16F, también conocido como modo legal, o sistema de interrupciones por prioridades.

### Sistema de interrupciones en modo Legal

En este caso el sistema de interrupciones funciona como en los PIC16F, existe un único vector de interrupciones, el cual se encuentra en la dirección hexadecimal 0X000008. Cualquier fuente generadora de interrupción que genere una interrupción provocará que el programa deje de procesar la rutina en la que actualmente se encuentra y pase a leer las instrucciones que se encuentran a partir de la dirección 0x00008.

Sin embargo a diferencia de los PIC16F, los PIC18F poseen una serie de registros dedicados, denominados “registros sombra”, a los cuales el usuario no tiene acceso, y que son usados por el microcontrolador para salvaguardar el contenido de los registros mas importantes del núcleo del procesador (WREG, BSR, STATUS), de forma automática. En los PIC16F, esta operación esta a cargo del programador, sin embargo el lenguaje C se encarga de hacerlo él, siendo esta operación transparente al mismo.

En los PIC18F al cursar la interrupción el sistema automáticamente salvaguarda la dirección de la próxima instrucción a ejecutar, sobre la pila o STACK, y el contenido de los registros del núcleo del procesador, en los registros sombra y salta a la dirección 0X00008, inhabilitando la posibilidad de otra interrupción, para evitar que dentro de la interrupción actual, pueda cursar otra.

Para controlar las interrupciones existen 3 registros de control, conocidos como INTCON, INTCON2 e INTCON3, además existen 2 registros que permiten habilitar las interrupciones de los periféricos y que se denominan PIE1 y PIE2, 2 registros que generan la petición de interrupciones, a los cuales se los denomina registros bandera o FLAG (PIR1 y PIR2), estos son activados por el evento del hardware que desemboca la interrupción, y por ultimo existen 2 registros que permiten indicar la prioridad de la interrupción y que se conocen como IPR1 e IPR2, pero que cuando se trabaja en modo legal, no tienen efecto alguno sobre la interrupción porque solo existe un nivel de prioridades.

Para habilitar una interrupción debe setearse el bit habilitador del periférico que se desee, ya sea en el PIE1 o en el PIE2, según el periférico. Una vez habilitado el periférico deben habilitarse el bit habilitador global de periféricos o PEIE, y el bit habilitador global del sistema de interrupciones o GIE, los cuales se encuentran en el registro INTCON.

### **Sistema de interrupciones en modo Prioridad**

En este modo existen dos vectores de prioridades, el vector de la alta prioridad en la dirección 0x00008 y el vector de baja prioridad en la dirección 0x00018.

Las interrupciones de los periféricos se pueden dividir en alta y baja prioridad, según como se configuren los bits que forman el registro IRP1 o IRP2, existe 1 bit por cada periférico. Si el bit es seteado, se le asignará la alta prioridad al periférico que se trate, caso contrario quedará marcado como de baja prioridad.

El sistema de prioridades nos permite que si esta cursando una interrupción de baja prioridad y es solicitado el procesamiento de una de alta prioridad, el sistema abandone momentáneamente la interrupción de baja prioridad y pase a procesar la de alta prioridad. Al finalizar esta, vuelve a la de baja prioridad y luego al programa principal.

Si por el contrario esta cursando una interrupción de alta prioridad y ocurre la petición de una de baja prioridad, el sistema continua el procesamiento de la de alta y luego que esta finaliza, pasa a procesar la de baja.

Sin embargo en este sistema las interrupciones de baja prioridad deben salvar los registros más importantes del núcleo, conocidos como “contexto”, de forma “manual” es decir que deben estar a cargo del programador, y el sistema de alta prioridad los almacena en los sombra. En realidad ambos sistemas usan por default los registros sombra pero la interrupción de alta prioridad pisa el contenido de los registros sombra, y es por ello que deben ser respaldados por el usuario contra la RAM. Pero nuevamente es el compilador C, el que se encarga de esta operación y es totalmente transparente al programador, por lo cual el mismo no debe preocuparse de esta operación.

## Tratamiento de las interrupciones en XC8

El tratamiento se ha simplificado respecto a los compiladores anteriores como el MPLAB C18. Para tratar las interrupciones el usuario debe indicar su función de interrupciones como **void** ya que no puede recibir parámetros ni puede devolver valores tampoco. Pero para usarla debe colocar un indicador específico **interrupt** o **interrupt low** según sea de alta o baja prioridad. De esta forma el encabezado para las rutinas de interrupción debe ser el siguiente:

### Alta prioridad:

```
void interrupt nombre_de_funcion(void)
{
    Sentencias ;
}
```

### Baja prioridad :

```
Void interrupt low_priority nombre_de_fucion(void)
{
    Sentencias ;
}
```

**NOTA MUY IMPORTANTE:** Queda a cargo del programador la puesta a cero de los bits de FLAG de cada interruptor en la medida que estos son atendidos. Es decir que el programador debe poner a cero el bit del registro PIR que corresponda con el periférico que se está atendiendo en la interrupción. Si esto no se hace, cuando se termine la rutina de interrupción, el programa volverá a re-ingresar sobre la misma.

## Trabajando con el Reloj

Ya que hemos sentado todas las bases necesarias podemos realizar nuestro reloj. Para ello usaremos como comentamos líneas atrás el timer 1 configurado como contador externo con su oscilador interno habilitado y con la interrupción activada por desborde del timer 1. En la rutina de interrupciones además de recargar el Timer uno con la cuenta 32768, actualizaremos los registros segundo, minuto y hora.

El hardware a usar será la placa STARTER KIT STUDENT para PIC16 y PIC18F, la cual viene actualmente con PIC18F46K20 y es para el cual prepararemos este ejemplo.

Nota: Si bien actualmente esta laca esta dotada de un PIC18F46K20, en sus versiones anteriores venia con PIC16F887 y PIC18F4620, por tanto este programa se podrá adaptar a dichos microcontroladores con ligeras reformas (fusibles de configuración y registros ANSEL y ANSELH).

A continuación presentamos el código del Reloj:

```
#include <xc.h> //libreria del procesador generica
#include <xlcd.h> //libreria del LCD
#include <stdio.h>      //libreria standar io
#include <stdlib.h>      //Libreria para conversión de datos
#include <delays.h>      //Libreria de delays

//Seteamos los fusibles de configuración
#pragma config FOSC = XT //Oscilador a cristal
#pragma config PWRT = ON
#pragma config WDTEN = OFF
#pragma config LVP = OFF
#pragma config IESO = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config XINST = OFF
#pragma config DEBUG = OFF

//Definiciones de etiquetas
#define BTN1 PORTAbits.RA0 //BOTON 1
#define BTN2 PORTAbits.RA1 //BOTON 2
#define BTN3 PORTAbits.RA2 //BOTON 3

//funciones prototipo
void Control(void);
void PrintLCD(void);
void Setup_timer1(void);
unsigned int ReadTimer1(void);
void Set_timer1(unsigned int timer1);

//Variables Globales
unsigned int Hora=0,Minuto=0,Segundo=0;
char String[20];

// Definimos la función timer_isr
void interrupt timer (void)
{
    PIR1bits.TMR1IF = 0;// limpiamos el flag de interrupciones
    TMR1H = 0b1000000;
    TMR1L = 0;
    Segundo++;
    if(Segundo==60)
    {
```

```

        Segundo=0;
        Minuto++;

    }

    if(Minuto==60)
    {
        Minuto=0;
        Hora++;
    }

    if(Hora==24)
    {
        Hora=0;
    }

}

// Rutinas de tiempo auxiliares para la libreria XLCD
void DelayFor18TCY(void)
{
    Delay10TCYx(10);
}

void DelayPORXLCD(void)
{
    Delay1KTCYx(20); //Delay de 15 ms
}

void DelayXLCD(void)
{
    Delay1KTCYx(40); //Delay de 20 ms
}

void SendCmdLCD(unsigned char CMD)//escribe un comando al LCD
{
    while(BusyXLCD());
    WriteCmdXLCD(CMD);
}

void main(void)
{
    ANSEL=0x00; //Puertos analógicos como digitales
    ANSELH=0X00;

    OpenXLCD(FOUR_BIT & LINES_5X7);      // Iniciamos LCD.-
    Setup_timer1();
    Set_timer1(32768);//Inicializamos el Timer 0
    T1CONbits.TMR1ON=1;//encendemos el timer
    PIE1bits.TMR1IE=1;//habilita interrupcion por Timer 1
}

```

```

INTCONbits.PEIE = 1;//habilita INT de perifericos
INTCONbits.GIE = 1;// Habilitamos Global de Interrupciones
while(1)
{
    SendCmdLCD(3);           //Borramos LCD
    putsXLCD("RELOJ");      // escribimos en la linea 1
    SetDDRamAddr(0x40);
    sprintf(String,"%02d:%02d:%02d",Hora,Minuto,Segundo);
    putsXLCD(String); // Escribimos el String en el LCD
    Delay1KTCYx(10);        // Esperamos 100 ms,reduce efecto parpadeo
}
void Setup_timer1(void)
{
    T1CONbits.TMR1ON=0;
    T1CONbits.TMR1CS=1;//Timer 1 clock externo
    T1CONbits.T1CKPS0=0;
    T1CONbits.T1CKPS1=0;
    T1CONbits.T1SYNC=0;//modo sincronizado
    T1CONbits.T1OSCEN=1;//oscilador activado
    T1CONbits.RD16=1;//lectura en 16 bits
}
unsigned int ReadTimer1(void)
{
    unsigned int Timer1;
    Timer1=TMR1H;//leemos los TMR0L y TMR0H y los concatenamos
    Timer1=Timer1<<8;
    Timer1=Timer1+TMR1L;
    return Timer1;
}
void Set_timer1(unsigned int timer1)
{
    int temporal;
    temporal=timer1;
    temporal=temporal>>8;
    TMR1H=temporal;
    TMR1L=timer1;
}

```

El programa utiliza las librerías para manejo de LCD que incorpora el compilador y que se denominan XLCD. Como dicha librería se encuentra configurada para el PORTB de fabrica, y en la placa STUDENT se usa el PORTD y el PORTE, habrá que editar el archivo XLCD.h y modificar los #define donde se encuentra la conexión, además habrá que incluir todos los archivos que se encuentran en la carpeta XLCD y adjuntarlos al proyecto dentro de la carpeta Source Files:

The screenshot shows a software interface with a menu bar at the top: File, Edit, View, Navigate, Source, Refactor, Run, Debug, Team, Tools, Window, Help. Below the menu is a toolbar with various icons. A status bar at the bottom shows "PC: 0x0" and "n ov z dc c : W:0x0 : bank 0".

The main window has two tabs: "Projects" and "Files". The "Projects" tab is selected, showing a tree view of the project structure:

- Reloj
  - Header Files
    - xlcd.h
  - Important Files
  - Linker Files
  - Source Files
    - busyxlcd.c
    - main.c
    - openxlcd.c
    - putrxlcd.c
    - putsxlcd.c
    - readaddr.c
    - readdata.c
    - setcgram.c
    - setddram.c
    - wcmdxlcd.c
    - writdata.c

The "Files" tab is also visible, showing the "main.c" file. The code editor displays the "xlcd.h" header file with the following content:

```
33 /* DATA_PORT defines the port to which the LCD data lines are connected.
34 #define DATA_PORT PORTD
35 #define TRIS_DATA_PORT TRISD
36
37 /* CTRL_PORT defines the port where the control lines are connected.
38 * These are just samples, change to match your application.
39 */
40 #define RW_PIN LATEBits.LATE0 /* PORT for RW */
41 #define TRIS_RW TRISEbits.TRISE0 /* TRIS for RW */
42
43 #define RS_PIN LATEBits.LATE1 /* PORT for RS */
44 #define TRIS_RS TRISEbits.TRISE1 /* TRIS for RS */
45
46 #define E_PIN LATEBits.LATE2 /* PORT for D */
47 #define TRIS_E TRISEbits.TRISE2 /* TRIS for E */
48
```

En la figura vemos los archivos involucrados en el proyecto y el XLCD.h modificado

## **PROGRAMA 9: TERMÓMETRO CON PIC18F46K20**

Otro periférico muy importante en los microcontroladores es el conversor A/D, mediante el cual podemos medir cualquier señal analógica proveniente de sensores proporcionales. Casi todos los microcontroladores actuales incorporan conversores A/D, sin embargo son los microcontroladores de Microchip los que se destacan del resto por incorporar un conversor A/D de 10 bits en la mayoría de los chips y de 12 bits en algunos micros de 8bits y en todos los dsPIC.

La cantidad de bits en la palabra de resultado del proceso de conversión, que genera el microcontrolador, es muy importante, ya que de ello depende la resolución y precisión del conversor. Cuantos más bits tenga, menor será el error de conversión.

Al tener 10 bits el ADC de un microcontrolador PIC, el escalón de resolución, si la tensión de referencia del microcontrolador es de 5Volt, será de 4.88mv aproximadamente.

En este nuevo programa veremos como usar el conversor A/D para leer por ejemplo la tensión aplicada a la entrada del conversor. Antes de ello haremos una breve descripción de las características del ADC que vienen integrado en los PIC18F46K20.

### **El Conversor A/D**

El conversor integrado dentro del PIC18F así como en los PIC16, es del tipo SAR o Registro de Aproximación sucesiva. Tiene un total de 14 canales analógicos denominados AN0 a AN13. Todos estos canales aceptan una tensión de entrada de 0 a 3,3V.

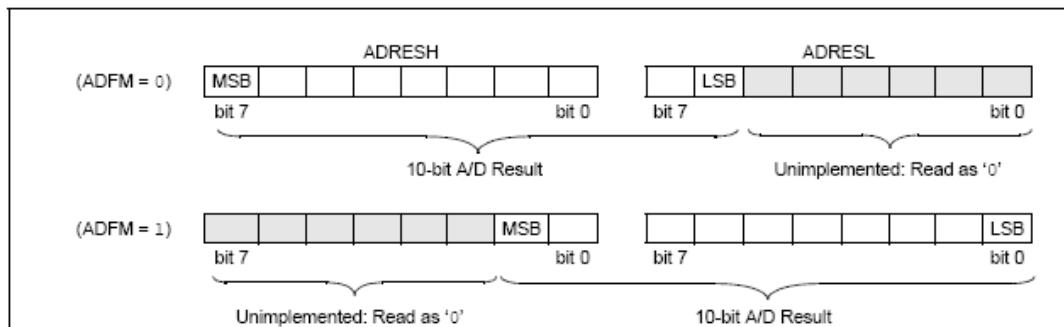
Las entradas analógicas AN0 a AN13, están multiplexadas con otras funciones. Cuando los PIC arrancan, dichos pines automáticamente son configurados como entradas analógicas. Sin embargo si uno quiere configurar que determinados pines sean analógicos y otros sean digitales, dicha configuración se realizará activando o clareando los bits respectivos de los registros ANSEL y ANSELH. Cuando se pone un bit en uno, dicho canal funciona como digital, mientras que si es puesto en cero, funciona como analógico.

Para que funcione el conversor además de configurarse que pines son analógicos debe también seleccionarse un clock que excite al conversor, el cual puede derivarse desde la misma frecuencia interna de operación Fosc/4, la cual pasa por un divisor de frecuencia.

Dicho divisor se puede configurar con los bits de control ADCS0 a ADCS2, los cuales se encuentran dentro del registro de control ADCON2. Otra de las cosas que debe configurarse es el tiempo de adquisición, que es el tiempo de espera que se realizará antes de iniciar la conversión. Este tiempo es fundamental para esperar que el capacitor de HOLD se cargue totalmente con la muestra de la señal a medir, para evitar que se produzcan errores de conversión.

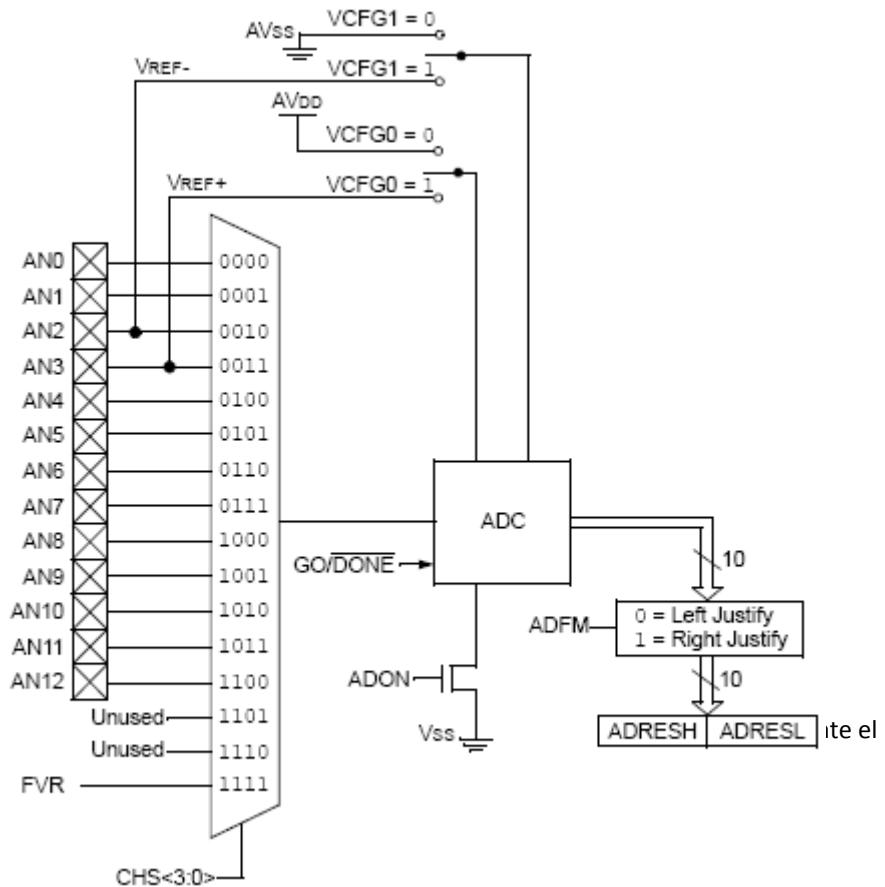
Este capacitor funciona como una memoria, desde el mismo se alimenta al proceso de conversión. Si el mismo no esta cargado totalmente, el tiempo de la muestra ha sido insuficiente y se genera un error en la conversión. Dicho tiempo es seleccionable mediante los bits ADCS0 a ADCS2.

Como el conversor genera un resultado en 10 bits, el mismo se almacena en 2 registros de 8 bits, denominados ADRESH y ADRESL. Según como se configure el bit ADFM, el resultado se ajustará a la izquierda o a la derecha:



Esto permite realizar una lectura en 10 bits (configurando el ajuste a la derecha) o en 8 bits(configurando el ajuste a la izquierda), este último conocido como lectura rápida ya que solo se necesita leer el registro ADRESL. Cuando se leen los 10 bits hay que concatenar el resultado de ambos registros dentro de una variable de 16 bits.

La estructura del conversor A/D se simplifica en el siguiente esquema:

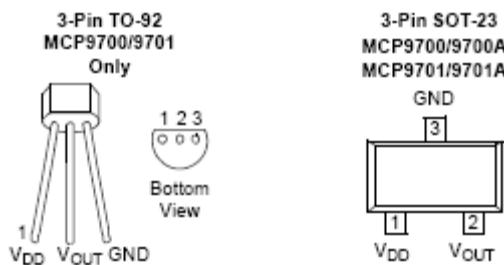


Para iniciar la conversión el hardware pone este pin en el estado finalizado.

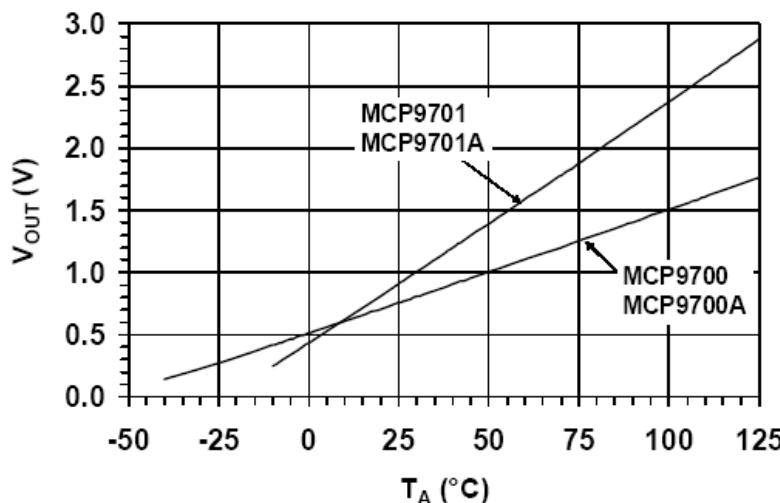
## El Termómetro:

Para implementar el termómetro usaremos el sensor de Microchip MCP9700A, el cual tiene un coeficiente de temperatura de 10mv/°C. Este sensor esta preparado para trabajar en el rango de -40 a 125°C, y dentro de este rango nos entregará una tensión que variará desde 0,1V (-40°C) hasta 1,75V (125°C). Cuando el sensor se encuentra a 0°C, nos entrega una tensión de 0,5V. Esto se tiene que tomar en cuenta porque dicho voltaje se debe restar de la tensión total que nos entrega el sensor (es su offset).

El sensor es lineal dentro de todo su rango, y tiene una precisión de 2°C aproximadamente, lo cual es aceptable para cualquier aplicación casera. En la siguiente figura podemos apreciar sus encapsulados más típicas con el respectivo PIN-OUT:



Y su recta de transferencia o curva característica de salida, donde se ha graficado la tensión que nos entrega en función de la temperatura que sensa:



En el siguiente listado les presentamos el código completo del termómetro. Para realizar el mismo hemos utilizado las librerías precompiladas que vienen en el compilador para manejar el conversor A/D:

```
#include <xc.h>

//librerias de delay desarrolladas por MCHP.
#include <delays.h>
//libreria para manejar el LCD
#include <plib\xlcd.h>
//libreria standard IO
#include <stdio.h>
//libreria para la conversion de datos
#include <stdlib.h>
//libreria para manejar el ADC
#include <plib\adc.h>

//Seteamos los fusibles de configuración
#pragma config FOSC = XT //Oscilador Interno
#pragma config PWRT = ON
#pragma config WDTEN = OFF
#pragma config LVP = OFF
#pragma config IESO = OFF
#pragma config LPT1OSC = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config XINST = OFF
#pragma config DEBUG = OFF

//Definiciones de etiquetas
#define BTN1 PORTAbits.RA4
#define BTN2 PORTBbits.RB0
#define VREFmax 5.0
#define RESOLUCION 1024.0
#define Voffset 0.5//500mv a 0°C
#define CoefTerm 0.01//rata de cambio del sensor 10mv/°C

//definimos variables globales
unsigned char string[10];
unsigned int valor=0;
float volt=0, grados=0;
//funciones prototipo
```

```

void PrintLCD(void);
float ConvertVolt(unsigned int adc);
float ConvertGrados(float volt);

// Rutinas de tiempo auxiliares para la libreria XLCD
void DelayFor18TCY(void)
{
    Delay10TCYx(10);
}

void DelayPORXLCD(void)
{
    Delay1KTCYx(20); //Delay de 15 ms
}

void DelayXLCD(void)
{
    Delay1KTCYx(40); //Delay de 20 ms
}

void SendCmdLCD(unsigned char CMD)//escribe un comando al LCD
{
    while(BusyXLCD());
    WriteCmdXLCD(CMD);
}

//Rutina Principal
void main (void)
{
    // Inicializamos los registros OSCCON,TRISB y PORTB.
    OSCCONbits.IRCF0=0;//seteamos para 4 MHz de salida
    OSCCONbits.IRCF1=1;
    OSCCONbits.IRCF2=1;
    ADCON1=0xD; //seteamos AN0 y AN1 como analogicas

    // Inicializamos el LCD
    OpenXLCD(FOUR_BIT & LINES_5X7 );

    // Inicializamos el ADC
    OpenADC( ADC_FOSC_64 & ADC_RIGHT_JUST,
             ADC_CH1 & ADC_INT_OFF & ADC_REF_VDD_VSS,
             ADC_0ANA|ADC_1ANA );
}

```

```

//Bucle que hace destellar el Led
while(1)
{
    ConvertADC();
    while(BusyADC());
    valor=ReadADC();
    volt=ConvertVolt(valor);
    grados=ConvertGrados(volt);
    PrintLCD(); //imprime en pantalla el contador y el estado
}
}

//Imprime en el display
void PrintLCD(void)
{
    SetDDDRamAddr(0x0);
    putsXLCD("ADC PIC18F46K20");
    sprintf(string,"AN1=%1.2fV..%2.1fG ",volt,grados);
    SetDDDRamAddr(0x40);
    putsXLCD(string);
}

// Convierte a Volt
float ConvertVolt(unsigned int adc)
{
    float voltaje;
    voltaje=(float)((VREFmax/RESOLUCION)*adc);
    return voltaje;
}

//Convierte a Grados
float ConvertGrados(float volt)
{
    float temperatura;
    temperatura=(float)((volt-Voffset)/CoefTerm);
    return temperatura;
}

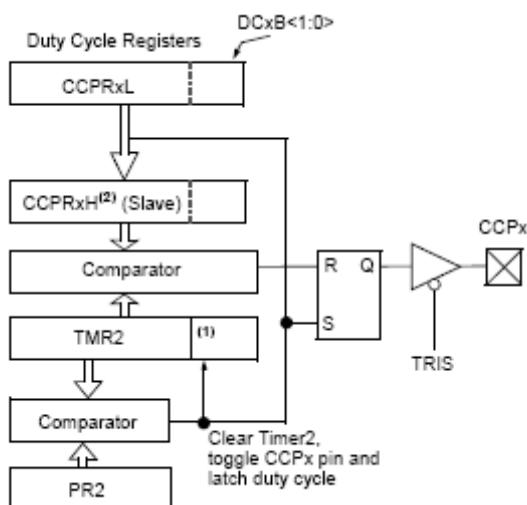
```

## PROGRAMA 10: USANDO EL MÓDULO PWM

El módulo PWM o Modulador de Ancho de Pulso, es una de las 3 formas en las cuales se puede configurar el funcionamiento del modulo CCP o Capturador-Comparador-PWM. Este módulo nos permite Capturar un evento externo, para medir por ejemplo el período de una señal, generar una señal de frecuencia variable o generar una señal en la cual podamos alterar la relación entre el Ton y el Toff, manteniendo fija la frecuencia, lo cual se conoce generalmente como PWM o Modulador de Ancho de Pulso.

### El Modulo PWM

El PWM utiliza el Timer 2 como motor del mismo, por lo cual debe configurarse el Timer 2 para que determine la frecuencia de la modulación de la señal PWM. Por otra parte, la relación Ton/T o Duty Cycle es controlada por los registros CCP. El PWM en modo Single puede usar el módulo CCP1 o el CCP2. En la siguiente figura podemos ver el CCP configurado para funcionar como PWM:



La resolución del PWM varía con la frecuencia, cuanto mas alta es la frecuencia, menor será la resolución, esto debe tomarse muy en cuenta para la aplicación del PWM. Por lo general las resoluciones más altas se obtienen en frecuencias menores a 1Khz. Para obtener resoluciones significativas el microcontrolador debe trabajar a frecuencias muy altas. En la siguiente tabla podemos ver lo expuesto:

TABLE 11-4: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS AT 40 MHz

PWM Frequency	2.44 kHz	9.77 kHz	39.06 kHz	156.25 kHz	312.50 kHz	416.67 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	FFh	FFh	FFh	3Fh	1Fh	17h
Maximum Resolution (bits)	10	10	10	8	7	6.58

TABLE 11-5: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 20 MHz)

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	6.6

TABLE 11-6: EXAMPLE PWM FREQUENCIES AND RESOLUTIONS (Fosc = 8 MHz)

PWM Frequency	1.22 kHz	4.90 kHz	19.61 kHz	76.92 kHz	153.85 kHz	200.0 kHz
Timer Prescale (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0x65	0x65	0x65	0x19	0x0C	0x09
Maximum Resolution (bits)	8	8	8	6	5	5

El valor que debe cargarse en PR2 es utilizado para obtener el PWM máximo (95%). En siguiente ecuación podeos ver la relación de la frecuencia con el TMR2:

$$\text{PWM Period} = [(PR2 + 1) \cdot 4 \cdot TOSC \cdot (TMR2 Prescale Value)]$$

**Note:** TOSC = 1/FOSC.

Y en la siguiente su relación con la resolución del PWM:

$$\text{Resolution} = \frac{\log[4(PR2 + 1)]}{\log(2)} \text{ bits}$$

## Trabajando con el PWM:

En el siguiente código hemos realizado un control PWM, para lo cual hemos usado las librerías precompiladas que trae el compilador para el control de sus periféricos internos para PIC18F. En este caso con ellas configuramos el Timer 2 y el módulo CCP para que trabaje como PWM. El porcentaje de PWM lo podremos ver en el en LCD. El PWM se controlará desde un Preset conectado en RA0 y la variación del PWM se podrá apreciar sobre el LED que viene en la placa STUDENT. Si se colca un Oscloscopio en el punto de prueba PWM se podrá apreciar la variación del mismo.

```
#include <xc.h>

//Incluimos las librerias desarrolladas por MCHP.
#include <delays.h>
#include <plib\xlcd.h>
#include <stdio.h>
#include <stdlib.h>
#include <plib\adc.h>
#include <plib\ pwm.h>
#include <plib\timers.h>

//Seteamos los fusibles de configuración
#pragma config FOSC = XT //Oscilador Interno
#pragma config PWRT = ON
#pragma config WDTEN = OFF
#pragma config LVP = OFF
#pragma config IESO = OFF
#pragma config LPT1OSC = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config XINST = OFF
#pragma config DEBUG = OFF

//Definiciones de etiquetas
#define BTN1 PORTAbits.RA4
#define BTN2 PORTBbits.RB0
#define VREFmax 5.0
#define RESOLUCION 255.0
```

```

//definimos variables globales
unsigned char string[10];
unsigned int valor=0;
float DutyCicle=0;
//funciones prototipo

void PrintLCD(void);
float ConvertDC(unsigned int adc);

// Rutinas de tiempo auxiliares para la libreria XLCD
void DelayFor18TCY(void)
{
    Delay10TCYx(10);
}

void DelayPORXLCD(void)
{
    Delay1KTCYx(20); //Delay de 15 ms
}

void DelayXLCD(void)
{
    Delay1KTCYx(40); //Delay de 20 ms
}

void SendCmdLCD(unsigned char CMD)//escribe un comando al LCD
{
    while(BusyXLCD());
    WriteCmdXLCD(CMD);
}

//Rutina Principal
void main (void)
{

// Inicializamos los registros OSCCON,TRISB y PORTB.

OSCCONbits.IRCF0=0;//seteamos para 4 MHz de salida
OSCCONbits.IRCF1=1;
OSCCONbits.IRCF2=1;
ADCON1=0x0E; //seteamos AN0 como analogica
TRISB = 0X01; //RB0 como entrada digital
PORTB = 0;
TRISCbits.TRISC2=0;
TRISDbits.TRISD7=0;
LATDbits.LATD7=1;// enciendo el VCC del LCD

```

```

TRISA =255;

OpenXLCD(FOUR_BIT & LINES_5X7 );

//configuramos el funcionamiento del Timer 2
OpenTimer2(TIMER_INT_OFF & T2_PS_1_4 & T2_POST_1_1);

//configuramos el valor del PR2
OpenPWM1(240);//se carga en PR2

//Configuramos el funcionamiento del CCP como PWM
SetOutputPWM1(SINGLE_OUT,PWM_MODE_1);

//Configuramos el funcionamiento del ADC
OpenADC( ADC_FOSC_64 & ADC_RIGHT JUST,
         ADC_CH0 & ADC_INT_OFF & ADC_REF_VDD_VSS,
         ADC_OANA );

//Bucle que hace destellar el Led
while(1)
{
    ConvertADC();
    while(BusyADC());
    valor=(ReadADC()>>2);//8 bits de resolucion
    SetDCPWM1(valor<<2);
    DutyCicle=ConvertDC(valor);
    PrintLCD();//imprime en pantalla el contador y el estado
}
}

void PrintLCD(void)
{
    SetDDRamAddr(0x0);
    putrsXLCD("PWM PIC18F4520");
    sprintf(string,"AN0=%04u..DC:%.2f",valor,DutyCicle);
    SetDDRamAddr(0x40);
    putsXLCD(string);
}

float ConvertDC(unsigned int adc)
{
    float Duty;
    Duty=(float)((adc/RESOLUCION)*100.0);
    return Duty;
}

```

## **Programa 11: Trabajando con la USART**

La USART es un módulo que le permite comunicarse al microcontrolador en forma serie. Este módulo puede ser configurado para realizar una comunicación sincrónica, por ejemplo para leer un viejo teclado de PC tipo PS2, o de forma asincrónica para implementar una comunicación contra una PC por medio de una interfaz RS232 o como actualmente se hace por USB usando para ello un transceptor USB-USART.

En nuestro caso usaremos este último método ya que nos interesa enviar datos hacia la PC usando el USB, tan popular hoy en día.

Desde hace ya unos años Microchip simplificó la conexión USB desarrollando un CHIP denominado MCP2200, el cual tiene un controlador USB embebido.

El MCP2200, trabaja como un “puente USB-USART” permitiéndonos conectar cualquier microcontrolador con USART a USB2.0

En la placa STUDENT ya viene conectado el MCP2200 a la USART del PIC18F46K20. En el siguiente código realizaremos un control ON/OFF de los LEDs conectado al puerto RB0 y un menú desplegable por el programa HYPERterminal que viene en WINDOWS XP.

Mediante el HYPERterminal podremos controlar remotamente nuestra laca conectandola al USB, con un BAUD RATE de 9600.

Cuando usted conecte la placa al USB de su PC, la computadora le pedirá el driver respectivo del MCP2200, el cual puede descargar desde la WEB de Microchip.

Como en los casos anteriores usamos las librerías precompiladas por microchip para el control de la USART.

A continuación les presentamos el código completo de la USART:

```
#include <xc.h>
//Incluimos las librerías desarrolladas por MCHP.
#include <delays.h>
#include <stdlib.h>
#include <stdio.h>
#include <uart.h>
#include "EMU_terminal.h"

//Seteamos los fusibles de configuración
#pragma config FOSC = XC
#pragma config PWRT = ON
#pragma config WDTEN = OFF
#pragma config LVP = OFF
#pragma config BOR = ON
#pragma config IESO = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config XINST = OFF
#pragma config DEBUG = OFF

void main(void)
{
    unsigned char data;

    ANSEL=0x00;
    ANSELH=0X00;
    TRISB = 0;
    PORTB = 0;
    TRISA =255;
    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_OFF & USART_SYNCH_MODE &
    USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH,25);
    TERM_CURSOR_X_Y(1,1); // posición x, y en hyper terminal.
    printf("*****");
    TERM_CURSOR_X_Y(2,1);
    printf("*      MENU de OPCIONES      *");
    TERM_CURSOR_X_Y(3,1);
    printf("*****");
    TERM_CURSOR_X_Y(4,1);
    printf(" PULSE: ");
    TERM_CURSOR_X_Y(5,1);
    printf(" -->[0]....TOGGLE RB0");
    TERM_CURSOR_X_Y(6,1);
    printf(" -->[1]....TOGGLE RB1");
    while(1)
```

```

    {
        while(!DataRdyUSART( )); // verifica si llego un dato
        data=getcUSART();      //captura dato recibido
        switch(data)
        {
            case '0':
                PORTBbits.RB0=!PORTBbits.RB0;
                break;
            case '1':
                PORTBbits.RB1=!PORTBbits.RB1;
                break;
        }
    }
}

```

Librería EmuTerminal.h :

// Códigos útiles para la terminal, permite posicionar el texto en coordenadas X, Y.

```

#define ESC 0x1B
#define TERM_reset         printf("%c[0m",ESC)
#define TERM_clear          printf("%c[2J",ESC)
#define TERM_HOME           printf("%c[H",ESC)

#define TERM_FORE_bright    printf("%c[1m",ESC)
#define TERM_FORE_underscore printf("%c[4m",ESC)
#define TERM_FORE_blink      printf("%c[5m",ESC)
#define TERM_FORE_reverse    printf("%c[7m",ESC)

#define TERM_CURSOR_X_Y(X,Y) printf("%c[%u;%uf",0x1B,X,Y)

#define TERM_BACK_black     printf("%c[40m",ESC)
#define TERM_BACK_red        printf("%c[41m",ESC)
#define TERM_BACK_green       printf("%c[42m",ESC)
#define TERM_BACK_yellow      printf("%c[43m",ESC)
#define TERM_BACK_blue        printf("%c[44m",ESC)
#define TERM_BACK_purple      printf("%c[45m",ESC)
#define TERM_BACK_cyan        printf("%c[46m",ESC)
#define TERM_BACK_white       printf("%c[47m",ESC)

```

```
#define TERM_FORE_black      printf("%c[30m",ESC)
#define TERM_FORE_red         printf("%c[31m",ESC)
#define TERM_FORE_green        printf("%c[32m",ESC)
#define TERM_FORE_yellow       printf("%c[33m",ESC)
#define TERM_FORE_blue         printf("%c[34m",ESC)
#define TERM_FORE_purple       printf("%c[35m",ESC)
#define TERM_FORE_cyan         printf("%c[36m",ESC)
#define TERM_FORE_white        printf("%c[37m",ESC)
#define TERM_FORE_reverse      printf("%c[7m",ESC)
```

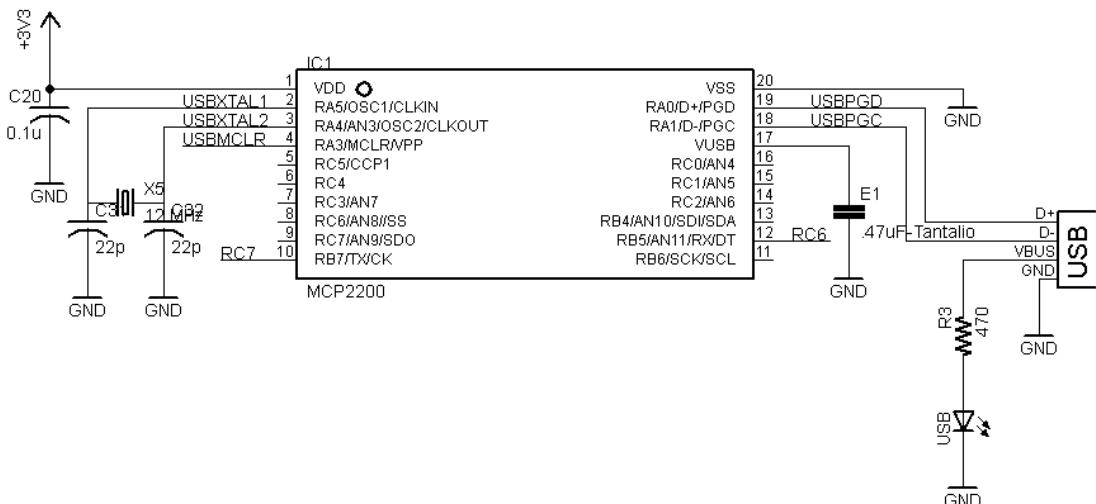
## Conversor USART - USB MCP 2200

El MCP2200 es un nuevo dispositivo de Microchip que permite realizar la conversión de los protocolos de comunicación serie USART-USB. Solo requiere la conexión de los Pines Tx y Rx al microcontrolador.

Estas son las características de este dispositivo.

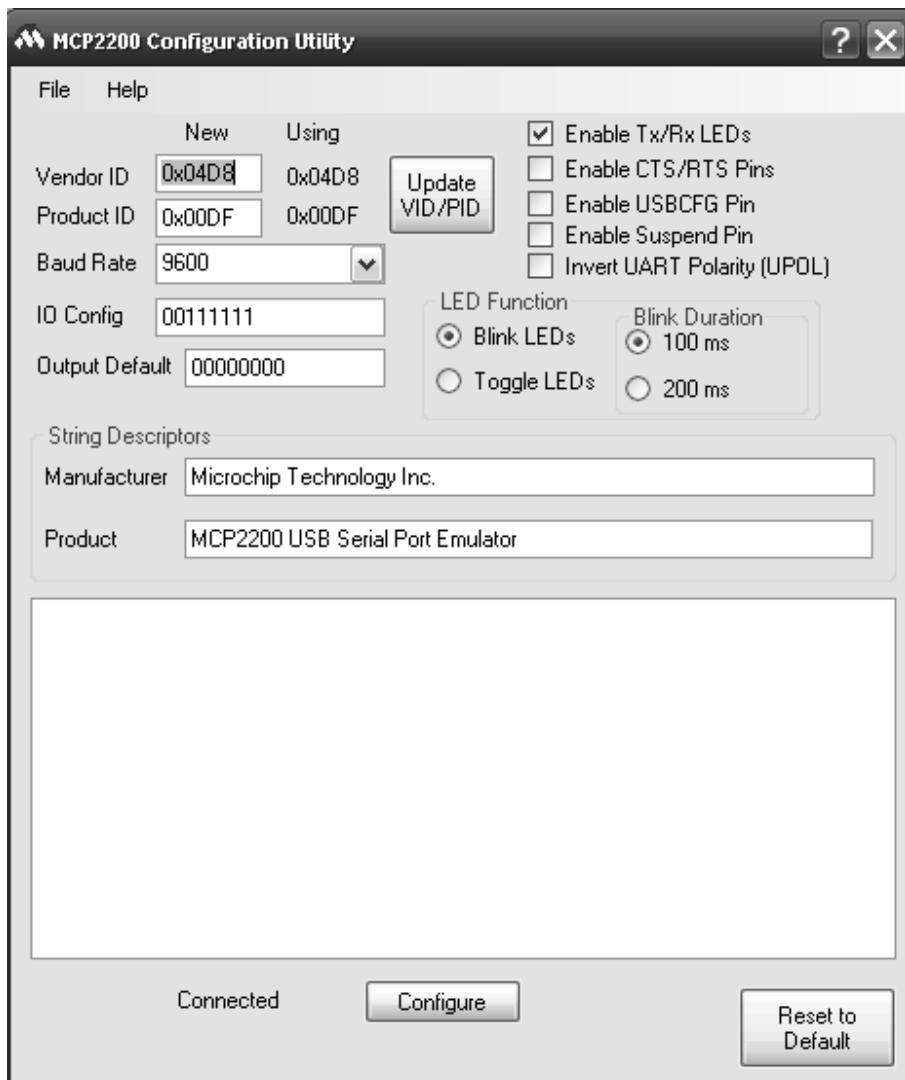
- **Adaptador de Interfaz USB a USART fabricado por Microchip**
  - **Soporta un BAUD RATE de 300bps a 1000Kbps**
  - **Control de flujo por Hardware**
  - **Incorpora hasta 8 puertos I/O**
  - **EEPROM de 256Bytes**
  - **Salidas para LED Tx y Rx**
  - **Opera desde 3 a 5.5V**
  - **Soporta USB Full Speed (12Mbs)**
  - **Buffer de 128 bytes para manejar el procesamiento de datos de la USART a cualquier BAUD RATE**

Círcuito del MCP2200 que realiza la interfaz entre los mensajes de la USART del PIC por medio de TX (RC7) y RX (RC6) y el puerto USB.



## MCP2200 Configuration Utility

Este software nos permite configurar diferentes parámetros del dispositivo MCP2200 como la frecuencia de comunicación USART, habilitar el control de flujo por Hardware, habilitar LEDs indicadores de una transmisión o recepción de datos, establecer el VID y el PID del dispositivo como así también los String de producto y del fabricante para personalizarlo a nuestra aplicación.



Ventana de la utilidad para configurar el MCP2200

## **INSTALACIÓN DEL DRIVER PARA EL MCP2200 SERIAL EMULATOR**

Pasos a realizar antes de enchufar la placa al USB.

- 1) Dirigirse a la página de microchip.

<http://www.microchip.com/mcp2200>

Donde se encuentra la descripción del dispositivo MCP2200.

En la zona de descargas seleccionar el archivo *MCP2200 Drivers and Utilities*

Descargarlo y descomprimirlo en la PC.

Directorio del archivo descargado

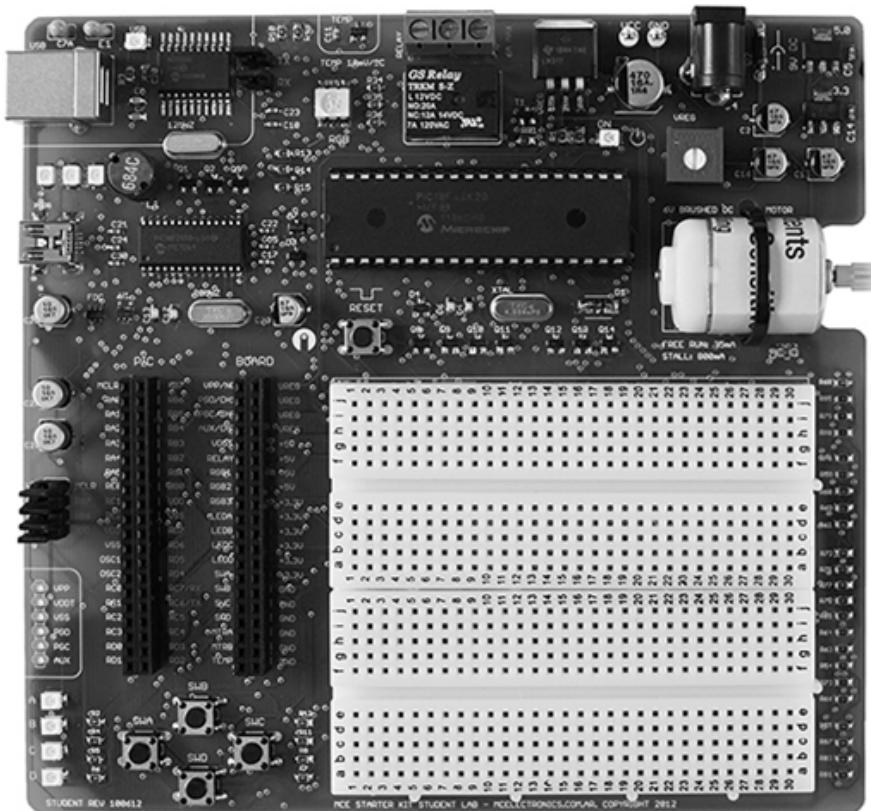
- Configuration Utility Rev 1.10
- MCP2200 Win INF
- Production Configuration Utility Rev 1.10
- User DLL

2) Enchufar la placa al USB de la PC y seleccionar la instalación del driver de manera manual. El driver se encuentra en la carpeta MCP2200 Win INF, se debe seleccionar esta en el momento de la instalación.

3) Esperar a que finalice la instalación y probar la placa desde el Hypertermina. Ir al administrador de dispositivo de Windows para ver el número de puerto asignado al COM virtual.

## LABORATORIO DE PRÁCTICAS

Aquellos alumnos que prefieran montar sus propios circuitos pueden optar por la placa MCE Starter KIT Student LAB. Esta placa contiene un protoboard e incluye varios componentes para realizar las prácticas. Además está provista de un programador con analizador lógico para monitorear las señales.



### MCE Starter KIT Student LAB

Para descargar el manual de usuario u obtener más información por favor visite el catálogo online: [www.mcelectronics.com.ar/desarrollos](http://www.mcelectronics.com.ar/desarrollos)



# **ARDUINO CON PIC**



## **QUÉ ES ARDUINO ?**

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.

Arduino puede monitorear el entorno y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el Arduino Programming Language (basado en Wiring) y el Arduino Development Environment (basado en Processing).

Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con Flash, Processing, MaxMSP, etc.).

Las placas se pueden ensamblar a mano o encargarlas pre ensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades. Arduino recibió una mención honorífica en la sección Digital Communities del Ars Electronica Prix en 2006.

## **POR QUÉ ARDUINO ?**

Hay muchos otros microcontroladores y plataformas microcontroladoras disponibles para computación física. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, y muchas otras ofertas de funcionalidad similar. Todas estas herramientas toman los desordenados detalles de la programación de microcontrolador y la encierran en un paquete fácil de usar. Arduino también simplifica el proceso de trabajo con microcontroladores, pero ofrece algunas ventajas para profesores, estudiantes y aficionados interesados sobre otros sistemas:

Económico:

Las placas Arduino son relativamente económicas comparadas con otras plataformas microcontroladas.

Multiplataforma:

El software de Arduino se ejecuta en sistemas operativos Windows, Macintosh OSX y GNU/Linux. La mayoría de los sistemas microcontroladores están limitados a Windows.

Entorno de programación simple y claro:

El entorno de programación de Arduino es fácil de usar para principiantes, pero suficientemente flexible para que usuarios avanzados puedan aprovecharlo también. Para profesores, está convenientemente basado en el entorno de programación Processing, de manera que estudiantes aprendiendo a programar en ese entorno estarán familiarizados con el aspecto y la imagen de Arduino.

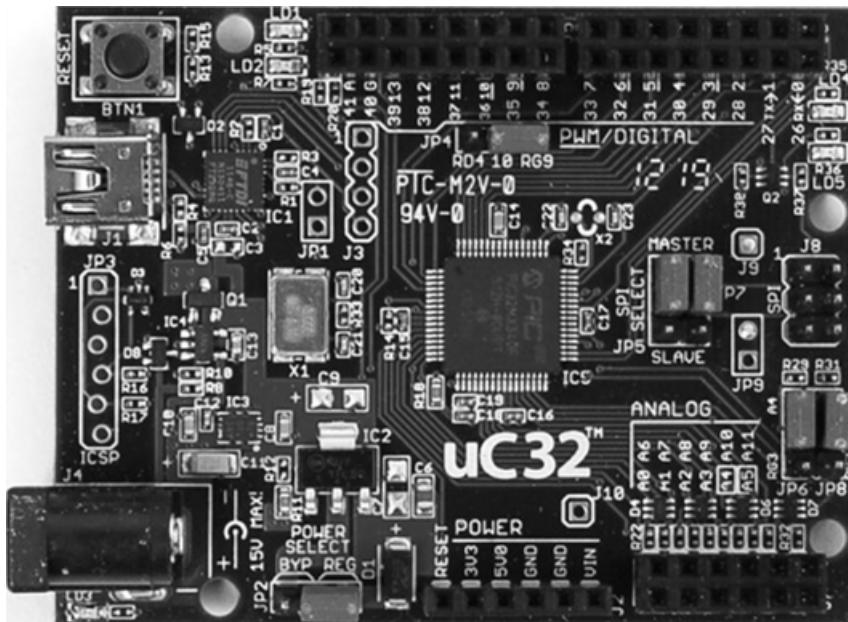
Código abierto y software extensible:

El software Arduino está publicado como herramientas de código abierto, disponible para extensión por programadores experimentados. El lenguaje puede ser expandido mediante librerías C++.

Código abierto y hardware extensible: El Arduino está basado en microcontroladores PIC32 de Microchip. Los planos para los módulos están publicados bajo licencia Creative Commons, por lo que diseñadores experimentados de circuitos pueden hacer su propia versión del módulo, extendiéndolo y mejorándolo. Incluso usuarios relativamente inexpertos pueden construir la versión de la placa del módulo para entender cómo funciona y ahorrar dinero.

## PLATAFORMA CHIPKITUNO32

La empresa DiGILENT desarrolló toda una plataforma de placas Arduino compatibles basadas en un microcontrolador PIC 32. De toda su plataforma hemos elegido la placa CHIPKIT Uno32 la cual tiene el factor de forma ARDUINO 1.

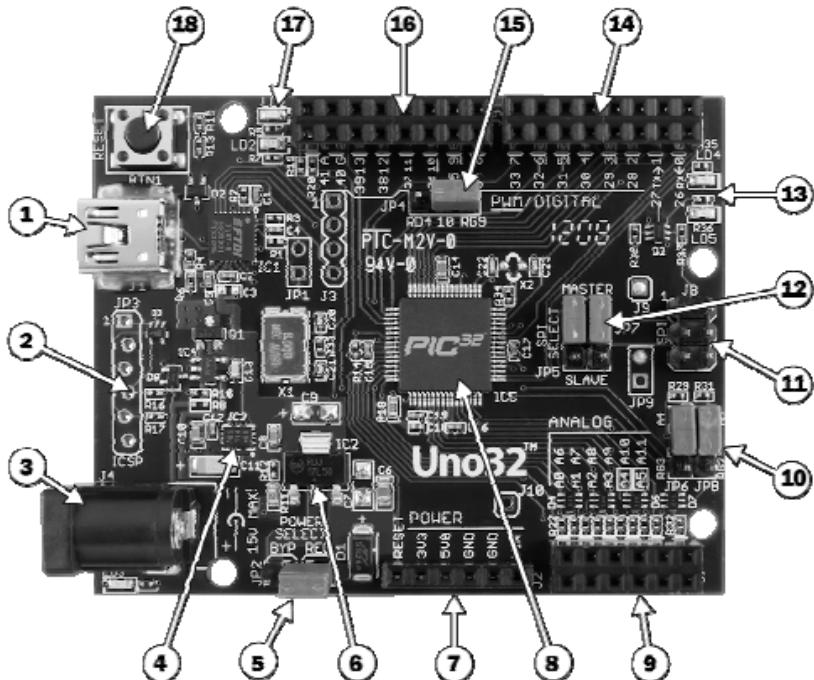


Placa chipKIT UNO PIC32

La placa presenta las siguientes características:

- Equipada con un PIC32MX320F128H de Microchip®, el cual es un microcontrolador de 32 bits (80 Mhz , 80MIPS,128K Flash de programa y 16K SRAM)
- Compatible con cualquier código Arduino existente
- Factor de forma Arduino Uno
- Compatible con cualquier shields Arduino
- 42 Puertos I/O
- 2 LEDs
- Conexión con PC vía un conector MINI-USB tipo A
- 12 Canales Analógicos
- Voltaje de Operación 3.3V
- Frecuencia de Operación 80MHz
- Corriente de consumo 75ma
- Voltaje de entrada recomendado 7 a 15V
- Voltaje máximo de entrada 20V
- Rango de voltaje en las entradas anaógicas: 0 a 3,3V
- Corriente Máxima por PIN +/- 18 ma

Detalle las prestaciones que nos presenta la placa ARDUINO compatible para PIC32:



## **1. Conector Mini USB para el Conversor USART-USB FTD232**

Este puerto de comunicaciones le permite conectar la placa a la PC para que de esta forma el entorno MPIDE pueda “hablar” con la placa CHIPKIT Uno32. También es usado para alimentar la placa sin necesidad de una fuente externa. Cuando uno descarga el firmware a la placa entonces también puede usar el puerto para comunicarse con una aplicación vía una emulación de puerto COM.

## **2. JP3 – Conector ICSP para programadores debuggers de Microchip**

Este conector es usado por los programadores de Microchip (PICKIT2, PICKIT3, ICD2 e ICD3). Esto le permite a la placa CHIPKIT Uno32 board trabajar como una placa de entrenamiento tradicional con el entorno de desarrollo de Microchip MPLAB® IDE.

## **3. J4 – Conector de Alimentación externo**

La placa CHIPKIT Uno32 puede ser alimentada desde una fuente externa, tomando en cuenta que el centro del conector debe estar conectado al positivo de la fuente.

Cuando se trabaja con fuente externa, la tensión de alimentación debe encontrarse entre 7V y 15V. Si bien la placa soporta una mayor tensión de entrada, no se recomienda el uso de la misma para mantener baja la disipación de los reguladores de tensión de entrada

## **4. Regulador de 3.3V**

Regulador de tensión de 3.3V para alimentar el microcontrolador PIC32 y generar la fuente de 3.3V para circuitos auxiliares. Esta fuente puede proveer hasta 500mA de corriente

## **5. JP2 – Jumper Selector de Power**

Este Jumper permite usar o bypassar el regulador de 5V que viene integrado en la placa. Cuando se encuentra en la posición REG, el conector de entrada de fuente envía energía a través del regulador de 5V. En la posición BYP se bypassa el regulador de 5V. En este caso (posición BYP) el máximo voltaje de entrada externo no debe exceder los 6V.

## **6. Fuente de 5V**

Este es el regulador integrado en la placa para reducir la tensión de entrada, proveniente del conector de fuente externa a 5V. Desde aquí luego se alimenta al regulador de 3.3V y al conector de expansión para alimentar otros PCBs externos. Este regulador puede proveer hasta una corriente de 800ma.

## **7. J2 – Conector de Alimentación para PCBs externos**

Este conector provee la alimentación para las placas externas que expanden las capacidades I/O de la placa CHIPKIT Uno32.

## 8. Microcontrolador PIC32

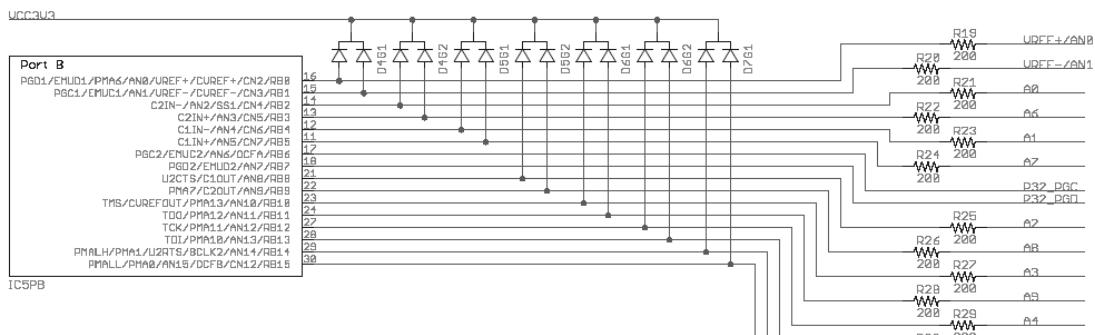
El microcontrolador principal de la placa es el PIC32MX320F128H.

## 9. J7 – Conector de Señales Analógicas

Este conector provee entrada para aplicar señales analógicas a los puertos analógicos del PIC32. En total hay 12 canales analógicos, denominados A0 a A11 por el compilador Arduino y que están cableados a los canales AN0-AN11 del PIC32.

Todas las entradas analógicas se encuentran protegidas por una “protección integral de compuerta” emplazada con 2 diodos de enclavamiento y una resistencia. De esta forma se evita destruir un canal analógico por aplicar un voltaje de entrada incorrecto.

La máxima tensión que se puede aplicar en las entradas analógicas es 3V. La tensión que puede leer cada canal debe ser siempre positiva respecto a masa y encontrarse en el rango 0 a 3V. No se admiten tensiones negativas respecto de cero.



#### **10. Jumpers selectores de señales JP6/JP7 – A4/A5**

Estos Jumper son para intercambiar el funcionamiento de los pines 9 y 11 de ser canales analógicos A4 y A5 a transformarse en los canales de comunicaciones I2C SDA y SCL.

#### **11. J8 – Conector para expandir puerto SPI**

Este conector provee acceso a las señales del SPI. Este generalmente es usado cuando se quieren conectar dispositivos SPI externos a la placa CHIPKIT Uno32, y que se encuentran en alguna placa de expansión

## **12. JP5/JP7 – SPI Master/Slave Select Jumpers**

Este Jumper es usado para cambiar la funcionalidad de las señales SPI para usar dicha interfaz en la placa CHIPKIT Uno32 como un SPI Maestro o Esclavo. Ambos JUMPERS deben ser cambiados.

Por favor conecte los Jumpers cortocircuitando el bloque correspondiente según el modo en el que va a trabajar el PIC32. Normalmente estos Jumpers vienen en la posición MAESTRO.

## **13. LEDs de usuario**

Dos LEDs han sido conectados a las señales digitales pines 13 y 43.

## **14. J6 – Conector de Señales Digitales**

Este conector provee acceso a los pines I/O digitales del microcontrolador.

## **15. JP4 – Jumper de señal en el Pin 10**

Este Jumper es usado para cambiar la funcionalidad del PIN 5 (señal digital 10) entre la salida del modulo PWM o la operacion en modo SPI. El jumper es conectado en el puerto RD4 del microcontrolador para tener la salida PWM o en el puerto RG9 para operar como SPI esclavo. Normalmente está conectado el puerto RD4, o salida PWM.

## **16. J5 – Conector de Señales digitales**

Este conector provee acceso a los pines I/O digitales del microcontrolador.

## **17. LED Indicador de la actividad del Puerto USB**

Este Led indica la actividad de la Interfaz serie USB.

## **18. Botón de Reset**

Este botón puede ser usado para resetear el microcontrolador PIC32.

## **INTERACCIÓN ENTRE EL MPIDE Y LA INTERFAZ USB DE LA PLACA CHIPKIT UNO32**

La placa Uno32 está diseñada para ser utilizada con el Multi -Platform IDE ( MPIDE ). Digilent produjo la plataforma de desarrollo MPIDE por medio de la modificación del IDE Arduino . Por tal motivo es totalmente compatible con el IDE Arduino .

Usted no necesita un programador externo para bajar su código a la placa CHIPKIT Uno32, ya que el MPIDE utiliza un puerto de comunicaciones serie para comunicarse con el microcontrolador que viene equipado con un código “bootloader” o cargador de arranque que se ejecuta en la

placa Uno32 y que le permite autoprogramar al micro a partir de los códigos que se descargan desde el MPIDE.

El puerto serie de la CHIPKIT Uno32 se implementa mediante un FTDI FT232R. Convertidor UART a USB.

Antes de intentar usar la MPIDE para comunicarse con el CHIPKIT Uno32, debe instalarse el driver USB adecuado, el cual está incluido en la propia carpeta del entorno MPIDE.

La placa CHIPKITUno32 utiliza un mini-USB estándar para la conexión a un puerto USB en la PC . Cuando el MPIDE necesita comunicarse con la placa CHIPKIT Uno32 , la placa es reseteada y comienza su sistema de arranque . El MPIDE a continuación establece comunicaciones con el programa bootloader y descarga el programa a la placa.

Al poner el pin MCLR en cero, se resetea el microcontrolador , y se reinicia la ejecución con la gestor de arranque .

Esta acción de reset automático (cuando se abre la conexión de comunicaciones serie) puede ser deshabilitada. Para desactivar esta operación, hay Jumper JP1 el cual normalmente no está conectado.

Dos LEDs rojos (LD1 y LD2 ) parpadearán cuando los datos se están enviando o recibiendo entre el CHIPKIT Uno32 y la PC a través de la conexión en serie .

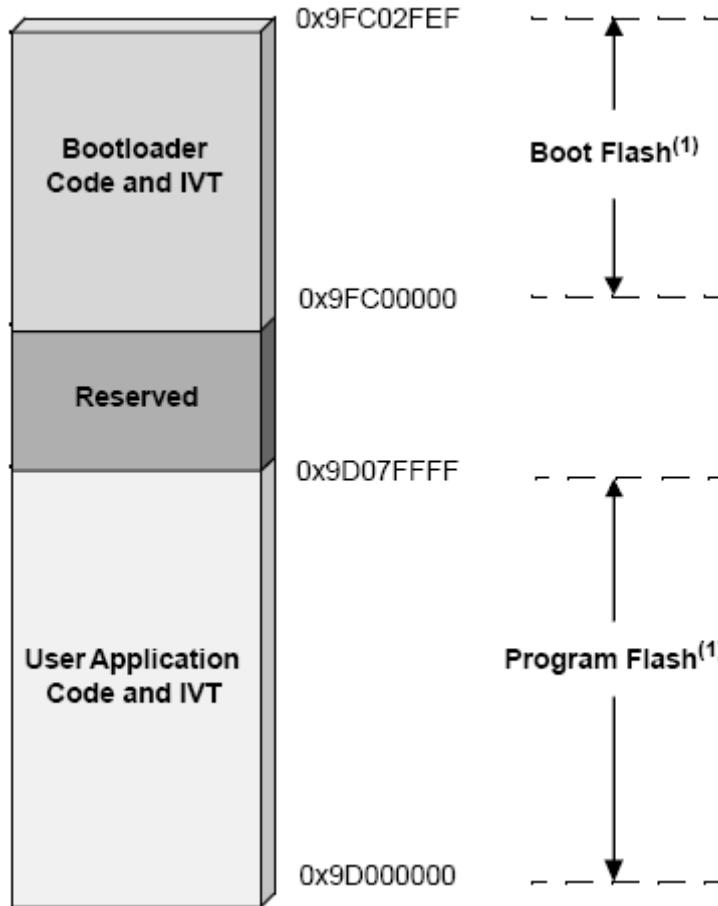
El conector J3 proporciona acceso a otras señales de comunicación serie proporcionadas por el FT232R. El conector J3 no viene de fábrica y puede ser instalado por el usuario para tener acceso a dichas señales.

## **QUE ES EL BOOTLOADER?**

El bootloader o gestor de arranque de los dispositivos PIC se utiliza para actualizar el firmware en un dispositivo sin la necesidad de un programador externo.

Se inicia el código del gestor de arranque cuando se produce el RESET del dispositivo. Si no hay condiciones para entrar en el modo de actualización del firmware, el gestor de arranque comienza la ejecución de la aplicación del usuario. El gestor de arranque realiza operaciones de borrado y auto programación de la memoria flash cuando se encuentra el modo de actualización del firmware.

La figura siguiente nos muestra como se instala el gestor de arranque dentro de la memoria de programa del PIC. Este programa es muy pequeño y utiliza mínimos recursos de memoria del microcontrolador. En el momento del RESET toma el control del arranque del microcontrolador, de allí su nombre “gestor de arranque” o bootloader, y si no detecta la “pueta en modo auto programación” salta a la primer instrucción del programa de aplicaciones realizado por el usuario. A partir de allí el control del microcontrolador queda a cargo del programa de usuario, y el bootloader queda inactivo hasta que ocurra un RESET del micro.



En la figura podemos observar el mapa de memoria parcial del PIC32 y como se aloja el bootloader en ella.

Por lo general el programa gestor de arranque, re direcciona la tabla de interrupciones principal a la tabla alternativa ya que él toma el control inicial de las interrupciones y luego se las cede al programa de aplicaciones.

## **INTRODUCCIÓN A LA ARQUITECTURA PIC32**

En el año 2007 Microchip introduce a su portfolio el microcontrolador más potente que la compañía haya diseñado, el PIC32.

Su primera familia fue el PIC32MX3, cuyos dispositivos eran en montaje superficial, y en un pin-out de 68 y 100 terminales.

El PIC32 es el primer microcontrolador de la historia de Microchip no diseñado por la compañía sino que fue encargado a otra empresa denominada MIPS Technologies, la cual es muy conocida en el campo de los núcleos de 32 y 64 bits, por ser una desarrolladora de núcleos para terceros de 32 y 64 bits.

Dicho núcleo se insertó en el interior de un PIC24H128, al cual Microchip lo adaptó para poder colocar su nuevo CORE.

De esta forma el usuario goza de un núcleo de 32 bits de altísimas prestaciones, y de los periféricos que tiene los PIC24H.

Actualmente la familia está integrada por los PIC32MX1, PIC32MX2, PIC32MX3, PIC32MX4, PIC32MX5, PIC32MX6 y PIC32MX7, el conjunto de todas estas familias determina que PIC32 actualmente esté formada por más de 70 modelos distintos.

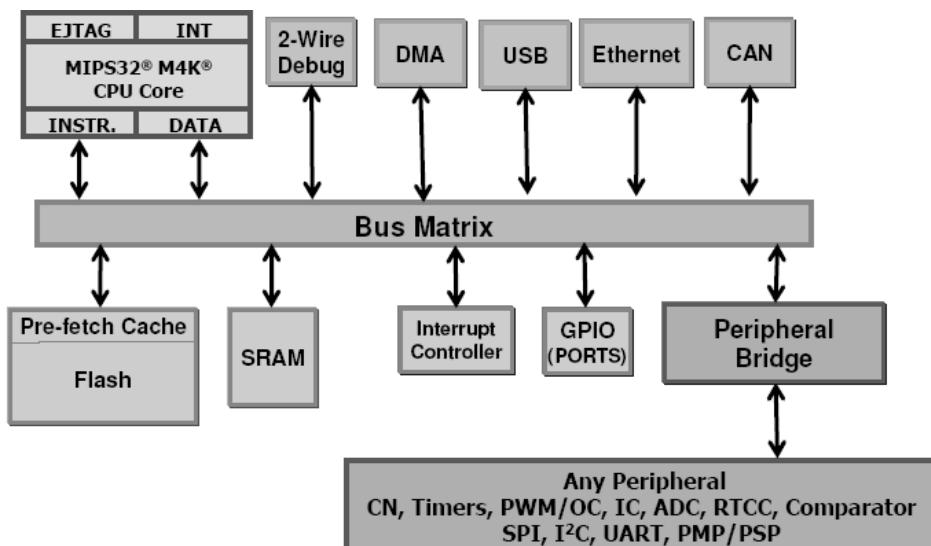
## **LOS PIC32 CARACTERÍSTICAS**

Para comenzar con nuestra descripción diremos que un PIC32 tiene internamente una arquitectura mixta, ya que dentro de su CORE, su arquitectura es HARVARD, es decir que existe un bus de instrucciones y otro de datos. Pero si lo vemos desde la memoria de programa y la memoria de datos, el PIC tiene una arquitectura VON NEWMAN.

Esta dualidad termina con el viejo problema de Microchip de los accesos a la memoria de programa para la lectura de tablas de datos, por existir dos buses diferenciados, y que las arquitecturas anteriores habían salvado el escollo, por medio de la implementación de una herramienta complementaria; en el caso de PIC18, se había echado mano al mecanismo de Punteros de Tabla y un registro TABLAT, y luego en los micros de 16 bits, PIC24 y dsPIC, se usaba el recurso de VSP, mediante el cual se podía visualizar parte de la memoria de programa desde la RAM.

Estos mecanismos aunque eficientes, perdían ciclos de máquina y fue por ello que MIPS decidió que ambas memorias podían enlazarse con un mismo BUS de datos, ya que este era de 32 bits y tenía el espacio suficiente para poder transferir las instrucciones.

Pero Microchip, para acelerar el procesamiento de las mismas solicito a MIPS que en el interior del CORE, las instrucciones y los datos se separasen. De esta manera las instrucciones podían ser procesadas por un nuevo pipeline de 5 niveles. Esto mejoró la performance del procesador permitiendo ejecutar las instrucciones en un pulso de clock, con lo cual, estando el pipeline cargado, el PIC32 a 80MHZ puede procesar 80 MIPS !



Este es un diagrama simplificado de la arquitectura de la familia PIC32.

El corazón del PIC32 es el nucleo M4K. Una interfaz JTAG mejorada (EJTAG) esta implementada con acceso directo al nucleo para soportar las funciones de debugging.

El nucleo tiene 2 buses uno usado para buscar las instrucciones, y el otro para acceder a los datos.

A fin de garantizar el rendimiento mas alto posible de transferencia de datos e instrucciones se llevo a cabo un “Bus Matriz” en lugar de la típica arquitectura “dato y direcciones”, bus común en los CPU de menor porte. El bus matriz puede visualizarse como una autopista con multiples carriles en comparación con el bus de vía única de un microcontrolador típico.

Debido a que hay disponibles muchos carriles para el tráfico, los datos y las instrucciones pueden ser transmitidos desde muchas fuentes y hacia muchos destinos en un mismo ciclo de máquina. Naturalmente hay limitaciones en cuanto a lo que es posible, pero el hardware se asegura de que todo el mundo tenga su tiempo justo en el BUS. De hecho el programador puede seleccionar entre diferentes variantes de prioridad de acceso. El bus matriz esta funcionando a la misma frecuencia que el core de la CPU.

La memoria FLASH almacena constantes e instrucciones del programa y la RAM almacena los datos y las variables, estan también conectadas a bus matriz. En la memoria también se puede observar un elemento nuevo: un módulo de prebúsqueda.

La FLASH implementada en los PIC32 no puede entregar la información a la CPU más rápido de lo que la CPU puede consumir dicha información. Con el fin de resolver este desequilibrio el módulo de prebúsqueda ofrece 2 características:

- 1) Un mecanismo de CACHE que puede almacenar 64 instrucciones de 32 bits. La propia CACHE es una memoria RAM de tecnología de alta velocidad que puede entregar datos e instrucciones al nucleo a la misma frecuencia que el micro está ejecutando el código.
- 2) Mecanismo de captación previo que usa un algoritmo de “predicción de salto”, mediante el cual se carga la CACHE con instrucciones antes de su tiempo de ejecución. Puede imaginarse dicho mecanismo como una “bola de cristal” que intenta predecir que instrucción se necesita.

Ambas características ayudan a que el código se ejecute con la mayor velocidad posible. Además cabe señalar que cada ciclo de búsqueda lee un bloque de datos de 128bits, lo que equivale a 4 instrucciones de 32 bits (u 8 para las instrucciones de 16 bits denominadas MIPS16e).

El BUS Matriz puede incluir la conexión a un controlador ETHERNET,CAN, USB, DMA, Puertos I/O.

El resto de los periféricos estan conectados a un puente de periféricos. Dicho puente puede trabajar a la misma velocidad que la del nucleo o a otra alternativa a traves de un divisor de frecuencia.

## **EL CORE PIC32 (M4K)**

El nucleo es capaz de ejecutar una instrucción por cada pulso de clock. Esta compuesto por 32 registros de 32 bits disponibles para el almacenamiento de datos locales, y otro conjunto completo de 32 registros de 32 bits que pueden ser utilizados por la rutina de interrupciones, reduciendo así la sobrecarga de entrada y salida de la interrupción.

La ALU es de 32 bits de ancho y soporta instrucciones del tipo MAC (Multiplica y Acumula) en un solo ciclo. Cuando el espacio del código generado es muy crítico, es posible compilar la totalidad o parte del código siguiendo las instrucciones MIPS16e donde las instrucciones son de 16 bits en lugar de 32. Esto permite ahorrar espacio en memoria de programa, pero aumenta el tiempo de ejecución de forma drástica.

Como en todos los microcontroladores PIC, los datos se almacenan en forma Little Indian, es decir, el byte menos significativo se encuentra en la dirección mas baja de memoria. El corazón del nucleo es la unidad de ejecución, la cual se ocupa de la aritmética estandar, las operaciones de rotación y las instrucciones de desplazamiento.

El propio nucleo utiliza un pipeline de 5 etapas necesario para cargar, ejecutar y guardar el resultado de cada instrucción, lo que requiere de un total de 5 pulsos de clock, sin embargo cuando el pipeline esta totalmente cargado, la CPU es capaz de ejecutar cada instrucción en un solo pulso de clock.

Las instrucciones de salto implican un retardo adicional, conocido como “retardo de salto”, debido a la estructura secuencial del pipeline, sin embargo la mayoría de los compiladores y ensambladores llenan ese retardo con una instrucción util (siempre que sea posible), eliminando así cualquier pérdida de rendimiento en la mayoría de los casos.

Se suma al nucleo la unidad de multiplicación y división (MDU), la cual realiza calculos de  $16 \times 16$  y  $16 \times 32$  bits en un solo ciclo de maquina. Las multiplicaciones de  $32 \times 32$ bits usan 2 ciclos de maquina. Las instrucciones de división necesitan de 35 ciclos pero de forma inteligente usa tan solo 13 ciclos si se detecta que la división es menor a 32 bits.

El compilador debe reordenar las instrucciones necesarias para garantizar que el resultado esta disponible cuando el pipeline del nucleo lo requiera. Dado que esta operación se ejecuta en paralelo al nucleo y con el reordenamiento inteligente de las instrucciones por parte del compilador, no debería haber pérdidas de rendimiento.

El nucleo no es muy inteligente cuando se inicia, no tiene idea por ejemplo de como se conecta la memoria disponible y no tiene ningún registro de estado. Con el fin de remediar esta situación existe lo que se denomina el coprocesador cero (copro 0). El nombre es un poco engañoso, el nucleo PIC32 puede soportar coprocesadores, pero no se implementa en el PIC32.

El coprocesador 0 es para implementar el registro de estado del nucleo, configurar el contador de memoria de cache, el manejo de excepciones e interrupciones, configurar la unidad de memoria y programar otras características.

Antes de que cualquier código se ejecute, los registros del coprocesador 0 deben ser inicializados apropiadamente. Esto es realizado por un código conocido como “codigo de arranque” y se vincula automáticamente al código de la aplicación.

## EL MPIDE

Es el entorno de desarrollo propio que ha desarrollado DIGILENT para trabajar con su placa Arduino compatible para PIC32. Este entorno es totalmente compatible con los entornos IDE de otras plataformas para otros MCU arduino compatibles:



Ventana principal de MPIDE para Windows

Funciones del programa:

**Verify/Compile:** Chequea el código en busca de errores.

**New:** Crea una nueva rutina.

**Open:** Muestra un menú con todas las rutinas de tu \_sketchbook\_.

**Save:** Guarda las rutinas.

**Upload to I/O board:** Carga el código a la placa Arduino I/O. Asegúrate de guardar o verificar tu rutina antes cargarla.

**Serial Monitor:** Muestra datos serie enviados desde la placa Arduino (placa serie o USB). Para enviar datos a la placa, introduce el texto y haz click en el botón Send o presiona Enter. Elige la velocidad de transmisión de datos desde el menú desplegable.

**Tab Menu** Permite gestionar las rutinas con más de un archivo (cada uno de los cuales aparece en su propia pestaña). Estos pueden ser:

Archivos de código de Arduino (sin extensión).

Archivos de C (extensión .c).

Archivos de C++ (extensión .cpp).

Archivos de cabecera (extensión .h).

**Sketch Verify/Compile:** Comprueba tu rutina para errores.

**Import Library:** Utiliza una librería en tu rutina. Trabaja añadiendo #include en la cima de tu código. Esto añade funcionalidad extra a tu rutina, pero incrementa su tamaño. Para dejar de usar una librería, elimina el #include apropiado de la cima de tu rutina.

**Show Sketch Folder:** Abre la carpeta de rutinas en tu escritorio.

**Add File... :** Añade otro archivo fuente a la rutina. El nuevo archivo aparece en una nueva pestaña en la ventana de la rutina. Esto facilita y agranda proyectos con múltiples archivos fuente. Los archivos pueden ser eliminados de una rutina usando el Tab Menu.

## Tools

**Auto Format:** Esto formatea el código amigablemente.

**Copy for Discourse:** Copia el código de la rutina al portapapeles.

**Board:** Selecciona la placa que está usando. Esto controla la forma en que la rutina es compilada y cargada así como el comportamiento de los elementos del menú Burn Bootloader.

**Serial Port:** Este menú contiene todos los dispositivos serie (reales o virtuales) de la máquina. Debería actualizarse automáticamente cada vez que abre el nivel superior del menú Tools. Antes de subir tu rutina, necesita seleccionar el elemento de este menú que representa a su placa Arduino.

En Windows, es probablemente COM1 o COM2 (para una placa Serie) o COM4, COM5, COM7 o superior (para una placa USB) – para descubrirlo, buscar el USB serial device en la sección puertos del Gestor de dispositivos de Windows.

**Burn Bootloader:** Los elementos en este menú le permiten grabar un bootloader en su placa con una variedad de programadores. Esto no es necesario para uso normal de una placa Arduino. Asegúrese que ha seleccionado la placa correcta del menú Boards de antemano.

## ESTRUCTURA DE UN PROGRAMA ARDUINO

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes.

```
void setup()
{
    sentencias;
}

void loop()
{
    sentencias;
}
```

En este esquema `setup()` es la parte encargada de recoger la configuración y `loop()` es la que contienen el programa que se ejecutará cíclicamente (de ahí el término `loop`). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar **pinMode (modo de trabajo de las E/S)**, configuración de la comunicación en serie y otras.

La función bucle (`loop`) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

La función `setup()` se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar.

```
void setup()
{
    pinMode(pin, OUTPUT); // configura el 'pin' como salida
}
```

Después de llamar a `setup()`, la función `loop()` hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la placa.

Por ejemplo el siguiente código hace titilar un LED, el cual ha sido colocado en el Pin 0:

```
void loop()
{
    digitalWrite(0, HIGH); // pone en uno el 'pin'0
    delay(1000); // espera un segundo (1000 ms)
    digitalWrite(0, LOW); // pone en cero el 'pin'0
    delay(1000);
}
```

## Las Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de sentencias que son ejecutadas cuando se llama a la función. Son funciones `setup()` y `loop()` de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa.

Las funciones se declaran asociadas a un tipo de valor. Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra "void".

Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
tipo nombreFunción (parámetros)
{
    sentencias;
}
```

El siguiente ejemplo de una función devuelve un número entero, `Val_adc()` se utiliza para leer el valor analógico de una entrada analógica donde hemos colocado un potenciómetro. Al principio se declara como una variable local, 'v' recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor 'v' y se retornaría al programa principal.

Esta función cuando se ejecuta devuelve el valor de tipo entero 'v'

```
int Val_adc()
{
    int v; // crea una variable temporal 'v'
    v= analogRead(A0); // lee el valor del potenciómetro
    v = v/ 4; // convierte 0-1023 a 0-255
    return v; // devuelve el valor final
}
```

### Las llaves delimitadoras de bloque

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), if., etc.

```
tipo funcion()
{
    sentencias;
}
```

Una llave de apertura “{“ siempre debe ir seguida de una llave de cierre “}”, si no es así el programa dará errores.

El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo hay que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

### Indicador de final de línea

El punto y coma “;” se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo “bucle for”.

**int x = 13; // declara la variable 'x' como tipo entero de valor 13**

**Nota:** Olvidarse de poner fin a una línea con un punto y coma se traducirá en un error de compilación. El texto de error puede ser obvio, y se referirá a la falta de una coma, o puede que no. Si se produce un error raro y de difícil detección lo primero que debemos hacer es comprobar que los puntos y comas están colocados al final de las instrucciones.

## **Los Comentarios**

Comentario tipo bloque:

Los bloques de comentarios, o multi-línea de comentarios, son áreas de texto ignorados por el programa que se utilizan para las descripciones del código o comentarios que ayudan a comprender el programa.

Comienzan con `/*` y terminan con `*/` y pueden abarcar varias líneas.

```
/*
esto es un bloque de comentario
no se debe olvidar cerrar los comentarios
estos deben estar equilibrados
*/
```

Debido a que los comentarios son ignorados por el programa y no ocupan espacio en la memoria de Arduino pueden ser utilizados con generosidad y también pueden utilizarse para "comentar" bloques de código con el propósito de anotar informaciones para depuración.

Nota: Dentro de una misma línea de un bloque de comentarios no se puede escribir otro bloque de comentarios (usando `/* .. */`).

Comentario tipo linea:

Una línea de comentario empieza con `//` y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

**// esto es un comentario**

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace esta o para recordarla más adelante.

## **Las variables en Memoria**

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia.

Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada `variableEntrada` y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int variableEntrada = 0; // declara una variable y le asigna el valor 0  
variableEntrada = analogRead(2); // la variable recoge el valor del PIN2
```

'variableEntrada' es la variable en sí. La primera línea declara que será de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código.

Una vez que una variable ha sido asignada, o re-asignada, usted puede probar su valor para ver si cumple ciertas condiciones (instrucciones if..), o puede utilizar directamente su valor.

Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto, se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (delay) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100) // pregunta si la variable es menor de 100  
{  
    entradaVariable = 100; // si es cierto asigna el valor 100 a esta  
}  
delay(entradaVariable); // usa el valor como retardo
```

Nota: Las variables deben tomar nombres descriptivos, para hacer el código más legible.

Nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de Arduino.

## Declaración de variables

Todas las variables tienen que declararse antes de que puedan ser utilizadas. Para declarar una variable se comienza por definir su tipo como int (entero), long (largo), float (coma flotante), etc, asignándole siempre un nombre, y, opcionalmente, un valor inicial.

Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas. El siguiente ejemplo declara la variable entradaVariable como una variable de tipo entero "int", y asignándole un valor inicial igual a cero. Esto se llama una asignación.

```
int entradaVariable = 0;
```

Una variable puede ser declarada en una serie de lugares del programa y en función del lugar en donde se lleve a cabo la definición esto determinará en que partes del programa se podrá hacer uso de ella.

### Utilización de las variables

Una variable puede ser declarada al inicio del programa antes de la parte de configuración `setup()`, a nivel local dentro de las funciones, y, a veces, dentro de un bucle, como para los bucles del tipo `if.. for.., etc.`

En función del lugar de declaración de la variable así se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella.

Una variable global es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de `setup()`.

Una variable local es aquella que se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que sólo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo muestra cómo declarar a unos tipos diferentes de variables y la visibilidad de cada variable:

```
int value;           // 'value' es visible para cualquier función

void setup()
{
    // no es necesario configurar
}

void loop()
{
    for (int i=0; i<20;)      // 'i' solo es visible
    {
        // dentro del bucle for
        i++;
    }
    float f;                // 'f' es visible solo dentro de loop
}
```

## Tipos de Variables

Existen distintos tipos de variables que se pueden definir en el lenguaje Arduino para almacenar datos:

**byte:** almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255

```
byte contador = 180; // declara 'contador' como tipo byte
```

**int:** Enteros son un tipo de datos primarios que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32,767 to -32,768.

```
int x = 1500; // declara 'x' como una variable de tipo entero
```

**Nota:** Las variables de tipo entero “int” pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor se x pasará a ser -32.768. (algo así como que el valor da la vuelta).

**long:** El formato de variable numérica de tipo extendido “long” se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

```
long y = 90000; // declara 'y' como tipo long
```

**float:** El formato de dato del tipo “punto flotante” “float” se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

```
float Pi = 3.14; // declara 'Pi' como tipo flotante
```

**Nota:** Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones.

## Arrays o Matrices:

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice.

El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado

```
int buffer[ ] = {valor0, valor1, valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición específica:

```
int buffer[5]; // declara un array de enteros de 6 posiciones
```

```
buffer[3] = 10; // asigna el valor 10 a la posición 4
```

Para leer de un array basta con escribir el nombre y la posición a leer:

```
x = buffer[3]; // x ahora es igual a 10 que está en la posición 3 del array
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array.

El siguiente ejemplo usa una matriz para el parpadeo de un LED.

Utilizando un bucle tipo **for**, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del array parpadeo[], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice “i”.

```
int ledPin = 10; // Salida LED en el PIN 10
byte parpadeo[] = {180, 30, 255, 200, 10, 90, 150, 60}; // array de 8 valores

void setup()
{
    pinMode(ledPin, OUTPUT); //configura la salida PIN 10
}

void loop() // bucle del programa
{
    for(int i=0; i<8; i++) // crea un bucle tipo for utilizando la variable i
    {
        analogWrite(ledPin, parpadeo[i]); // escribe en la salida PIN10
        delay(200); // espera 200ms
    }
}
```

## Operaciones Aritméticas con Arduino

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operandos:

```
y = y + 3;  
x = x - 7;  
i = j * 6;  
r = r / 5;
```

La operaciones se efectúa teniendo en cuenta el tipo de datos que hemos definido para los operandos (int, float, etc..), por lo que, por ejemplo, si definimos 9 y 4 como enteros “int”, 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 se valores de tipo entero “int” (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos. Recordemos el alcance de los tipos de datos numéricos que ya hemos explicado anteriormente.

Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si uno de los números (operandos) es del tipo float y otra de tipo integer, para el cálculo se utilizará el método de float es decir el método de coma flotante.

Elija el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo float, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el computo..

Nota: Utilice el operador (int) x para convertir un tipo de variable a otro sobre la marcha. Por ejemplo:

**i = (int) 3,6 establecerá i igual a 3.**

## Asignaciones combinadas con operaciones

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles tal como se describe más adelante. Estas asignaciones compuestas pueden ser:

```
x ++ // igual que x = x + 1, o incrementar x en +1  
x -- // igual que x = x - 1, o decrementar x en -1  
x += y // igual que x = x + y, o incrementar x en +y  
x -= y // igual que x = x - y, o decrementar x en -y  
x *= y // igual que x = x * y, o multiplicar x por y  
x /= y // igual que x = x / y, o dividir x por y
```

**Nota:** Por ejemplo,  $x^* = 3$  hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor.

## Comparaciones

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo **if..** para testear si una condición es verdadera. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

```
x == y // x es igual a y  
x != y // x no es igual a y  
x < y // x es menor que y  
x > y // x es mayor que y  
x <= y // x es menor o igual que y  
x >= y // x es mayor o igual que y
```

## Operadores Lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un **VERDADERO** o **FALSO** dependiendo del operador. Existen tres operadores lógicos, AND (**&&**), OR (**||**) y NOT (**!**), que a menudo se utilizan en sentencias de tipo **if..**:

Logica AND:

**if (x > 0 && x < 5) // cierto sólo si las dos expresiones son ciertas**

Logica OR:

**if (x > 0 || y > 0) // cierto si una cualquiera de las expresiones es cierta**

Logica NOT:

**if (!x > 0) // cierto solo si la expresión es falsa**

## Constantes usadas por ARDUINO

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos:

**TRUE/FALSE:** Estas son constantes booleanas que definen los niveles **HIGH** (alto) y **LOW** (bajo) cuando estos se refieren al estado de las salidas digitales. FALSE se asocia con 0 (cero), mientras que TRUE se asocia con 1, pero TRUE también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se define como TRUE.

```
if (b == TRUE);  
{  
    ejecutar las instrucciones;  
}
```

**HIGH/LOW:** Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. ALTO se define como en la lógica de nivel 1, ON, ó 3 voltios, mientras que BAJO es lógica nivel 0, OFF, o 0 voltios.

```
digitalWrite(13, HIGH); // activa la salida 13 con un nivel alto (3v.)
```

**INPUT/OUTPUT:** Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

```
pinMode(13, OUTPUT); // designamos que el PIN 13 es una salida
```

## Sentencias estructurales

Estas sentencias nos permiten procesar la información que incorporan los programas y que por lo general, le dan a estos su capacidad de toma de decisión:

### Sentencia “if” (si..?)

El **if** es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves.

El formato para if es el siguiente:

```
if (variable?? valor)
{
    ejecutaInstrucciones;
}
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

**Nota:** Tenga en cuenta el uso especial del símbolo '=' , poner dentro de if (x = 10), podría parecer que es válido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10.. Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual “==”.

### Sentencia if-else

**if... else** viene a ser un estructura que se ejecuta en respuesta a la idea “si esto no se cumple haz esto otro”. Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, usted escribiría que de esta manera:

```
if (inputPin == HIGH) // si el valor de la entrada inputPin es alto
{
    instruccionesA; //ejecuta si se cumple la condición
}
else
{
    instruccionesB; //ejecuta si no se cumple la condición
}
```

**Else** puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez.

Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo qué sólo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada:

```

if (inputPin < 500)
{
    instruccionesA; // ejecuta las operaciones A
}
else if (inputPin >= 1000)
{
    instruccionesB; // ejecuta las operaciones B
}
else
{
    instruccionesC; // ejecuta las operaciones C
}

```

**Nota:** Un **if** prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos

**(inputPin == HIGH)**

En este caso, la sentencia **if** sólo chequearía si la entrada especificada está en nivel alto (HIGH), o +3v.

## Bucle FOR

La declaración **for** se usa para repetir un bloque de sentencias encerradas entre llaves un número determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;) vemos el ejemplo de su sintaxis:

```

for (inicialización; condición; expresión)
{
    sentencias;
}

```

La inicialización de una variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple, el bucle termina.

El siguiente ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a 20 y si es cierta i se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves:

```

for (int i=0; i<20; i++) // declara i, prueba que es menor que 20, incrementa
{
    digitalWrite(13, HIGH); // envía un 1 al pin 13
    delay(250); // espera ¼ seg.
    digitalWrite(13, LOW); // envía un 0 al pin 13
    delay(250); // espera ¼ de seg.
}

```

**Nota:** El bucle en el lenguaje C es mucho más flexible que otros bucles encontrados en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera de los tres elementos de cabecera puede omitirse, aunque el punto y coma es obligatorio. También las declaraciones de inicialización, condición y expresión puede ser cualquier sentencia válida en el lenguaje C sin relación con las variables declaradas.

## While

Un bucle del tipo while es un bucle de ejecución continua mientras se cumpla la Expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor

```

while (variable ?? valor)
{
    Sentencias;
}

```

El siguiente ejemplo testea si la variable "x" es inferior a 200 y, si es verdad, ejecuta las declaraciones dentro de los corchetes y continuará ejecutando el bucle hasta que 'x' no sea inferior a 200.

```

While (x < 200) // testea si es menor que 200
{
    sentencia; // ejecuta las instrucciones entre llaves
    x++; // incrementa la variable en 1
}

```

## **Do-while**

El bucle **do-while** funciona de la misma manera que el bucle while, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

```
do{  
    Instrucciones;  
} while (variable ?? valor);
```

El siguiente ejemplo asigna el valor leído leeSensor() a la variable 'x', espera 50 milisegundos, y luego continua mientras que el valor de la 'x' sea inferior a 100:

```
do{  
    x = leeSensor();  
    delay(50);  
} while (x < 100);
```

## **FUNCIONES EMBEBIDAS DE ARDUINO**

El lenguaje Arduino toma como base elementos del lenguaje C y del C++, pero además está “condimentado” con una serie de funciones específicas que tienden a facilitarle al programador el manejo del hardware. Estas funciones están “incluidas dentro del compilador” y no son accesibles para su edición por parte del usuario, de allí su nombre “embebidas”. Para su uso no necesitan de la inclusión de ningún archivo de cabecera externo.

### **pinMode()**

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida).

**pinMode(pin, OUTPUT); // configura ‘pin’ como salida**

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas.

PIN en realidad debe ser reemplazado por el número de pin de la placa Arduino con el cual vamos a trabajar, y que generalmente está impreso en el conector de las pineras de acceso del PCB.

Los pines configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta impedancia.

Los pins configurado como OUTPUT (salida) se dice que están en un estado de baja impedancia y pueden proporcionar 25mA(miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocará una corriente elevada que puede dañar o destruir el chip PIC32. A menudo es una buena idea conectar en la OUTUPT (salida) una resistencia externa de 470 o de 1000 ohms.

### **digitalRead()**

Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

```
x = digitalRead(11); // hace que 'x' sea igual al estado leído en el pin 11
```

### **digitalWrite()**

Envía al 'pin' definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

```
digitalWrite(11, HIGH); // deposita en el 'pin 11' un valor HIGH (alto o 1)
```

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo escribe en el 'pin' de salida LED:

```
int led = 13; // asigna a LED el valor 13
int boton = 7; // asigna a botón el valor 7
int valor = 0; // define el valor y le asigna el valor 0
void setup()
{
    pinMode(led, OUTPUT); // configura el led (pin13) como salida
    pinMode(boton, INPUT); // configura botón (pin7) como entrada
}
void loop()
{
    valor = digitalRead(boton); //lee el estado de la entrada botón
    digitalWrite(led, valor); // envía a la salida 'led' el valor leído
}
```

### **analogRead()**

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023.

```
valor = analogRead(pin); // asigna a valor lo que lee en la entrada 'pin'
```

Nota: En la placa CHIPKIT Uno32, existen 12 canales analógicos.

Los pins analógicos (A0-A11) a diferencia de los pines digitales, no necesitan ser declarados como INPUT u OUTPUT ya que son siempre INPUT's.

La tensión aplicada a un pin analógico puede tener cualquier valor de tensión entre 0 y 3.3V como máximo.

### **analogWrite()**

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pin's de Arduino marcados como "pin PWM". El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

```
analogWrite(pin, valor); // escribe 'valor' en el 'pin'
```

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 3 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 3 voltios - el valor HIGH de salida equivale a 3v.

Teniendo en cuenta el concepto de señal PWM , por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 3 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 3 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 3 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generará una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue a ejecutar otra instrucción analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).

**Nota:** Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como INPUT u OUTPUT.

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo PWM o salida analógica:

```

int led = 10; // define el pin 10 como 'led'
int analog = 0; // define el pin 0 como 'analog'
int valor; // define la variable 'valor'

void setup()
{
} // no es necesario configurar entradas y salidas

void loop()
{
    valor = analogRead(analog); // lee el pin 0 y lo asocia a la
    // variable valor
    valor /= 4; // divide valor entre 4 y lo reasigna a valor
    analogWrite(led, value); // escribe en el pin10 valor
}

```

### **delay(ms)**

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

**delay(1000); // espera 1 segundo**

### **millis()**

Devuelve el número de milisegundos transcurrido desde el inicio del programa en Arduino hasta el momento actual. Normalmente será un valor grande (dependiendo del tiempo que esté en marcha la aplicación después de cargada o después de la última vez que se pulsó el botón "reset" de la tarjeta)..

**valor = millis(); // valor recoge el número de milisegundos.**

**Nota:** Este número se desbordará (si no se resetea de nuevo a cero), después de aproximadamente 9 horas.

### **min(x,y)**

Calcula el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

**x = min(x, 100); // asigna a x el más pequeños de los dos números**

Si 'x' es menor que 100 recogerá su propio valor si 'x' es mayor que 100 valor pasara a valer 100.

### **max(x,y)**

Calcula el máximo de dos números para cualquier tipo de datos devolviendo el número mayor de los dos.

```
x = max(x, 100); // asigna a x el mayor de los dos números 'x' y 100.
```

De esta manera nos aseguramos de que valor será como mínimo 100.

### **randomSeed()**

Establece un valor, o semilla, como punto de partida para la función random().

```
randomSeed(valor); // hace que valor sea la semilla del random
```

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, randomSeed le permite colocar una variable, constante, u otra función de control dentro de la función random, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas, o funciones, que pueden ser utilizados en esta función, incluido millis () o incluso analogRead () que permite leer ruido eléctrico a través de un pin analógico.

### **random()**

La función random devuelve un número aleatorio entero de un intervalo de valores especificado entre los valores min y max.

```
x = random(100, 200); // asigna a 'x' un numero aleatorio entre 100 y 200
```

**Nota:** Use esta función después de usar el randomSeed().

El siguiente ejemplo genera un valor aleatorio entre 0-255 y lo envía a una salida analógica PWM:

```
int randNumber; // variable que almacena el valor aleatorio
int led = 10; // define led como 10

void setup() {} // no es necesario configurar nada

void loop()
{
    randomSeed(millis()); // genera una semilla para aleatorio
    randNumber = random(255); // genera número aleatorio entre 0-255
    analogWrite(led, randNumber); // envía a la salida led tipo PWM
    delay(500); // espera 0,5 seg.
}
```

### **Serial.begin(baud-rate)**

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
void setup()
{
    Serial.begin(9600); // abre el Puerto serie
} // Configurando la velocidad en 9600 bps
```

**Nota:** Cuando se utiliza la comunicación serie los pins digital 0 (RX) y 1 (TX) no puede utilizarse al mismo tiempo.

### **Serial.println(datos)**

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.print (), pero es más fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println(analogValue); // envía el valor 'analogValue' al puerto
```

Nota: Para obtener más información sobre las distintas posibilidades de **Serial.println ()** y **Serial.print ()** puede consultarse el sitio web de Arduino.

El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al ordenador cada 1 segundo.

```
void setup()
{
    Serial.begin(9600); // configura el puerto serie a 9600bps
}

void loop()
{
    Serial.println(analogRead(A0)); // envía valor analógico
    delay(1000); // espera 1 segundo
}
```

### **Serial.println(dato, tipo de dato)**

Vuelca o envía un número o una cadena de caracteres al puerto serie, seguido de un carácter de retorno de carro "CR" (ASCII 13, or '\r')y un carácter de salto de línea "LF"(ASCII 10, or '\n'). Toma la misma forma que el comando Serial.print()

**Serial.println(b)** vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

**Serial.println(b, DEC)** vuelca o envía el valor de b como un número decimal en caracteres ASCII seguido de "CR" y "LF".

**Serial.println(b, HEX)** vuelca o envía el valor de b como un número hexdecimal en caracteres ASCII seguido de "CR" y "LF".

**Serial.println(b, OCT)** vuelca o envía el valor de b como un número Octal en caracteres ASCII seguido de "CR" y "LF".

**Serial.println(b, BIN)** vuelca o envía el valor de b como un número binario en caracteres ASCII seguido de "CR" y "LF".

**Serial.println(b, BYTE)** vuelca o envía el valor de b como un byteseguido de "CR" y "LF".

**Serial.println(str)** vuelca o envía la cadena de caracteres como una cadena ASCII seguido de "CR" y "LF".

**Serial.println()** sólo vuelca o envía "CR" y "LF". Equivaldría a printNewline().

### **Serial.print(dato, tipo de dato)**

Vuelca o envía un número o una cadena de caracteres, al puerto serie. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que utilicemos para definir el formato de volcado de los números.

Parámetros

**data:** el número o la cadena de caracteres a volcar o enviar.

**data type:** determina el formato de salida de los valores numéricos (decimal, octal, binario, etc...) DEC, OCT, BIN, HEX, BYTE , si no se pe nada vuelva ASCII

### **Ejemplos**

#### **Serial.print(b)**

Vuelca o envía el valor de b como un número decimal en caracteres ASCII. Equivaldría a **printInteger()**.

**int b = 79;**

**Serial.print(b); // prints the string "79".**

**Serial.print(b, DEC)**

Vuelca o envía el valor de b como un número decimal en caracteres ASCII.

Equivaldría a **printInteger()**.

**int b = 79;**

**Serial.print(b, DEC); // prints the string "79".**

**Serial.print(b, HEX)**

Vuelca o envía el valor de b como un número hexdecimal en caracteres ASCII.

Equivaldría a **printHex()**;

**int b = 79;**

**Serial.print(b, HEX); // prints the string "4F".**

**Serial.print(b, OCT)**

Vuelca o envía el valor de b como un número Octal en caracteres ASCII. Equivaldría a

**printOctal()**;

**int b = 79;**

**Serial.print(b, OCT); // imprime el string "117".**

**Serial.print(b, BIN)**

Vuelca o envía el valor de b como un número binario en caracteres ASCII. Equivaldría a

**printBinary()**;

**int b = 79;**

**Serial.print(b, BIN); // imprime el string "1001111".**

**Serial.print(b, BYTE)**

Vuelca o envía el valor de b como un byte. Equivaldría a **printByte()**;

**int b = 79;**

**Serial.print(b, BYTE); // Devuelve el carácter "O", el cual representa el carácter ASCII del valor 79. (Ver tabla ASCII ).**

**Serial.print(str)**

Vuelca o envía la cadena de caracteres como una cadena ASCII. Equivaldría a

**printString()**.

**Serial.print("Hola Mundo!"); // vuelca "Hola Mundo!".**

**serial.available()**

Obtiene un número entero con el número de bytes (caracteres) disponibles para leer o capturar desde el puerto serie. Equivaldría a la función **serialAvailable()**.

Devuelve Un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, SerialAvailable() será mayor que 0.  
El buffer serie puede almacenar como máximo 64 bytes.

Ejemplo

```
int bufferRX = 0; // almacena el dato serie
void setup()
{
    Serial.begin(9600); // abre el puerto serie a velocidad de 9600bps
}

void loop()
{
// envía datos sólo si los recibe:
    if (Serial.available() > 0)
    {
        // lee el byte de entrada por USART:
        bufferRX = Serial.read();
        //lo vuelca a la USART
        Serial.print("recibe: ");
        Serial.println(bufferRX, DEC);
    }
}
serial.Read()
```

Lee o captura un byte (un carácter) desde el puerto serie. Equivaldría a la función **serialRead()**.  
Devuelve :El siguiente byte (carácter) desde el puerto serie, o -1 si no hay ninguno.

Ejemplo:

```
int buffer = 0; // almacenar el dato serie

void setup()
{
    Serial.begin(9600); // abre el puerto serie,y le asigna a 9600 Bps
}

void loop()
{
// envía datos sólo si los recibe:
    if (Serial.available() > 0)
    {
// lee el byte de entrada:
        buffer = Serial.read();
//lo envía a la USART
        Serial.print("I receive: ");
        Serial.println(buffer, DEC);
    }
}
```



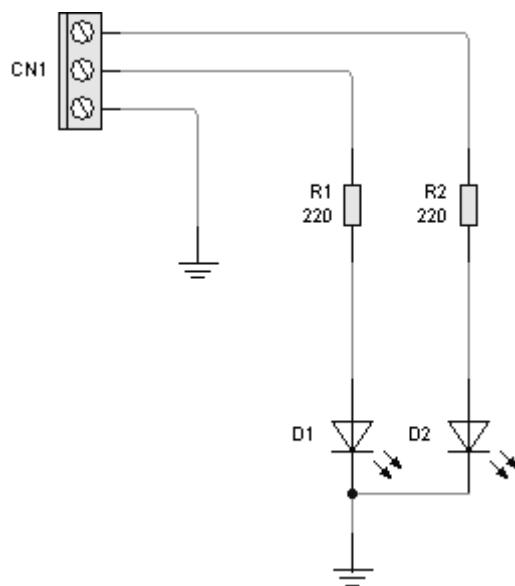
# **EJERCICIOS CON ARDUINO**



## EJERCICIO 1: BASCULA DE LEDS

**Objetivo:** La idea de este ejercicio inicial es introducir los elementos básicos de un programa ARDUINO típico. Para este caso se propone hacer bascular 2 LEDs, los cuales encenderán de forma alternativa.

**El Hardware necesario:**



**El Programa:**

```
//Definiciones generales

#define LED1 13 //definimos etiqueta LED1 para el PIN 13
#define LED2 26 //definimos etiqueta LED2 para el PIN 26
#define Tiempo 200 //definimos el tiempo

//La siguiente función es la que nos permite configurar I/O
void setup()
{
// inicializamos los pines como salida.
    pinMode(LED1, OUTPUT);
    pinMode(LED2, OUTPUT);
}
```

```
//La siguiente función es la Principal
void loop()
{
    digitalWrite(LED1, HIGH); // encendemos el LED
    digitalWrite(LED2, LOW); // apagamos el LED
    delay(Tiempo); // esperamos 100 ms
    digitalWrite(LED2, HIGH); // encendemos el LED
    digitalWrite(LED1, LOW); // apagamos el LED
    delay(Tiempo); // esperamos 100 ms
}
```

## EJERCICIO 2: USART

**Objetivo:** En el ejercicio anterior pudimos ver los elementos necesarios de un código ARDUINO típico, ahora pondremos en marcha la USART, y usaremos la propia USART por donde programamos a la placa Arduino.

Esto nos permitirá enviarles mensajes a la PC, un “HOLA MUNDO”

### El Hardware necesario:

Usamos el propio hardware de la PLACA, no necesitamos extras. La placa CHIPKIT Uno32 tiene conectada la USART a un circuito integrado (CHIP) denominado FT232, el cual es fabricado por la firma FTDI. Este circuito es un “puente USART-USB”, mediante el cual se baja el código a la placa CHIPKIT o podemos comunicarnos con la PC. Una vez bajado el código, la interfaz FT232 queda liberada y podemos usarla para comunicarnos con la PC.

Sin embargo se advierte que la operación no es instantánea, puede llevar unos dos segundos de retardo, ya que una vez bajado el código el BOOTLOADER debe reiniciar y entregarle el control al programa de aplicaciones.

Para poder ver el mensaje que enviamos se hace necesario usar el programa HYPETERMINAL (HTPE) que viene con Windows XP, pero que ha sido removido del VISTA y SEVEN. Sin embargo puede bajarse gratuitamente de internet. En este caso recomendamos la versión HTPE6.3.

### El Programa:

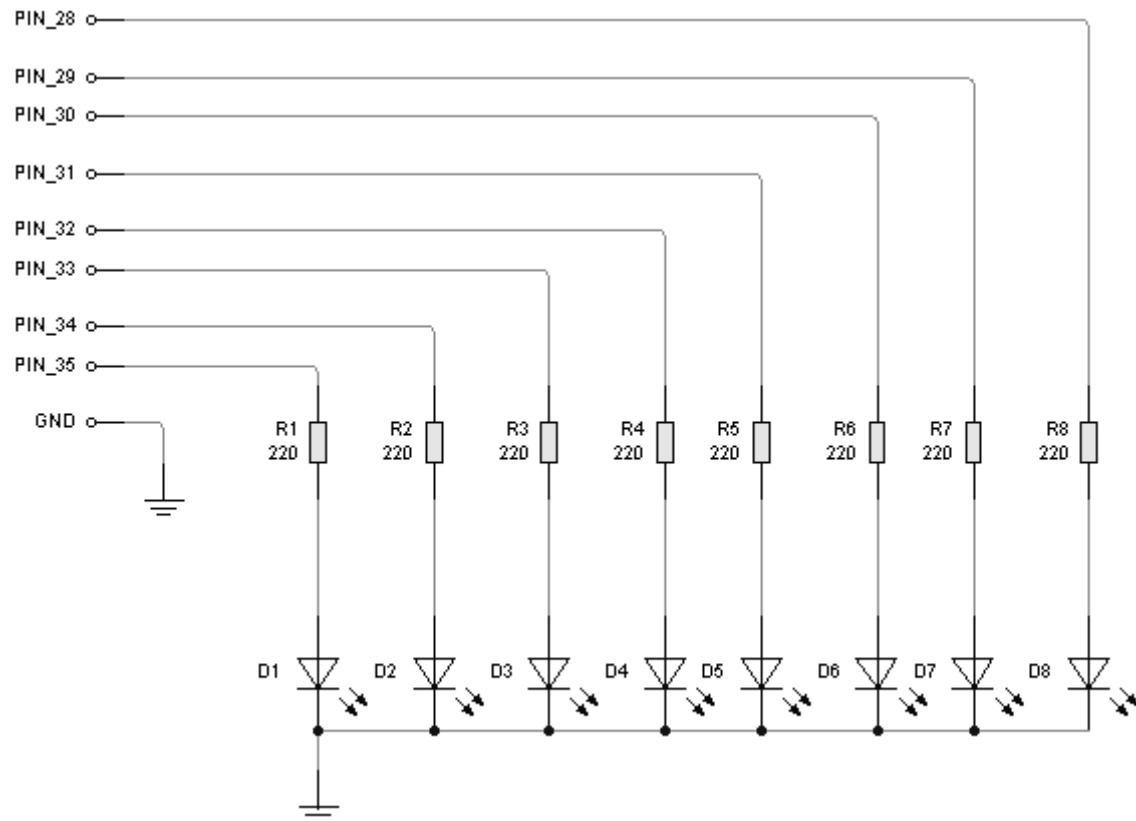
```
void setup() {  
    Serial.begin(9600);//Baud Rate a 9600bps  
}  
  
//La siguiente función es la Principal  
  
void loop()  
{  
    Serial.println("HOLA MUNDO");//enviamos el mensaje  
    delay(100);//esperamos 100ms  
}
```

Nota observe que usamos println para incluir el retorno de carro y la nueva linea para hacer el scroll.

### EJERCICIO 3: SECUENCIA DE LEDS

**Objetivo:** En este nuevo ejercicio vamos a practicar las estructuras del lenguaje arduino y su uso en la aplicación de un secuenciador de LEDs el cual varía la velocidad a la cual se desplaza el encendido de los LEDs de forma incremental.

**El Hardware necesario:**



**El Programa:**

```
//declaramos una variable de uso global  
char t;  
//configuramos el hardware
```

```

void setup() {
    //configuramos los puertos de salida
    pinMode(LED1,OUTPUT);
    pinMode(LED2,OUTPUT);
    pinMode(LED3,OUTPUT);
    pinMode(LED4,OUTPUT);
    pinMode(LED5,OUTPUT);
    pinMode(LED6,OUTPUT);
    pinMode(LED7,OUTPUT);
    pinMode(LED8,OUTPUT);
}

//loop principal
void loop() {
    for(t=28;t<36;t++) //Creamos un ciclo FOR ascendente
    {
        digitalWrite(t,HIGH); //si es así encendemos el LED
        delay(100);//esperamos 100 ms
        digitalWrite(t,LOW); //apagamos el LED indicado
        delay(100);//esperamos 100 ms
    }
    for(t=36;t>28;t--) //Creamos un ciclo FOR descendente
    {
        digitalWrite(t,HIGH); //si es así encendemos el LED
        delay(100);//esperamos 100 ms
        digitalWrite(t,LOW); //apagamos el LED
        delay(100);//esperamos 100 ms
    }
}

```

Nota: Observe que usamos la variable **t** del ciclo **for** como un puntero para indicar en cada bucle que led debe encenderse y apagarse.

## EJERCICIO 4: MANEJANDO UN LCD

### Objetivo:

En este nuevo ejercicio vamos a poner en marcha un LCD del tipo inteligente de 2 x 16 caracteres, para lo cual vamos a usar una librería de las que incorpora ARDUINO para el manejo de periféricos externos.

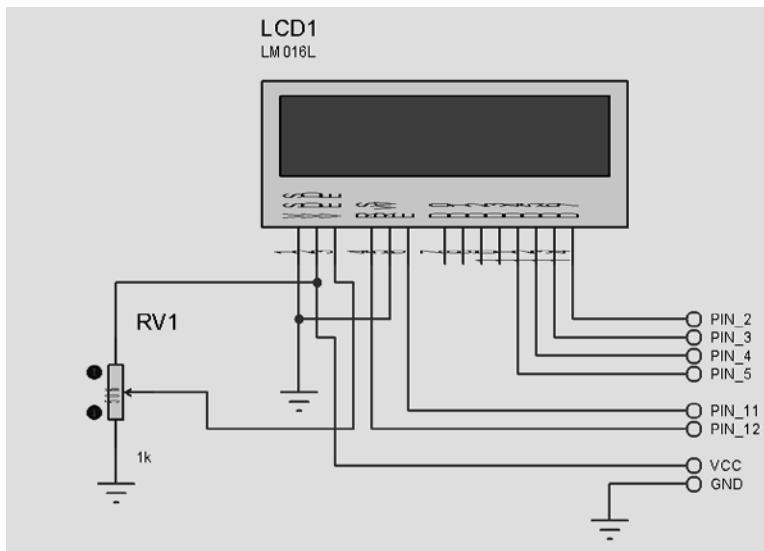
El objetivo es hacer desplazar un mensaje por la pantalla, al mismo tiempo que desplazamos los LEDs del ejercicio anterior. Estamos con esto comenzando a integrar los conocimientos que hemos puesto en marcha hasta ahora

**El Hardware necesario:** Obviamente vamos a necesitar no solo los LEDs que usamos en la práctica anterior, sino que además vamos a incorporar un LCD 2 x16. Yo le recomiendo el uso de un LCD del tipo fondo azul, letras en blanco, esto es porque el LCD de este tipo son los de bajo consumo, es decir que la luz de fondo, solo va a necesitar de una corriente de unos 10 a 15 ma, como si fuera un LED, lo cual nos permite alimentar el LCD de la misma placa sin cargar el USB ni tener que utilizar una fuente externa.

Además vamos a necesitar un PRESET de 1K lineal, para poder ajustar el contraste del LCD. Una advertencia muy importante: debido a que los LCD que normalmente se consiguen en mercado latinoamericano, son de 5V, vamos a conectar el PIN R/W del LCD directamente a masa. Es decir que el LCD solo lo vamos a usar como interfaz de salida y en ningún caso vamos a explorar su estado de ocupado, por lo cual entre caracter y caracter, la librería va a esperar un tiempo (algunos milisegundos).

Esto es necesario porque el PIC32 se alimenta con 3V y cuando los puertos trabajan como entradas, la tensión no puede superar los 3.6V, de otra forma se dañan. Resulta que el LCD no tiene problemas para leer datos enviados por el PIC hacia él, pues su “fan-in” o capacidad de lectura de entrada es equivalente a la de un integrado TTL, pudiendo reconocer como un uno a cualquier tensión que supere los 2,5V. Sin embargo su fan-out, o capacidad de salida, supera los 4V lo cual puede dañar las entradas del PIC32, si el LCD le envía algún dato al PIC por encontrarse en modo lectura.

Por ello para evitar inconvenientes fijamos el PIN R/W a masa, seteando así al LCD en modo escritura.



### El Programa:

```
/*
```

```
Uso de la Libria- LiquidCrystal para control de LCDs Alfanuméricos:
```

La Librería LiquidCrystal soporta cualquier LCD con un controlador compatible con el HD44780

El programa Imprime en pantalla un mensaje y lo hace recircular

La conexión eléctrica usada es la siguiente:

- \* LCD RS conectado al pin digital 12
- \* LCD Enable conectado al pin digital 11
- \* LCD D4 conectado al pin digital 5
- \* LCD D5 conectado al pin digital 4
- \* LCD D6 conectado al pin digital 3
- \* LCD D7 conectado al pin digital 2
- \* LCD R/W conectado a masa

Esta conexión es pasada a la librería por la función

```
LiquidCrystal lcd(PIN_RS,PIN_E, PIN_D4, PIN_D5, PIN_D6, PIN_D7);
```

```
*/
```

```

// include de la librería:
#include <LiquidCrystal.h>
#define LED1 28
#define LED2 29
#define LED3 30
#define LED4 31
#define LED5 32
#define LED6 33
#define LED7 34
#define LED8 35
#define CLS_1 lcd.print("      "); //CLS Linea1
//definimos una variable global
char t;
// inicializamos los pines de conexión
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

//configuramos el hardware
void setup() {
    // configuramos el número de COLUMNAS y FILAS del LCD:
    lcd.begin(16, 2);
    //configuramos los puertos de salida
    pinMode(LED1,OUTPUT);
    pinMode(LED2,OUTPUT);
    pinMode(LED3,OUTPUT);
    pinMode(LED4,OUTPUT);
    pinMode(LED5,OUTPUT);
    pinMode(LED6,OUTPUT);
    pinMode(LED7,OUTPUT);
    pinMode(LED8,OUTPUT);
}

//loop principal
void loop() {
    for(t=0;t<17;t++) //creamos un ciclo FOR
    {
        lcd.setCursor(t,0);//ponemos el cursor en linea 1
        lcd.print("Microchip 2013");//imprimimos en el lcd
        lcd.setCursor(1,1);//re posicionamos el cursor
        lcd.print("Arduino PIC32");//imprimimos en linea 2
        if(t<8)//preguntamos si es menor a 8
            digitalWrite((t+28),HIGH); //si es encendemos LED
        delay(300);//esperamos 300 ms
        lcd.setCursor(t,0);//repositionamos el cursor
    }
}

```

```
if(t<8)//si es menor a 8  
    digitalWrite((t+28),LOW); //apagamos el LED  
    CLS_1;//borramos linea 1  
}  
}
```

## EJERCICIO 5: LEYENDO DOS ENTRADAS ANALÓGICAS

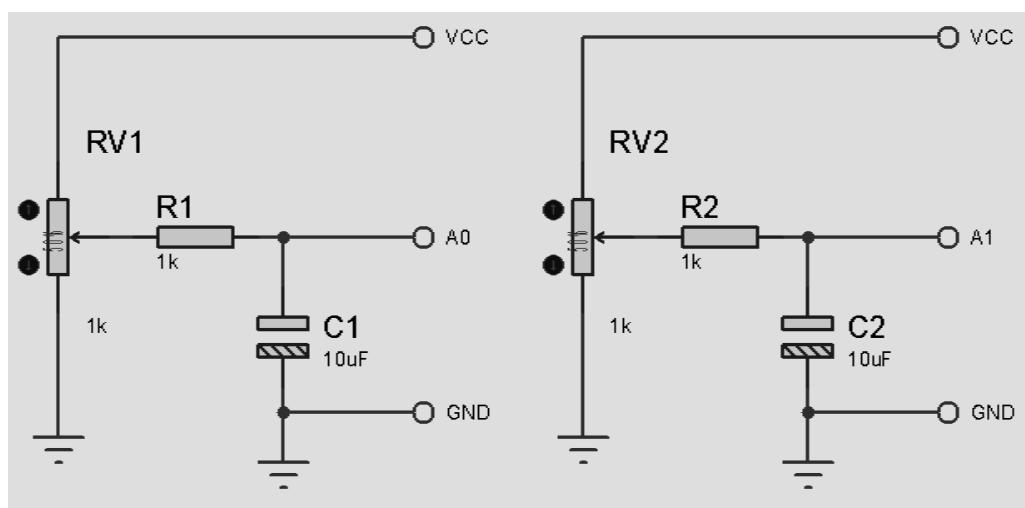
**Objetivo:** Llegamos al punto vibrante ya que vamos ahora a leer una señal analógica, lo cual es fundamental en cualquier sistema de control, esto nos permite “conectar nuestra placa arduino con el mundo real”. En este ejercicio seguimos integrando el conocimiento que traemos, es decir que vamos a usar el LCD para mostrar “lo que ve nuestra placa Arduino”.

### El Hardware necesario:

No desconecte el LCD porque lo va a usar como en el ejercicio anterior, pero ahora le vamos a agregar dos presets mas unos filtros RC lo cual va a entregarnos una tensión entre 0 y 3V, tensión máxima que podemos aplicarle a las entradas analógicas.

Un comentario importante que usted debe saber es que el PIC32 con su conversor A/D va a transformar esa tensión externa en un número el cual va a variar entre 0 y 1023 a fondo de escala, es decir cuando llegue a los 3V.

Esto determina que la resolución digital de entrada sea de unos 3mv de entrada, súper sensible para la mayoría de las aplicaciones industriales, y esto es porque el conversor internamente es de 10 bit.



El filtro RC (1K-10uF) es para evitar fluctuaciones en la lectura del valor analógico debido al riple de la fuente de 3V y a la alta sensibilidad del ADC.

Esas fluctuaciones son molestas sobre todo en los dígitos menos significativos.

### **El Programa:**

```
/*
```

El programa Imprime en pantalla el valor del ADC

La conexión eléctrica usada para el LCD es la siguiente:

- \* LCD RS conectado al pin digital 12
- \* LCD Enable conectado al pin digital 11
- \* LCD D4 conectado al pin digital 5
- \* LCD D5 conectado al pin digital 4
- \* LCD D6 conectado al pin digital 3
- \* LCD D7 conectado al pin digital 2
- \* LCD R/W conectado a masa

Esta conexión es pasada a la librería por la función

```
LiquidCrystal lcd(PIN_RS,PIN_E, PIN_D4, PIN_D5, PIN_D6, PIN_D7);  
*/
```

// include de la librería:

```
#include <LiquidCrystal.h>  
#define LED1 28  
#define LED2 29  
#define LED3 30  
#define LED4 31  
#define LED5 32  
#define LED6 33  
#define LED7 34  
#define LED8 35  
  
#define CLS_1 lcd.print("      "); //CLS Linea 1  
//variables que contienen los valores analógicos  
int adc0,adc1;  
// inicializamos los pines de conexión  
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  
  
//configuramos el hardware  
void setup() {  
    // configuramos el número de COLUMNAS y FILAS del LCD:  
    lcd.begin(16, 2);  
}  
  
//loop principal  
void loop() {  
    adc0=analogRead(A0);//leemos el canal analogico 0
```

```
adc1=analogRead(A1);// leemos el canal analogico 1
lcd.setCursor(1,0);//ponemos el cursor
lcd.print("ADC PIC32");//imprimimos en el lcd linea 1
lcd.setCursor(1,1);//re posicionamos el cursor
lcd.print("A0=");
lcd.print(adc0);//el valor del canal analogico 0
lcd.print(" ");
lcd.print("A1=");
lcd.print(adc1);//el valor del canal analogico 1
lcd.print(" ");
delay(100);
}
```

## EJERCICIO 6: CONTROL PIROMÉTRICO

### Objetivo:

Con todo lo que ya se ha practicado podemos finalmente construir una aplicación completa.

Vamos a realizar un control de temperatura. Para ello integraremos todo lo que aprendimos en el capítulo hasta el momento.

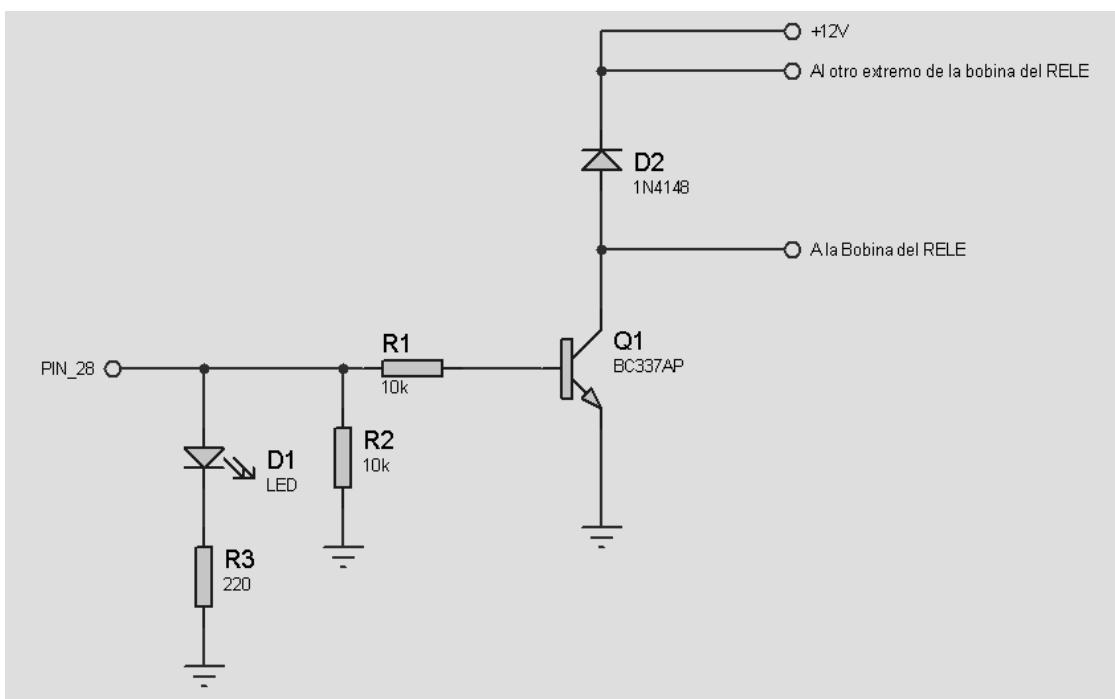
Como se ha visto ARDUINO es una gran herramienta la cual nos facilita el emplazamiento de cualquier aplicación sobre un fantástico microcontrolador, el PIC32.

### El Hardware necesario:

Ya hemos empleado todo lo necesario para nuestro control, solo nos queda agregar 3 elementos más, un sensor de temperatura, vamos a usar el MCP9700 de Microchip, y un transistor BC337 con una resistencia de 10K para accionar un relé.

El MCP9700 es un sensor de Microchip que puede medir temperaturas tan bajas como -40ºC y tan altas como 125ºC. Es ideal para un PIC32 pues puede ser alimentado con tensiones desde 2,5V en adelante hasta los 5V.

A 0ºC el sensor nos entrega 0,5V, y esta tensión se incrementa o decrementa con la temperatura a razón de 10 mV por ºC. Es decir que en -40ºC nos entrega 0,1V y en 125ºC nos entrega 1,75V.



## El Programa

```
/*
```

El programa permite controlar la Temperatura

La conexión eléctrica usada para el LCD es la siguiente:

- \* LCD RS conectado al pin digital 12
- \* LCD Enable conectado al pin digital 11
- \* LCD D4 conectado al pin digital 5
- \* LCD D5 conectado al pin digital 4
- \* LCD D6 conectado al pin digital 3
- \* LCD D7 conectado al pin digital 2
- \* LCD R/W conectado a masa

Esta conexión es pasada a la librería por la función

```
LiquidCrystal lcd(PIN_RS,PIN_E, PIN_D4, PIN_D5, PIN_D6, PIN_D7);
```

```
*/
```

// include de la librería:

```
#include <LiquidCrystal.h>
```

```
#define RELAY 28
```

```
#define CLS_1 lcd.print(" "); //cls linea 1
```

//definimos las variables globales

```
int adc0,adc1,adc2;
```

```
float grados,th tl;
```

// inicializamos los pines de conexión

```
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

//definimos la funcòn que calcula la temperatura

```
float ConverGrados(int raw);
```

//definimos la función que imprime en pantalla

```
void ImprimeLCD(void);
```

//definimos la función que lee las entradas analógicas

```
void LeerAnalogicos(void);
```

//definimos la función control de temperartura

```
void Control(void);
```

//definimos la función Imprime UART

```

void ImprimeUART(void);

//configuramos el hardware

void setup() {
  // configuramos el número de COLUMNAS y FILAS del LCD:
  lcd.begin(16, 2);
  //configuramos salida digital
  pinMode(RELAY,OUTPUT);
  //Iniciamos UART
  Serial.begin(9600);
}

//loop principal
void loop() {
  LeerAnalogicos();
  ImprimeLCD();
  ImprimeUART();
  Control();
  delay(100);
}

float ConverGrados(int raw)
{
  float temperatura;
  temperatura=(float)((raw*(3.3/1024.0))-0.5)/0.01;
  return temperatura;
}

void Control(void)
{
  ifadc2>adc0)
    digitalWrite(RELAY,HIGH);
  ifadc2<adc1)
    digitalWrite(RELAY,LOW);
}

void LeerAnalogicos(void)
{
  adc0=analogRead(A0);//leemos el canal analogico 0
  adc1=analogRead(A1);// leemos el canal analogico 1
  adc2=analogRead(A2);// leemos el canal analogico 2
}

```

```
void ImprimeUART(void)
{
    Serial.print("Grados="); //imprimimos en el lcd linea 1
    Serial.print(grados); //el valor de la temperatura
    Serial.print("\t");
    Serial.print("TH="); //imprimimos en el linea 2
    Serial.print(th); //el valor del canal analogico 0
    Serial.print("\t"); //borramos exceso
    Serial.print("TL="); //imprimimos en el linea 2
    Serial.print(tl); //el valor del canal analogico 1
    Serial.print("\t");
    Serial.print("\r");
}
```

```
void ImprimeLCD(void)
{
    lcd.setCursor(0,0); //ponemos el cursor en el lcd
    lcd.print("Grados="); //imprimimos en el lcd linea 1
    grados=ConverGradosadc2(); //convertimos a grados
    lcd.print(grados); //el valor de la temperatura
    lcd.print(" ");

    lcd.setCursor(0,1); //re posicionamos el cursor
    lcd.print("TH="); //imprimimos en el linea 2
    th=ConverGradosadc0();
    lcd.print(th); //el valor del canal analogico 0
    lcd.print(" "); //borramos exceso

    lcd.print("TL="); //imprimimos en el linea 2
    tl=ConverGradosadc1();
    lcd.print(tl); //el valor del canal analogico 1
    lcd.print(" ");
}
```

La información de la temperatura se puede ver en el LCD y también es enviada a la PC vía el HYPERTERMINAL. Por otra parte los PRESET que usamos en el ejercicio anterior servirán ahora para que carguemos los SET POINTS de la Temperatura, es decir poner a qué temperatura se acciona el RELE y a qué temperatura se apaga el mismo.

El programa ha sido estructurado en funciones para favorecer luego su posterior mantenimiento.



# **CONTROL INTELIGENTE DE LEDS**



## Iluminación LED

Hoy en día la utilización de los LEDs se ha masificado en todo tipo de aplicación donde exista la emisión de luz. Es habitual ver a los LEDs cumpliendo la función de iluminación que normalmente se realizaba una fuente de luz incandescente o fluorescente convencional. Al utilizar LEDs con MCU se puede agregar inteligencia a los sistemas de iluminación.

	Traditional Options	Microchip Human Interface Options
	<p><b>Legacy Control</b></p> <p><b>Non-Intelligent Lighting Control</b></p> <ul style="list-style-type: none"> <li>Mechanical Interface</li> <li>Limited Luminaire Control</li> <li>No User Feedback</li> <li>No Communication</li> <li>No Product Differentiation</li> </ul> 	<p><b>Intelligent Control</b></p> <p><b>mTouch™ Capacitive Touch</b></p> <p><b>all PIC® MCU families</b></p> <ul style="list-style-type: none"> <li>Metal Over Cap capability</li> <li>Projected capacitive capability</li> </ul> <p><b>Segmented LCD - PIC16, PIC18, PIC24</b></p> <ul style="list-style-type: none"> <li>Up to 480 segments</li> <li>Low power display modes</li> <li>Contrast Control</li> </ul> <p><b>Graphics - PIC24, PIC32, dsPIC® DSC</b></p> <ul style="list-style-type: none"> <li>Integrated graphics controllers</li> <li>Direct drive for QVGA and W-QVGA</li> </ul> 

La iluminación de Estado Sólido (IES) puede reemplazar la tecnología existente de luz, ya sea incandescente, fluorescente o lámpara fluorescente del cátodo frío (LFCC) usada en los backlight de las pantallas LCD y Plasma.

Si vemos una comparativa de funcionamiento de los dispositivos mencionados vemos que la lámpara construida con led posee una vida útil mayor a 5 años y un consumo muchas veces menor.

Tipo de Luz	Operación 1 Hora [WattxHora]	Energía de encendido Vs. Operacional normal (seg)	Funcional >5 años Ciclos de Potencia
Bombilla LED blanco	1	1.3	SI
Fluorescentes Compactas (FC)	10	0.015	NO
Fluorescentes	10	23	NO
Halógenas	70	0.5	NO
Halógenas	90	0.36	NO

Si además analizamos la relación consumo Vs potencia lumínica entregada vemos que el led es el que mejor relación nos presenta frente a los otros dispositivos

<b>Light Output</b>	<b>LEDs</b>	<b>CFLs</b>	<b>Incandescents</b>
Lumens	Watts	Watts	Watts
450	4 - 5	8 - 12	40
300 - 900	6 - 8	13 - 18	60
1100 - 1300	9 - 13	18 - 22	75 - 100
1600 - 1800	16 - 20	23 - 30	100
2600 - 2800	25 - 28	30 - 55	150

Observando los cuadros anteriores podemos deducir las siguientes ventajas:

Bajo consumo: Una lámpara LED se alimenta a baja tensión, consumiendo así poca potencia.

*Ejemplo: Una lámpara halógena de 50W de potencia ilumina 25 lumens/W consiguiendo un total de 1250 lumens. Para conseguir la misma iluminación con una lámpara de LEDs vamos a necesitar 179 LEDs (utilizando LEDs de alta luminosidad que iluminan 7 lumens/unidad), de esta forma tendremos la misma iluminación con ambas lámparas, sin embargo nuestro consumo con la lámpara de LEDs va ser 4 veces menor, ya que sólo consumirá 13W.*

Funcionamiento en baja tensión: Se alimentan con corriente continua, adaptándose perfectamente a la mayoría de las fuentes de alimentación existentes, de esta manera se reduce al mínimo los posibles riesgos de electrocución. (Rangos de tensión más comunes 2-3,6v).

Baja temperatura: El LED se alimenta con baja tensión, consumiendo así poca energía y por lo tanto emitiendo poco calor. Esto es debido a que el LED es un dispositivo que opera a baja temperatura en relación con la luminosidad que proporciona. Los demás sistemas de iluminación en igualdad de condiciones de luminosidad que el LED emiten mucho más calor.

Emite luz monocromática: ésta es una de las más importantes características, la luz emitida es directamente en la longitud de onda del color requerido, por lo que no existe la transformación de luz en calor como ocurre con otros dispositivos.

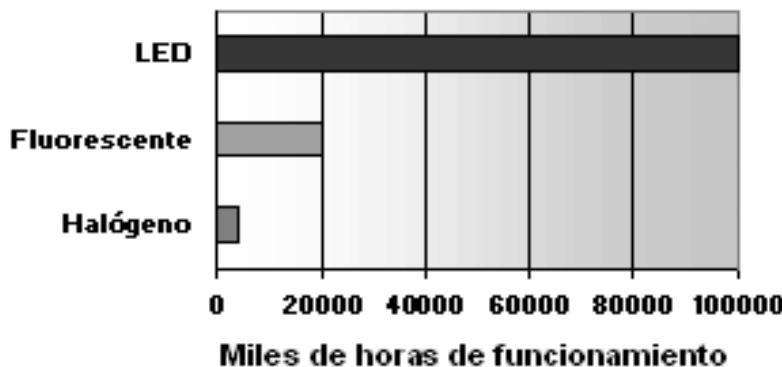
*Esta diferencia en la emisión de luz, entre la incandescencia más el filtro y el diodo LED, hace que sea más eficiente, ya que toda la luz emitida es aprovechada en la iluminación.*

Mayor rapidez de respuesta: El LED tiene una respuesta de funcionamiento mucho más rápida que el halógeno y el fluorescente, del orden de algunos microsegundos. Ello lo hace ideal para funcionar con un estrobo, aumentando así las prestaciones.

Luz más brillante: En las mismas condiciones de luminosidad que sus rivales, la luz que emite el LED es mucho más nítida y brillante.

**Sin fallos de iluminación (durabilidad):** Absorbe las posibles vibraciones a las que pueda estar sometido el equipo sin producir fallos ni variaciones de iluminación. Esto es debido a que el LED carece de filamento luminiscente evitando de esta manera las variaciones de luminosidad del mismo y su posible rotura.

**Mayor vida útil y fiabilidad:** La vida de un LED es muy larga en comparación con los demás sistemas de iluminación. La degradación de la luz es mínima en relación a la de halógenos y fluorescentes. Utilizándolo por 10 horas diarias, podrían utilizarse hasta por más de 13 años, posee de 5 a 10 veces más de vida útil que las lámparas tradicionales de sodio y mercurio.

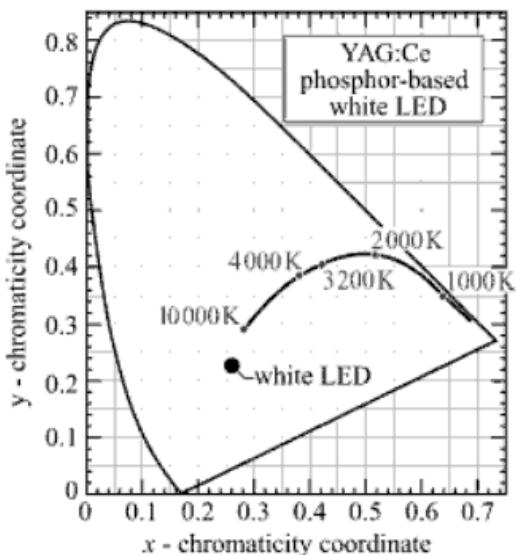
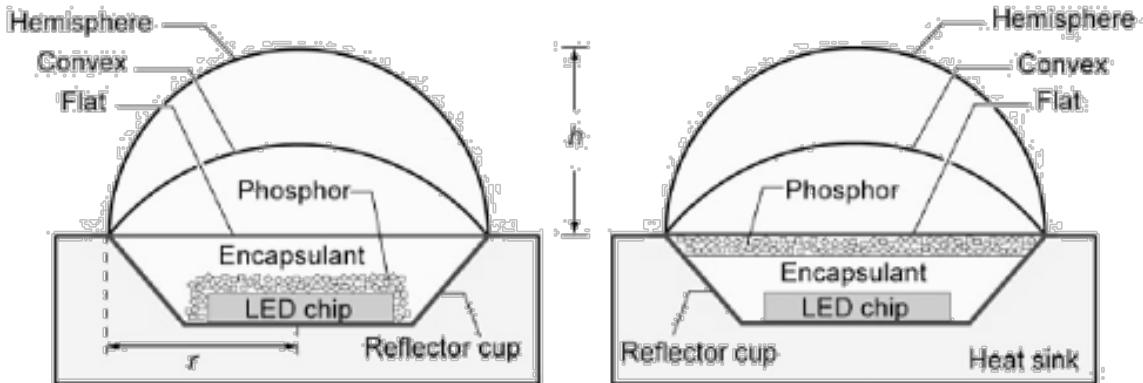


Podemos observar la vida útil de cada dispositivo de iluminación.

Una de las desventajas que presenta las lámparas a LEDs, es el precio, el mayor inconveniente que tiene el LED sin duda es su precio. Pero si evaluamos sus múltiples e inmejorables condiciones de funcionamiento, y sobre todo, su larga vida en comparación con los demás sistemas de iluminación estamos en condiciones de afirmar que es la inversión más sensata, eficaz y rentable que podemos hacer.

En la actualidad se han desarrollados innumerables luminarias con leds blancos, éstos poseen distintas tonalidades.

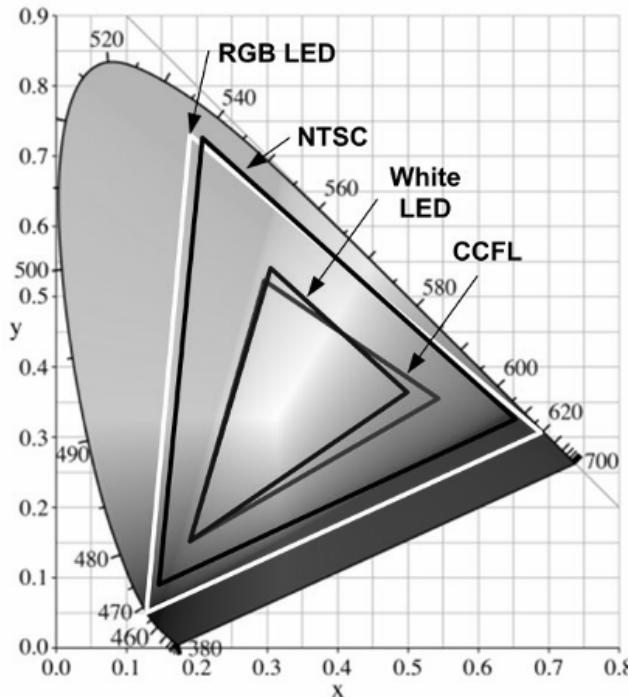
Para obtener LEDS de alta potencia podemos combinar LEDs azules con un fósforo (convertidor de longitud de ondas). El LED blanco obtenido es un diodo azul cubierto de fósforo para convertir una parte de la luz azul y amarillo claro. Este espectro combinado se percibe como luz blanca. Este método de fabricación es simple de implementar, de bajo costo, baja resistencia térmica, compacto y no posee control de deriva del color a largo plazo.



La ubicación de la luz blanca en el diagrama cromático sugiere que el color emitido es blanco azulado.

Otro método de obtener los LEDs blancos es mezclar LED monocromáticos Multicolores (RGB, RGBA) en las proporciones adecuadas. De esta manera podemos obtener una gama de luz visible mayor comparada con una luz fluorescente de cátodo frío (CCFL), que tienen un limitado espectro de color y falta de viveza. CCFL hace exhibición de alrededor del 80 por ciento de la National Television System Committee (NTSC), mientras que los colores RGB definidos pueden revelar hasta 130 por ciento del espectro de color NTSC - permitiendo una representación más exacta de las imágenes en la pantalla. Por otra parte, el mayor espectro posible de color, se consigue mediante el uso de tres fuentes de luz monocromática como el rojo, azul y verde.

En la figura siguiente podemos ver las distintas gamas cromáticas obtenidas con los LEDs:



Una de las características a tener en cuenta con los LEDs es la variación de la potencia de salida con el tiempo, la temperatura y las variaciones de fabricación.

También es importante mencionar que se obtienen variaciones espectrales (de la longitud de onda), o sea, la tonalidad del color con la corriente, la temperatura y las variaciones de fabricación.

Los efectos del envejecimiento se perciben como un cambio en la emisión de luz con el tiempo. En algunos casos la salida inicial se incrementa, luego decrece durante el transcurso de su vida útil. Podemos decir que la potencia disminuye generalmente cerca de 50% del valor inicial después de 50.000 horas de uso.

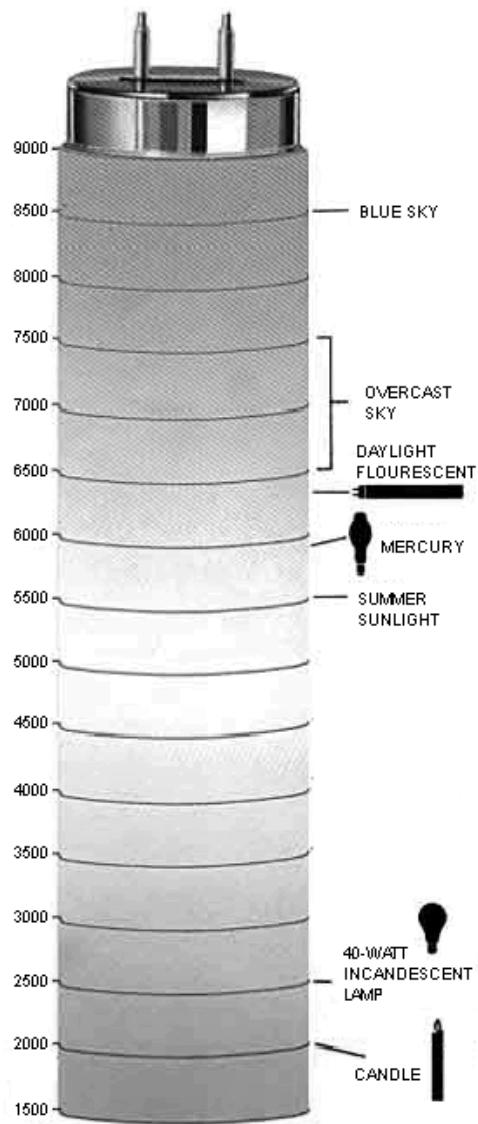
#### El “BLANCO” de los LEDs

*Los colores vistos a la luz de la vela, no son los mismos que se ven durante el día.*

- Elizabeth Barrett Browning, 1844

Para medir el “blanco” de luz, se utiliza un sistema de medición de temperatura. Este sistema se refiere a la temperatura a la que habría que calentar un cuerpo teórico “cuerpo negro” en grados Kelvin (K) para producir la luz del mismo color visual.

El rango visible en que se irradia la energía comienza con el rojo, cambiando de color hacia el naranja, blanco, y, finalmente, blanco azulado.



La TCC (Temperatura de color correlacionada) es una medida científica para describir el nivel de "calidez" o "frialdad" de una fuente lumínica. Se basa en el color de la luz emitida por una fuente incandescente. Al calentar una pieza de metal (un radiador de cuerpo negro teórico) cambia de color rojizo a naranja, amarillo, blanco, blanco azulado. El color de la luz emitida por un objeto incandescente depende sólo de la temperatura. Podemos usar esta medida para describir el color de una fuente de luz por su "temperatura de color".

Cuando decimos que una lámpara tiene una temperatura de color de 3.000 grados Kelvin, significa que un metal ardiente a 3.000 grados Kelvin produciría una luz del mismo color

que la lámpara. Si el metal se calienta hasta 4.100 grados Kelvin, genera una luz mucho más blanca. La luz solar directa corresponde a unos 5.300 grados Kelvin, mientras que la luz diurna, mezclada con la luz del cielo, es de unos 6.000 grados Kelvin o más.

*Una lámpara incandescente convencional tiene un filamento a 2.700 grados Kelvin, y por definición, una temperatura de color de 2.700 grados Kelvin*

En resumen:

Muchas fuentes de luz, tales como lámparas fluorescentes emiten luz principalmente por procedimientos distintos de la radiación térmica. Esto significa que la radiación emitida no sigue la forma de un espectro de cuerpo negro. Estas fuentes se asignan lo que se conoce como temperatura de color correlacionada (TCC)

#### Tipos de LEDs

Hay tres tipos principales de LEDs. La mayoría de ellos son familiares y son de baja potencia utilizados en señalización (on/off, alarmas, ...) y solo requieren pocos mA de corriente para funcionar.

Los LEDs de media potencia por lo general se utilizan en iluminación interior, la corriente de funcionamiento se encuentra en un rango de 20 mA a 150 mA

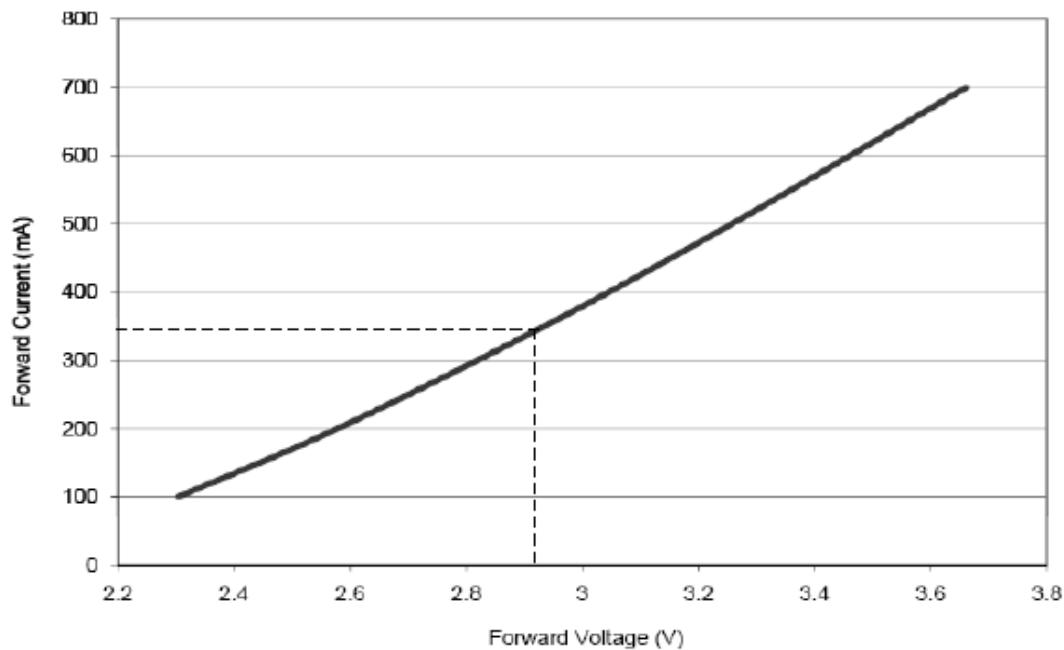
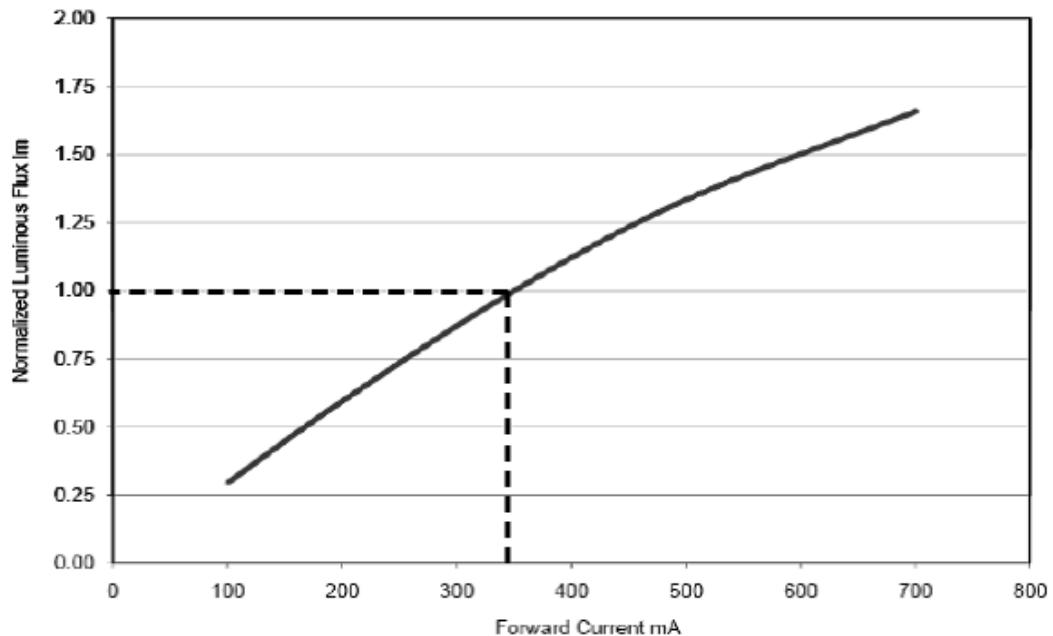
LEDs de alta potencia o de alto brillo son aquellos que poseen una corriente directa de 350mA o más y son utilizados generalmente en aplicaciones de exterior y automóviles.

## LED

El término de LED es el acrónimo de Diodo Emisor de Luz. Un LED consiste en un material semiconductor dopado con impurezas para crear una juntura p-n similar a los diodos normales y emitir luz cuando una corriente pasa a través de ella.

Como en un diodo normal, la corriente fluye desde el ánodo (lado p) al cátodo (lado n), esta corriente se denomina corriente directa. La luz de un LED está determinada por la corriente directa que fluye por él.

*Debemos recordar que la salida de luz es medida en flux, la cual es la percepción de potencia de luz emitida y tiene una unidad de lumen (lm). El color de un LED es medido en la escala de (CCT) Correlated Color Temperature (CCT), la cual se mide en grados kelvin (K).*



En función de estas curvas podemos ajustar los parámetros de funcionamiento del LED para obtener el mejor rendimiento.

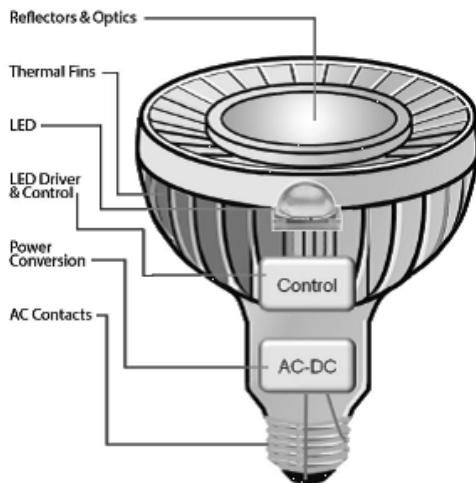
## LED

### Ventajas

- **Gran eficiencia**
- **~25 a 50% menos energía que el tubo fluorescente**
- **Más de 100 lumens/watt**
- **Larga vida >50000 hs**
- **No necesita calentarse, no irradia calor**
- **Bueno para aplicaciones de interior y exterior**

### Desventajas

- **Requiere una solución activa o pasiva para disipar calor**



Similar a un LED tradicional, un Diodo emisor de luz de alto brillo (HBLED) produce luz por medio de la recombinación electrón-hueco que libera fotones de luz. La salida de luz de un LED es una función directa del flujo de corriente, muy poca corriente y la luz se atenuara, demasiada corriente y la vida se acortará.

Un típico controlador de LED es una fuente de alimentación de CC que convierte alimentación de CA o CC para controlar directamente el led. Por medio de la corriente se puede utilizar técnicas como: modulación de ancho de pulso (PWM), u otras variantes

Controladores de LED pueden ser diseñados para ofrecer atenuación y capacidades de mezcla de colores RGBW mediante PWM de alta resolución (o variantes como VFM de modulación de frecuencia variable), o variando la corriente constante.

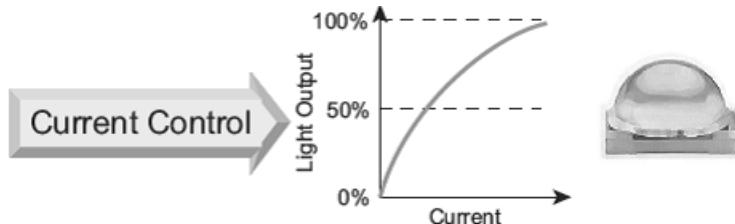
Los principales beneficios de los LEDs son una larga vida útil, durabilidad y eficiencia. Cuando se maneja adecuadamente un LED de potencia puede entregar decenas de miles de horas sin una degradación de la luz entregada.

La eficiencia típica de un LED de potencia, medida en Lúmenes / Watts está entre los 40 / 80 Lúmenes / Watt. Estos valores son decididamente mayores que los de las fuentes incandescentes de luz tradicionales y solamente son superados por las fuentes de luz Fluorescentes. Partiendo de que el LED es un dispositivo de estado sólido, éste puede soportar condiciones desfavorables como golpes y vibraciones sin sufrir daño como sí lo sufrirían los bulbos o filamentos de las fuentes anteriormente citadas.

### Regulación por corriente constante

#### Método de corriente constante

- Salida de luz mantenida por un nivel de I cte.
- Control de atenuación a través de corriente variable.
- Requiere un control de alta resolución.



### Regulación por PWM

#### Método de modulación de corriente

- Control Corriente fija por medio del PWM
- Control de atenuación variando el duty cycle del PWM

Power  
Conversion  
& Control

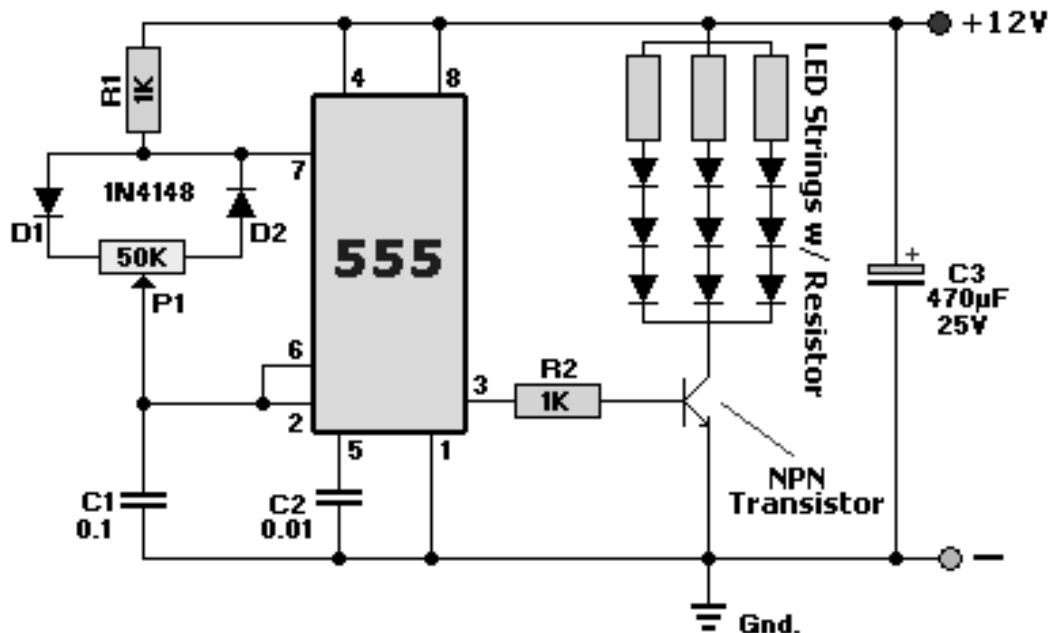
PWM Control



Los LEDs deben manejarse con una fuente de corriente constante. La mayoría de los LEDs tienen un nivel de corriente específico en el cual alcanzan el máximo brillo sin sufrir fallas prematuras. Un LED podría manejarse con un regulador lineal de tensión configurado como una fuente de corriente constante. Sin embargo, esta solución no es práctica para LEDs de alta potencia, debido a la alta disipación de potencia en el circuito del regulador. Una gran cantidad de energía sería disipada en forma de temperatura por el circuito regulador disminuyendo notoriamente el rendimiento del sistema.

Una fuente de alimentación conmutada (Switching - SMPS) provee una eficiencia mucho mayor para manejar el LED.

Una solución para el control es el circuito propuesto a continuación:

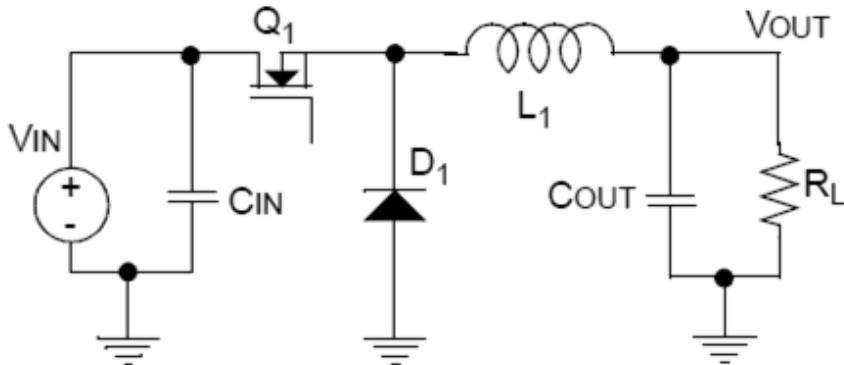


Se debe dimensionar el transistor en función de la potencia a controlar, la cual dependerá del número de leds a conectar. Esta solución no posee una realimentación que controle la corriente por los leds ni un control térmico de los mismos.

## CONVERTIDOR NO SINCRÓNICO

Si deseamos suministrar energía a un simple LED de 10W se requerirá 3A a 3.3V. Un convertidor no-sincrónico es muy simple y puede ser implementado con cualquier PIC

El diodo D<sub>1</sub> es del tipo Schottky y posee una VD de 0.7V. a 3A de corriente la disipación de potencia será aprox. 2W en conducción, lo cual es inaceptable para equipos alimentados a batería.



### Control Eficiente del LED

Los LEDs deben manejarse con una fuente de corriente constante. La mayoría de los LEDs tienen un nivel de corriente específico en el cual alcanzan el máximo brillo sin sufrir fallas prematuras. Un LED podría manejarse con un regulador lineal de tensión configurado como una fuente de corriente constante. Sin embargo, esta solución no es práctica para LEDs de alta potencia, debido a la alta disipación de potencia en el circuito del regulador. Una fuente de alimentación conmutada (Switching - SMPS) provee una eficiencia mucho mayor para manejar el LED.

Un LED tendrá una caída de tensión directa a través de sus terminales para un nivel de corriente dado. La tensión de la fuente de alimentación y la característica de la tensión directa del LED determinarán la topología de la SMPS (**Switch Mode Power Supply**) a utilizar.

Múltiples LEDs pueden conectarse en "series" para así incrementar la caída de tensión directa a la corriente elegida.

Las topologías de circuito SMPS adoptadas para regular la corriente en el LED son las mismas usadas para controlar la tensión en aplicaciones de fuentes de alimentación.

Esta guía de diseño presenta dos tipos de soluciones para el manejo de los LED. La primera, un CI analógico de manejo puede utilizarse independientemente o en forma conjunta con un MCU para agregar "inteligencia". La segunda, la función de manejo del LED puede integrarse en la aplicación del MCU.

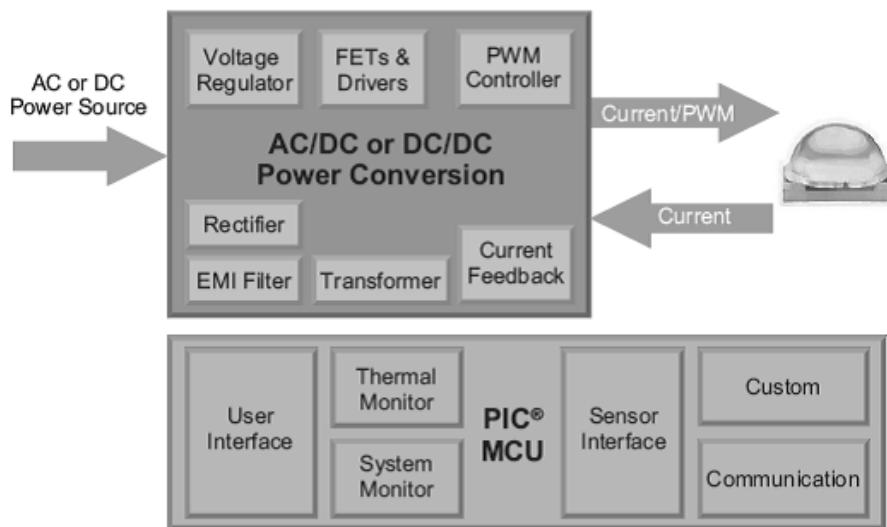
## 1 - PIC conectado a una Fuente de energía simple

Diseño simplificado.

Características configurables.

Modificación simple por medio de actualizaciones de firmware.

Capacidad de control inteligente.



## 2 - SMPS (Switch Mode Power Supply) con PIC MCU & Componentes Analógicos Microchip

Mayor integración MCU

Totalmente configurable

Una mayor eficiencia

Corrección de factor de potencia (PFC)

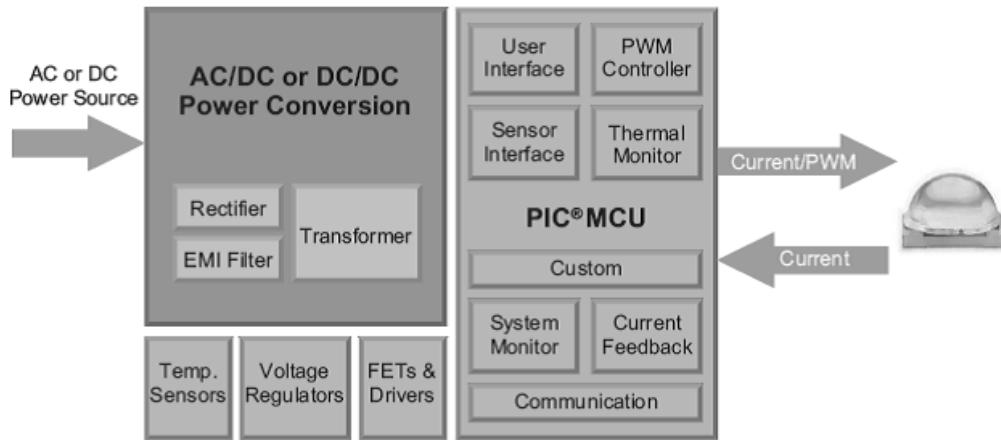
Topologías flexibles

Modificaciones simples por medio de actualizaciones de firmware

Control de lazo cerrado

PWM de alto rendimiento y control de corriente

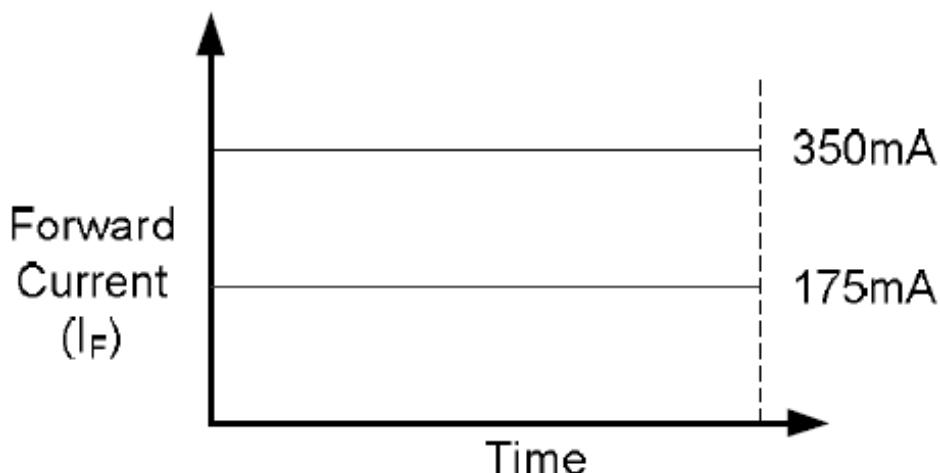
Capacidad de control Inteligente



## Control de brillo

Hay dos formas de controlar el brillo de un LED. El primer método es utilizando una atenuación analógica la cual varía la corriente directa que circula por el LED para ajustar el brillo.

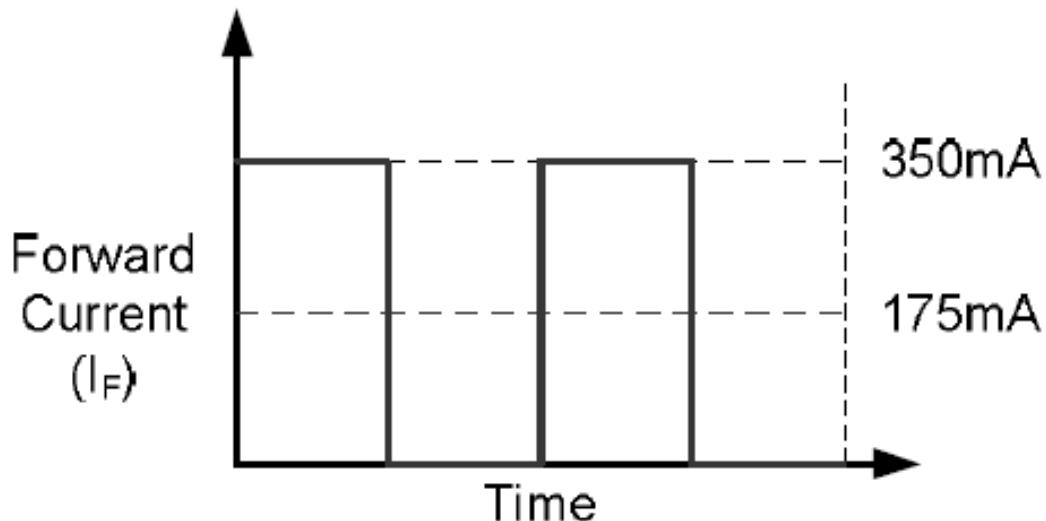
En la figura siguiente se muestra una atenuación analógica donde la corriente directa se reduce para obtener un 50% de brillo. La mayor desventaja de la atenuación analógica es que cuando se ajusta la corriente, el color de la temperatura del LED varía. Esto no es deseable para la mayoría de las aplicaciones.



El segundo método es la técnica de atenuación digital, la cual comuta la corriente directa por períodos cortos de tiempo. El ojo humano promedio no captará el apagado y encendido del led. Este ejemplo también muestra una reducción del 50% del brillo mostrada por la línea punteada.

Con este método estamos reduciendo efectivamente la corriente promedio y al mismo tiempo mantenemos la consistencia del color del LED.

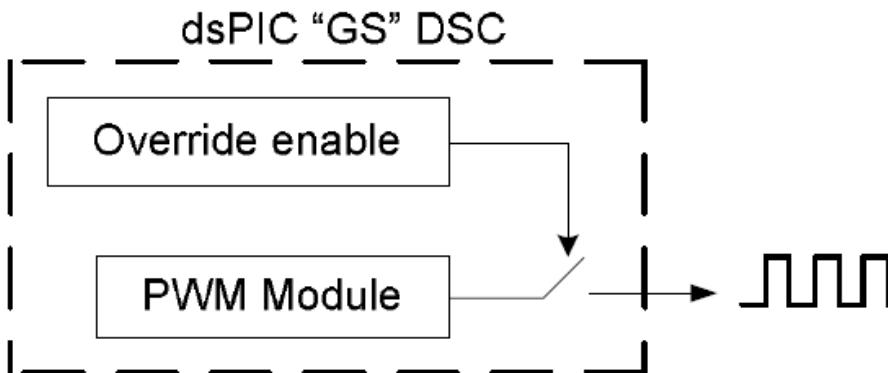
Algo importante es que la atenuación digital posee una frecuencia elevada para evitar la percepción del destello, desde 400Hz a 1,2Khz es una frecuencia típica.



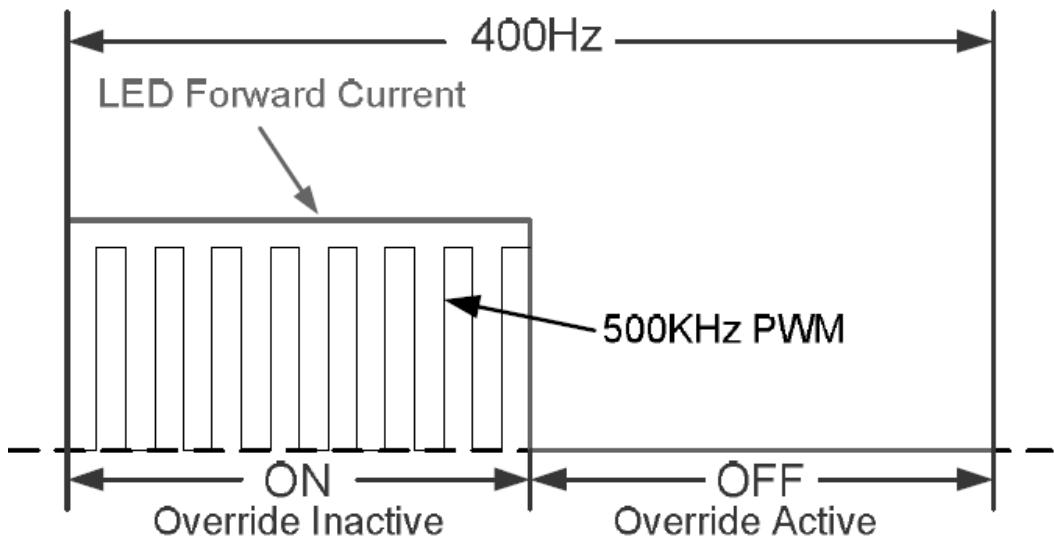
#### PWM – Corriente constante y Atenuación

La alta frecuencia PWM (500 KHz) se utiliza para controlar la corriente y mantenerla constante. Para la atenuación del brillo se utiliza sobre la señal de 500KHz una señal envolvente de 400Hz.

Para controlar digitalmente el brillo del LED, primero creamos una frecuencia de atenuación de 400Hz. Los dsPIC de la serie "GS" poseen una ventaja de PWM, cuando están activos los módulos PWM, éste puede desconectar el modulo PWM del PIN. Esta ventaja, proviene de un contador de 8bits, el cual es utilizado para crear la frecuencia envolvente de 400Hz.



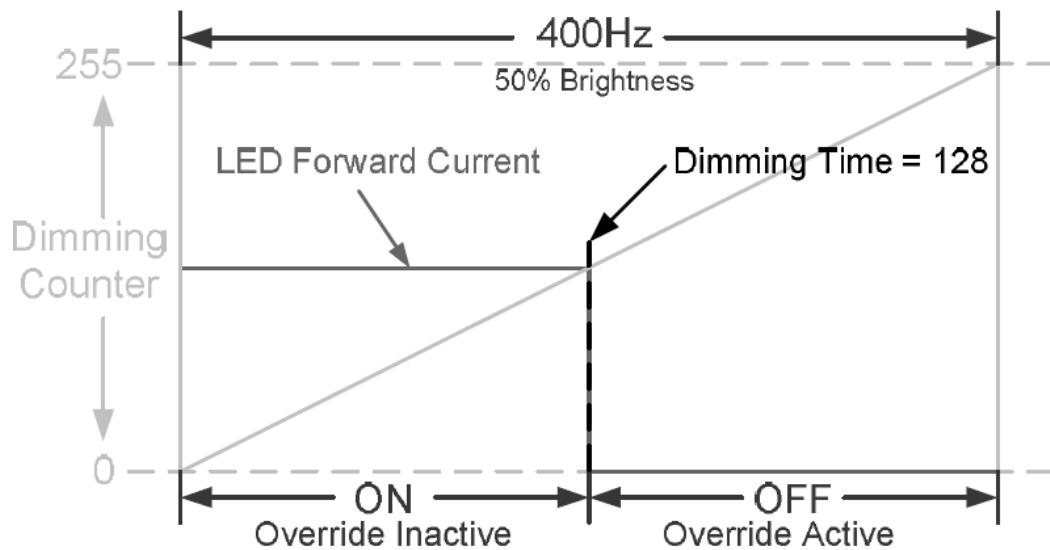
En la figura siguiente podemos ver un periodo de la frecuencia de atenuación de 400Hz. El PWM de 500Khz mostrado es utilizado para regular y mantener la corriente directa constante del LED durante el tiempo de encendido. Al comienzo del periodo de atenuación, el control de salida de PWM está inactivo y el PWM es conectado al pin del dispositivo. Durante el tiempo de apagado, el control de salida está activo, el PWM se desconecta del pin y la corriente que circula por el LED será cero. Este proceso se reiniciara en el inicio del próximo periodo.



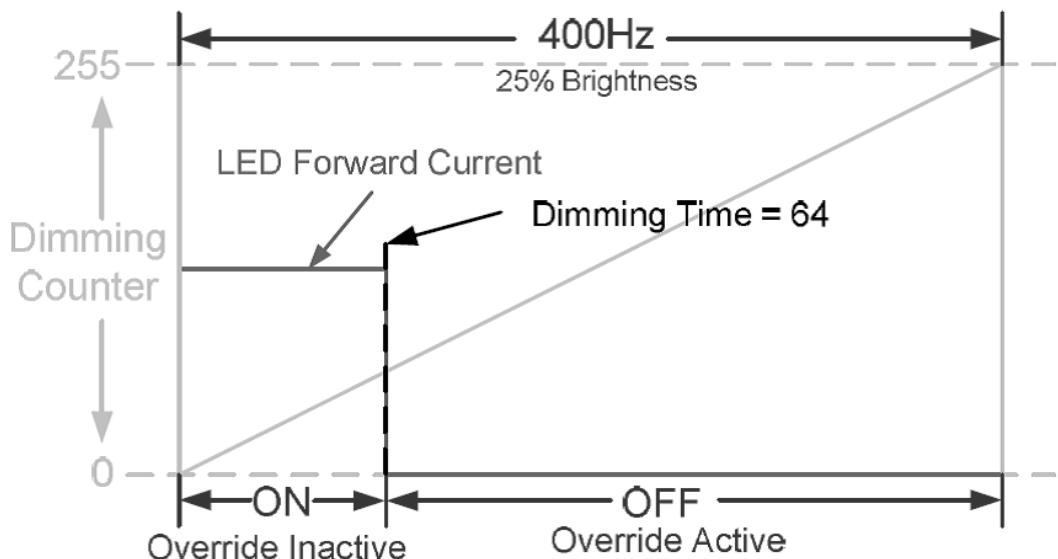
Veamos cómo se crea la frecuencia de atenuación y cómo el punto de anulación es determinado para controlar el brillo. El PWM de 500 KHz Ha sido removido para una visualización más fácil.

La línea diagonal dorada representa un contador de atenuación de 8 bits, el cual se incrementa durante el periodo de atenuación. Una vez que el contador de atenuación alcanza 25 se resetea a 0 para el comienzo del próximo periodo.

EL tiempo de atenuación variable se muestra aquí con un valor igual a 128 y es comparado continuamente con el contador de atenuación para activar el punto de anulación PWM. Como el tiempo de atenuación es 128, y este valor es la mitad del contador, el brillo del LED es el 50%.

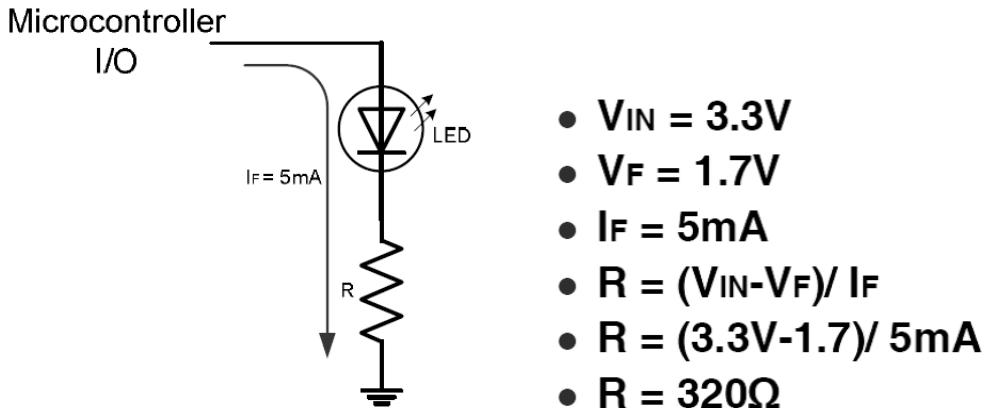


En el siguiente ejemplo, el tiempo variable de atenuación se configura en 64 y como este valor es el 25% del contador de atenuación cuyo rango máximo es 255, el brillo del LED será del 25 % de la intensidad máxima.



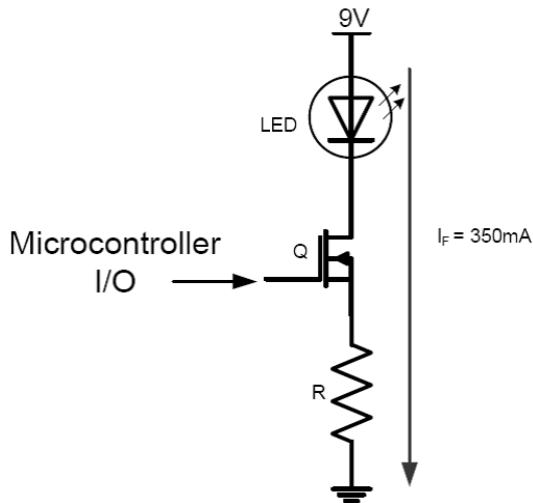
## CÓMO CONTROLAR LOS LEDS DE ALTA INTENSIDAD

Este es un ejemplo de cómo un LED de baja potencia puede ser alimentado. El pin de entrada/salida en un microcontrolador puede fácilmente entregar 5mA de corriente directa para encender un LED. El valor de corriente es determinado por el valor de la resistencia en serie con el LED.



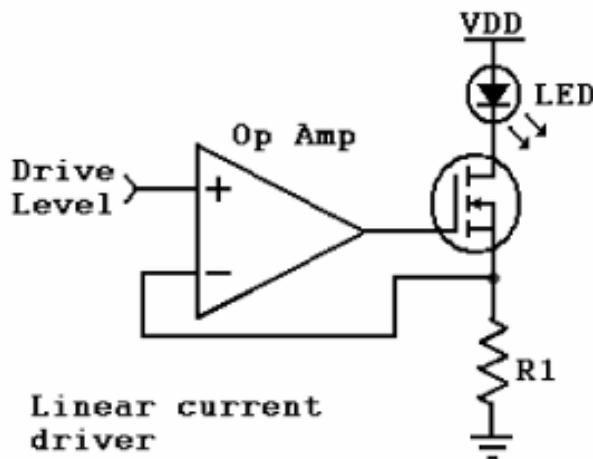
Un LED de alta intensidad requiere una corriente mucho mayor a 350mA, mucho más de lo que un micro controlador puede suministrar en sus pines entrada/salida. Una solución es conectar el LED directamente a la fuente de energía y agregar un MOSFET en serie con el LED, como se muestra en la figura, para controlar el brillo. Sin embargo, este diseño presenta algunos inconvenientes, la resistencia necesita disipar alrededor de 2W de potencia, lo cual es ineficiente y se requerirá para ello de una resistencia de gran tamaño.

Se puede observar que este diseño carece de una regulación de la entrada de tensión, por ejemplo, si es un sistema el cual se alimenta de una batería, cuando esta se descargue, la tensión descenderá, lo que provocará que la corriente directa también disminuya.



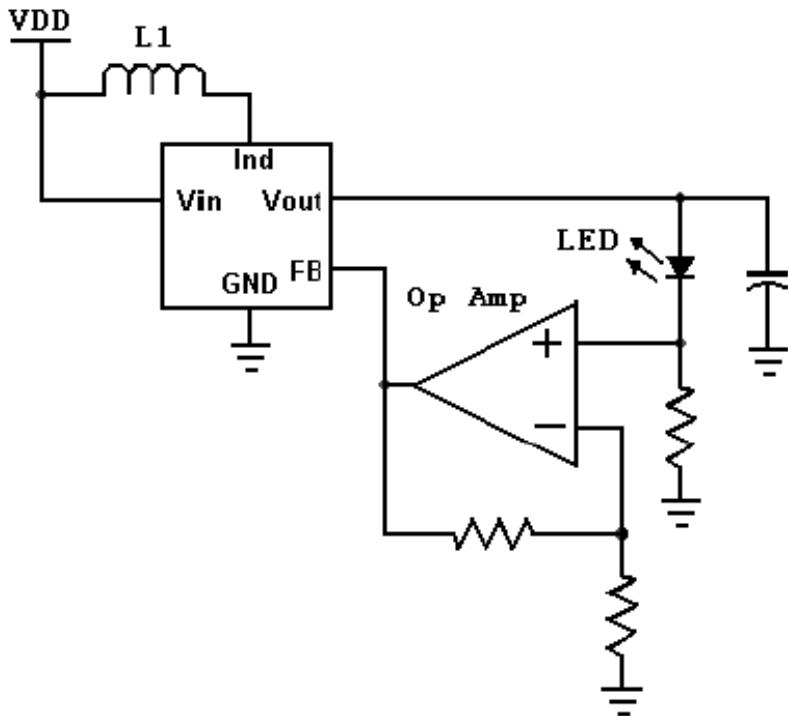
$I_F \text{ LED} \gg I \text{ Pin} \rightarrow \text{MOSFET}$

Otro método de control es colocar al circuito anterior un amplificador operacional, el cual tome la tensión de la resistencia R como referencia para compararla con una tensión inyectada que defina la intensidad del LED. Este circuito permite un control de la corriente continua, pero la transferencia de potencia al LED continua siendo ineficiente. El MOSFET disipa calor la cual es un consumo de energía.



Una solución ideal al problema es utilizar fuentes switching. Debemos tomar una muestra de la corriente que circula por el LED e inyectarla a algún dispositivo, que en función del valor modifique la energía que se le suministra al LED.

En este circuito, el amplificador operacional amplifica la caída de tensión en la resistencia y actúa en el circuito para modificar la tensión que llega al LED.

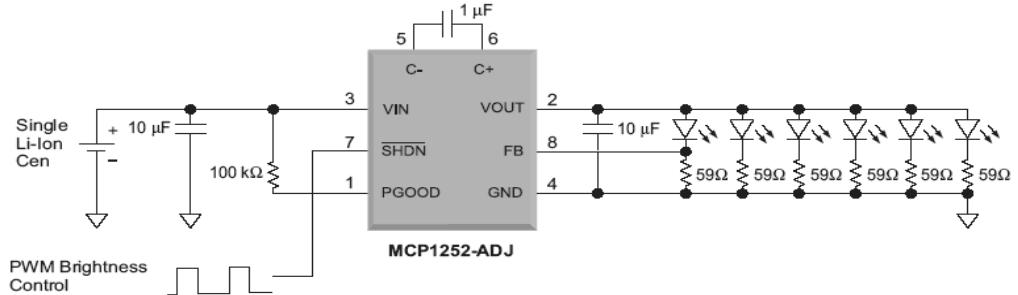


## TIPOS DE CONVERTIDORES

En función de la tensión de la fuente de energía existen distintos modelos de circuitos a desarrollar, los cuales fueron nombrados anteriormente.

### BOMBA DE CARGA

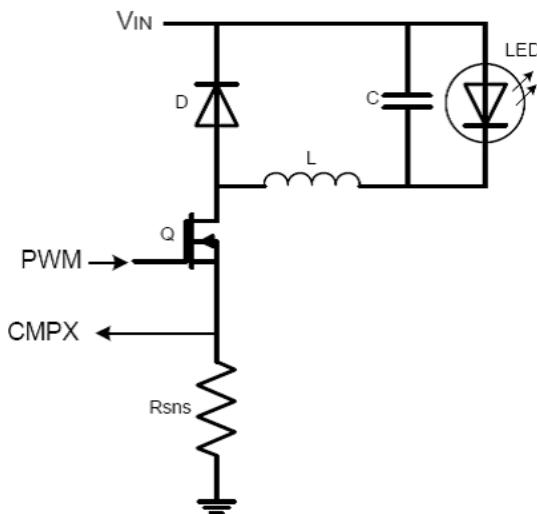
Una fuente de alimentación por “Bomba de Carga” no tiene inductores que se requieren en otras topologías SMPS, lo que la hace muy simple. Esto permite que los circuitos sean más compactos y económicos. El lado “débil” de esta configuración es que las bombas de carga no pueden suministrar grandes cantidades de corriente comparado con otras topologías. Los circuitos con Bombas de Carga se utilizan con mayor frecuencia en iluminación del tipo backlight, por ejemplo, en PC's, Display LCD, e instrumental del automóvil.



## CONVERTIDOR BUCK

Este es un ejemplo de un circuito convertidor BUCK. El convertidor BUCK convierte una tensión de entrada alta a un valor de salida baja. Debido a esto, la tensión de entrada siempre se mantendrá mayor que la tensión directa del LED.

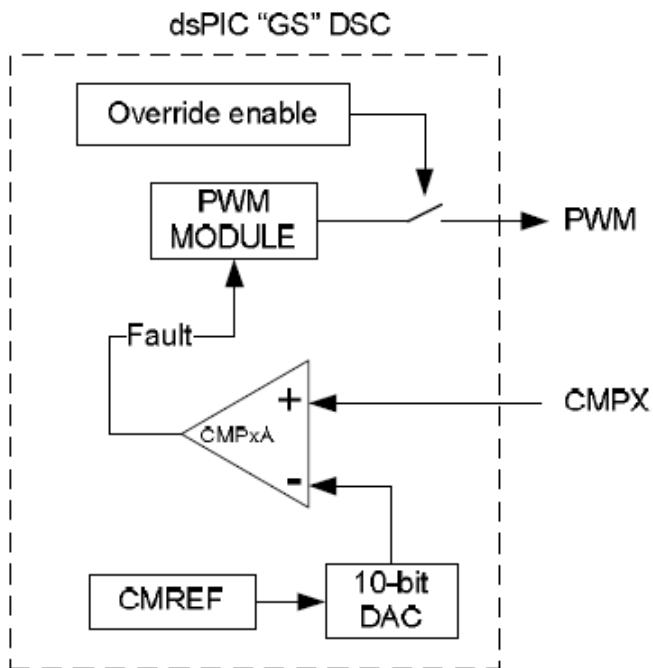
La resistencia sensora  $R_{sns}$  es utilizada para monitorear la corriente directa del LED utilizando el comparador. Veamos ahora cómo la realimentación con la resistencia funciona en el dsPIC.



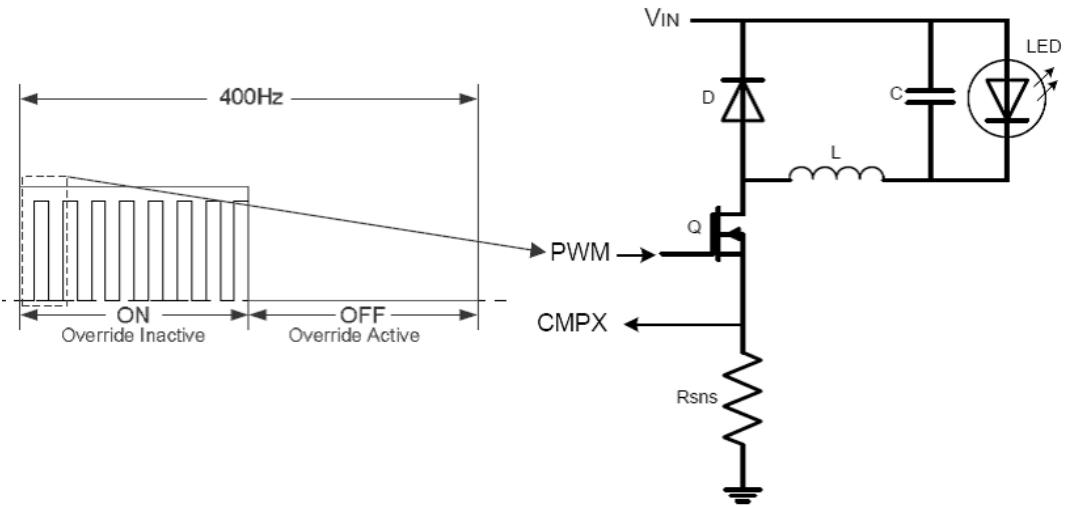
Este diagrama es una representación general de un comparador analógico y cómo interactúa con el módulo PWM en el dsPIC.

La corriente de la resistencia sensora será igual a la corriente directa del led por estar en serie. La tensión producida por dicha corriente es ingresada a la entrada no inversora de un comparador. El registro C<REF es utilizado para configurar una tensión de referencia con un DAC de 10 bits. Si la entrada de la Resistencia sensora es igual o mayor de la de referencia, el comparador generará

un lapso, el cual causará que la salida PWM se coloque en estado bajo por el resto del periodo PWM. Si el lapso no está presente en el inicio del siguiente periodo PWM, el PWM se reinicia. Gracias a su rápido tiempo de respuesta de 20 nano segundos, una ventaja significativa en el uso de comparadores analógicos, no se necesitan ciclos de CPU para regular la corriente directa, ya que la generación del lapso esta directamente conectado al modulo PWM. Esto permite al PCU realizar otras funciones y todavía mantener estable la corriente por el LED.

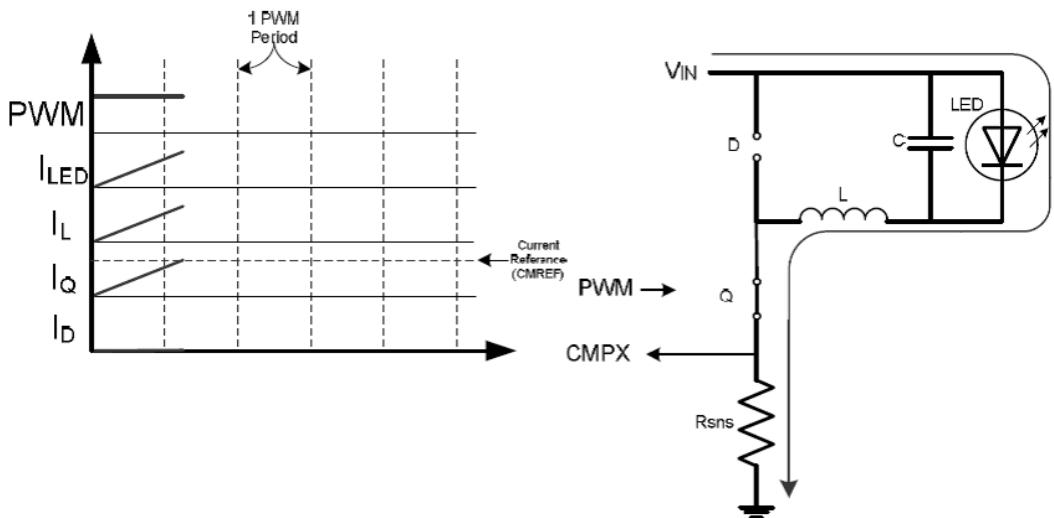


Ahora miremos de cerca el tiempo de encendido del período de atenuación en un convertidor tipo Buck. Veremos cómo el PWM de 500 Khz regula la corriente directa en el LED durante los primeros ciclos.

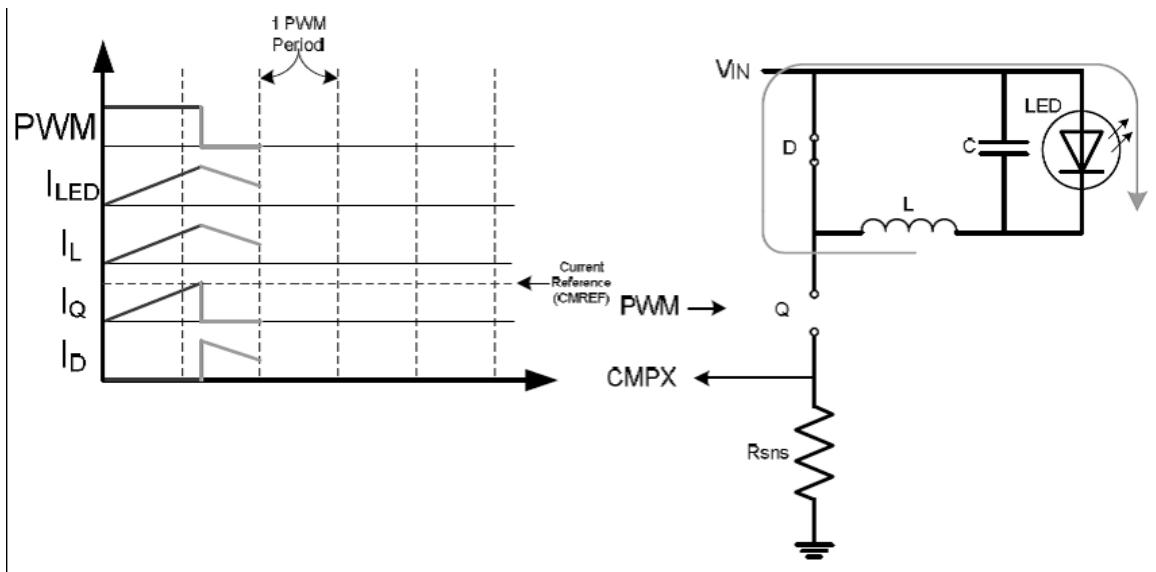


Sigamos el camino con el inicio de un Nuevo período de atenuación donde todas las corrientes comienzan en cero. En el inicio del tiempo de encendido (ON), el PWM es activado y posee un ciclo de actividad del 100%. El diodo D se encuentra en reversa, por lo que no conduce y la corriente toma el camino indicado con línea roja a través del LED, inductor, MOSFET y la resistencia sensora.

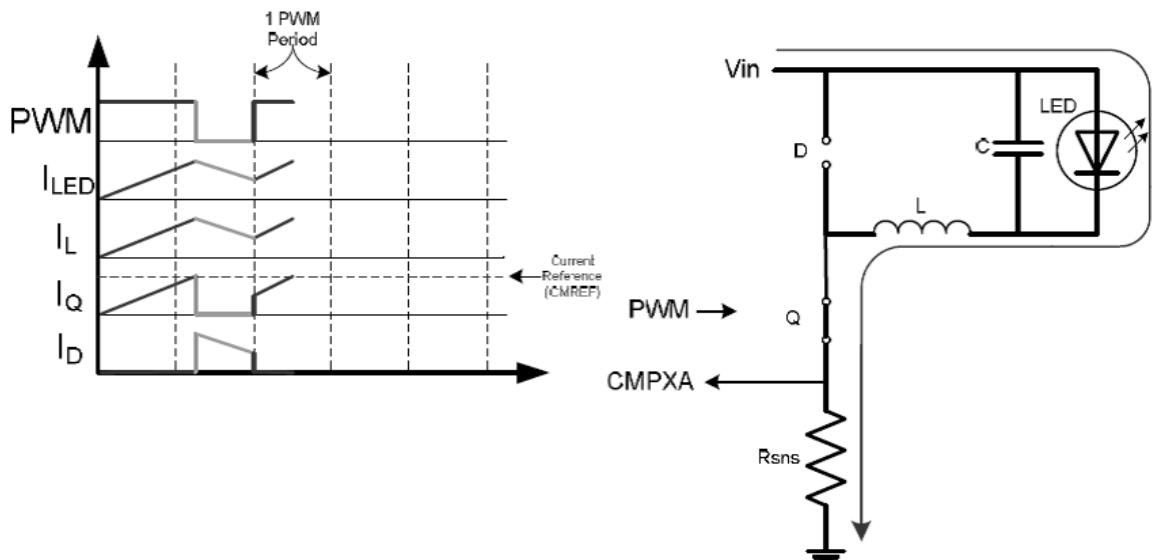
El PWM sigue activo hasta pasado el primer período hasta que la corriente del MOSFET alcanza la corriente de referencia. En este punto, el comparador analógico reconoce que hay una falla y pone el PWM en estado bajo por el resto del período del PWM.



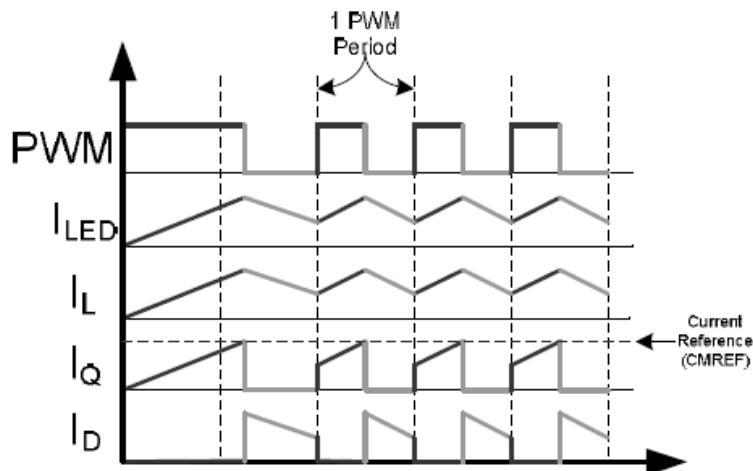
Como el MOSFET Q está ahora abierto, el diodo D se encuentra en directa y el inductor provee ahora la corriente para el LED como se muestra con la línea clara.



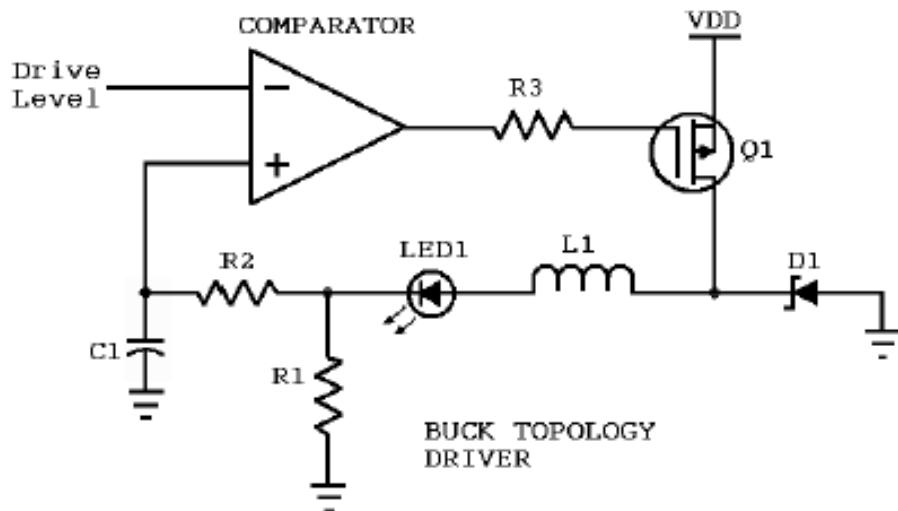
Ahora que el nuevo período PWM ha comenzado, el PWM se reinicia y la corriente del MOSFET vuelve a subir a la de referencia. En este punto se producirá un fallo y el módulo PWM volverá a ser puesto a nivel bajo.

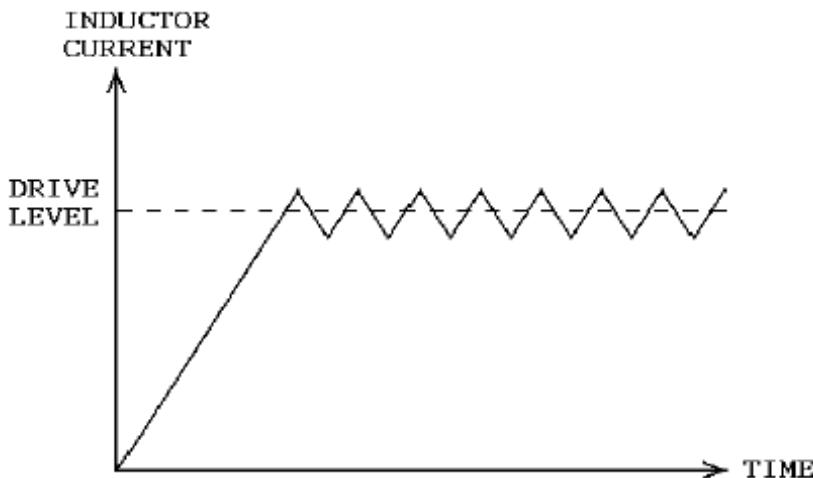


Este proceso es continuo durante el tiempo de encendido (ON) del periodo de la atenuación.



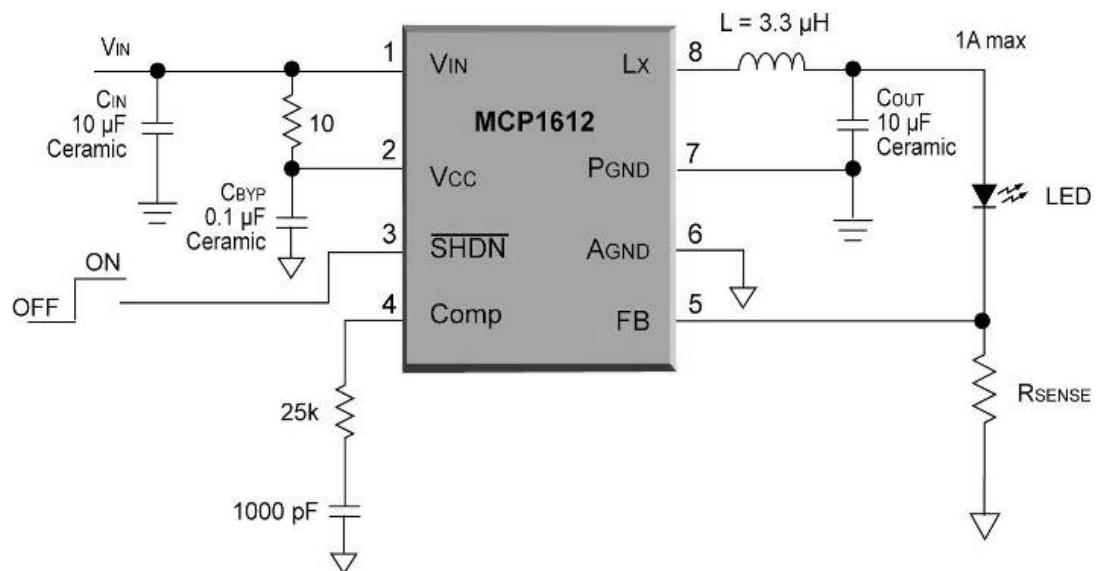
#### ANALICEMOS OTRO CIRCUITO DEL CONVERTIDOR BUCK





El funcionamiento es similar al descrito anteriormente y podemos observar en el grafico cómo el valor de corriente varia alrededor del valor deseado.

Un regulador Switching en la configuración Buck permite una gran eficiencia cuando  $V_{IN} > V_{LED}$  como se observa en la figura siguiente.



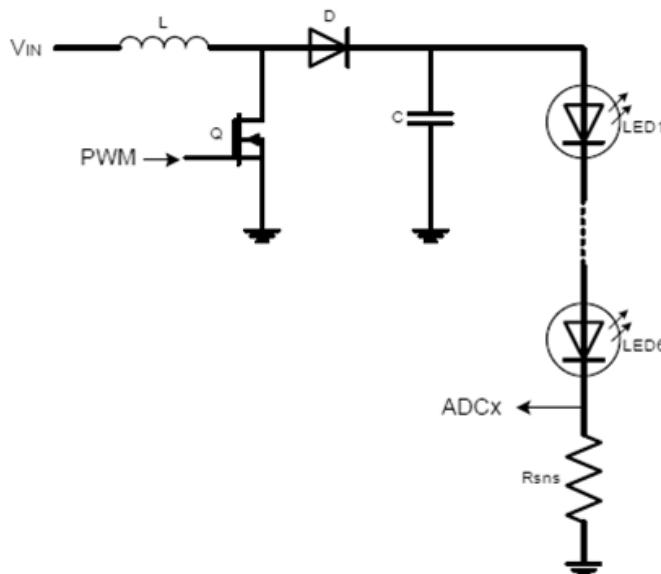
## CONVERTIDOR BOOST

Un convertidor Boost convierte una tensión de entrada baja a un nivel mayor de salida.

Como resultado, la entrada siempre debe ser menor que la tensión directa del LED

El valor promedio de la corriente directa es monitoreado por medio de la resistencia sensora  $R_{sns}$  utilizando un conversor ADC del dsPIC.

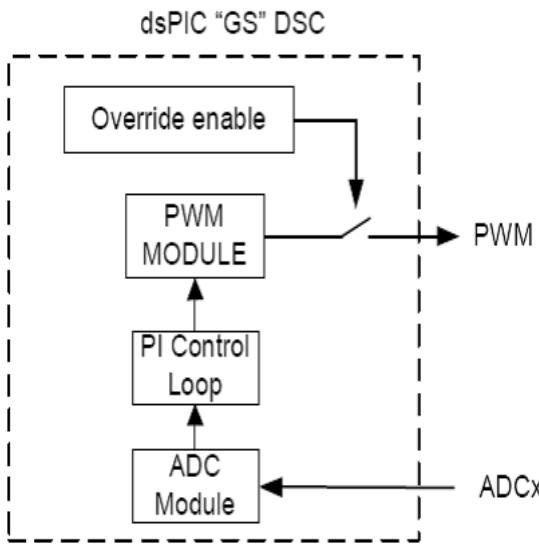
Los reguladores Switching en una configuración Boost permiten controlar una gran cantidad de LEDs en serie partiendo de una pequeña fuente de tensión  $V_{IN} < V_{LED}$



Este diagrama es una representación general de cómo los módulos ADC y PWM interactúan en un DSP (Digital Signal Controller).

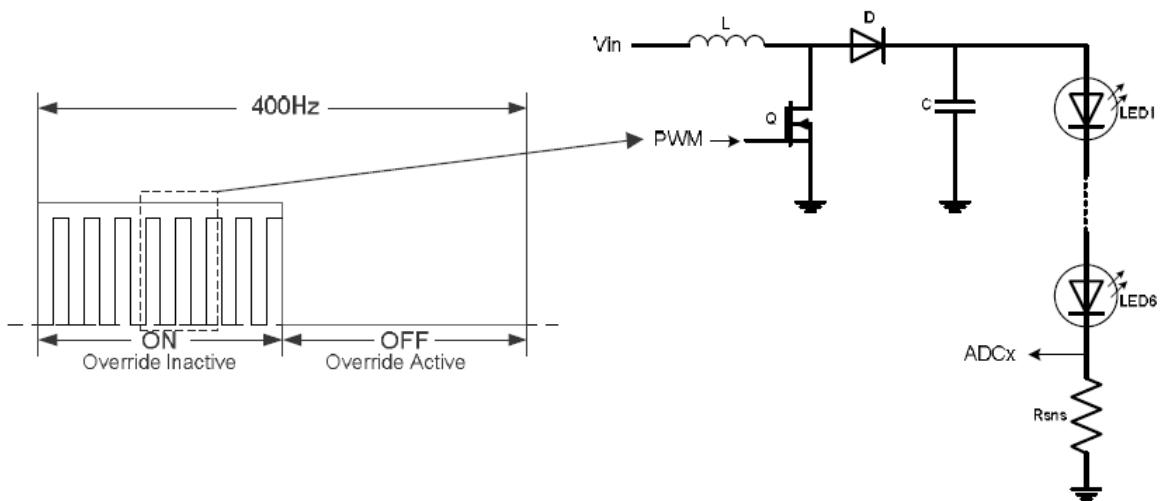
La corriente de la resistencia sensora es introducido en el modulo ADC, un lazo cerrado PI procederá por medio del software a modificar el ciclo de actividad del PWM.

Como un dsPIC serie “GS” posee unos 40 MIPS de velocidad, es posible controlar el brillo de varios LEDs de alta intensidad con el mismo método.



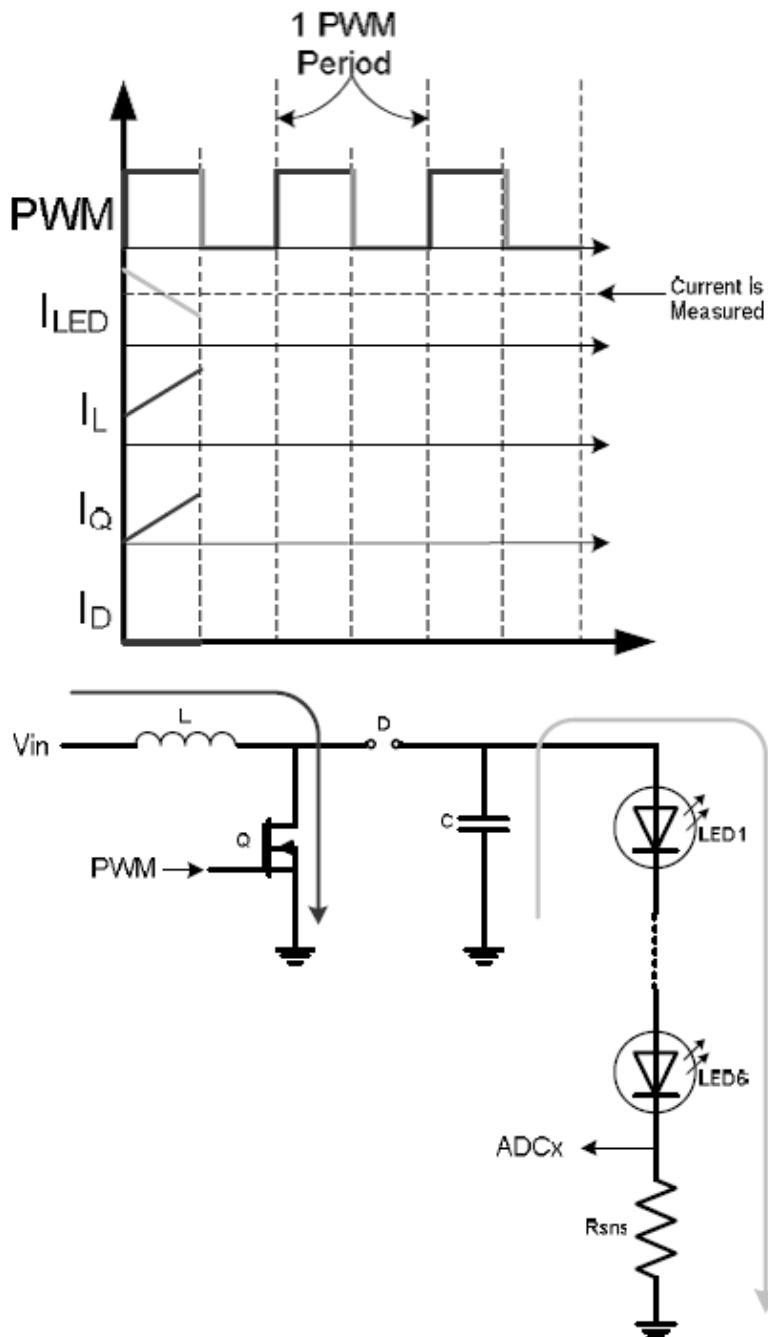
La corriente promedio a través de los LED es monitoreada usando el modulo ADC  
Un lazo de control PI se utiliza para regular la corriente directa (IF) del LED

Ahora veamos el ciclo de ON del convertidor Boost, veremos que la señal de 500Khz regula la corriente del LED:

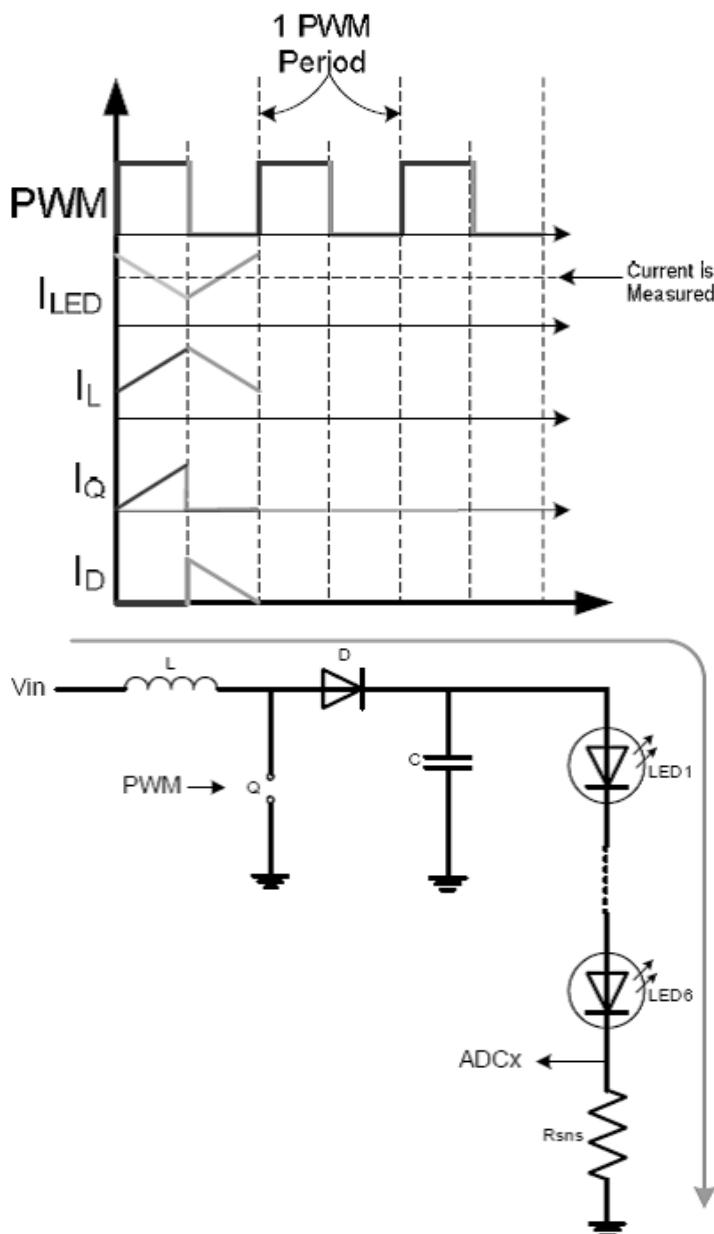


En el inicio del período PWM la corriente de entrada toma la ruta representada por la flecha roja a través del inductor y MOSFET. Durante este tiempo el condensador alimenta la corriente para el led, tal como se muestra en la flecha dorada.

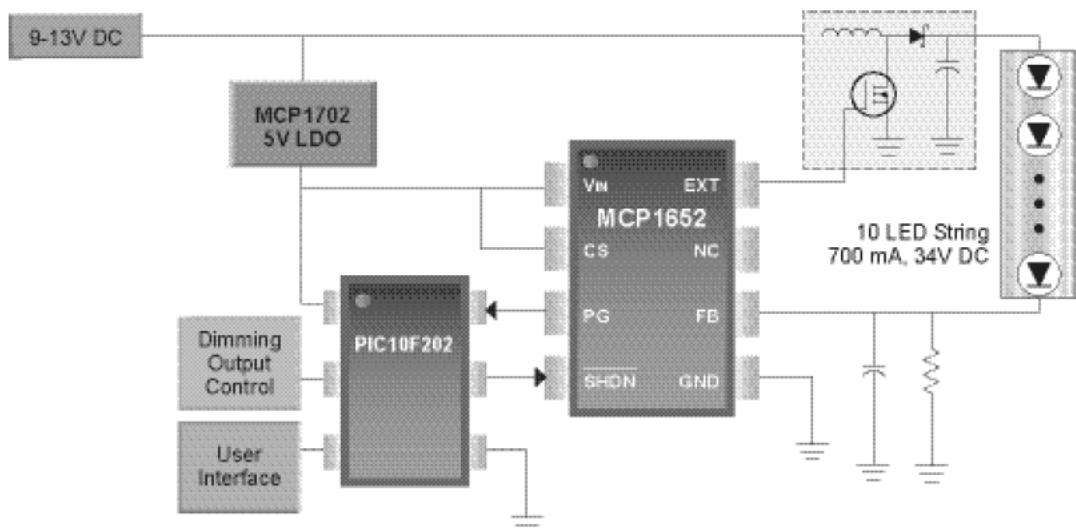
La corriente de avance de los LED se mide en el punto medio del PWM con tiempo suficiente para que el promedio actual de 1 ciclo de la señal PWM se pueda determinar. La medición actual ahora se pueden usar en el bucle de control PI para ajustar el ciclo de trabajo PWM según sea necesario.



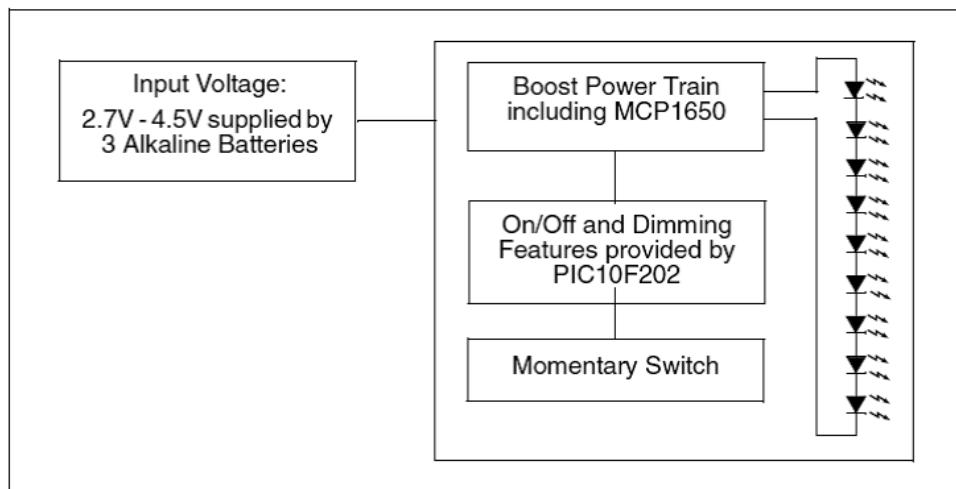
Veamos ahora el punto medio del periodo del PWM, el PWM está en estado bajo, el MOSFET está abierto y la corriente fluye por el camino indicado con la flecha azul a través del inductor, el diodo LED y la resistencia sensora.



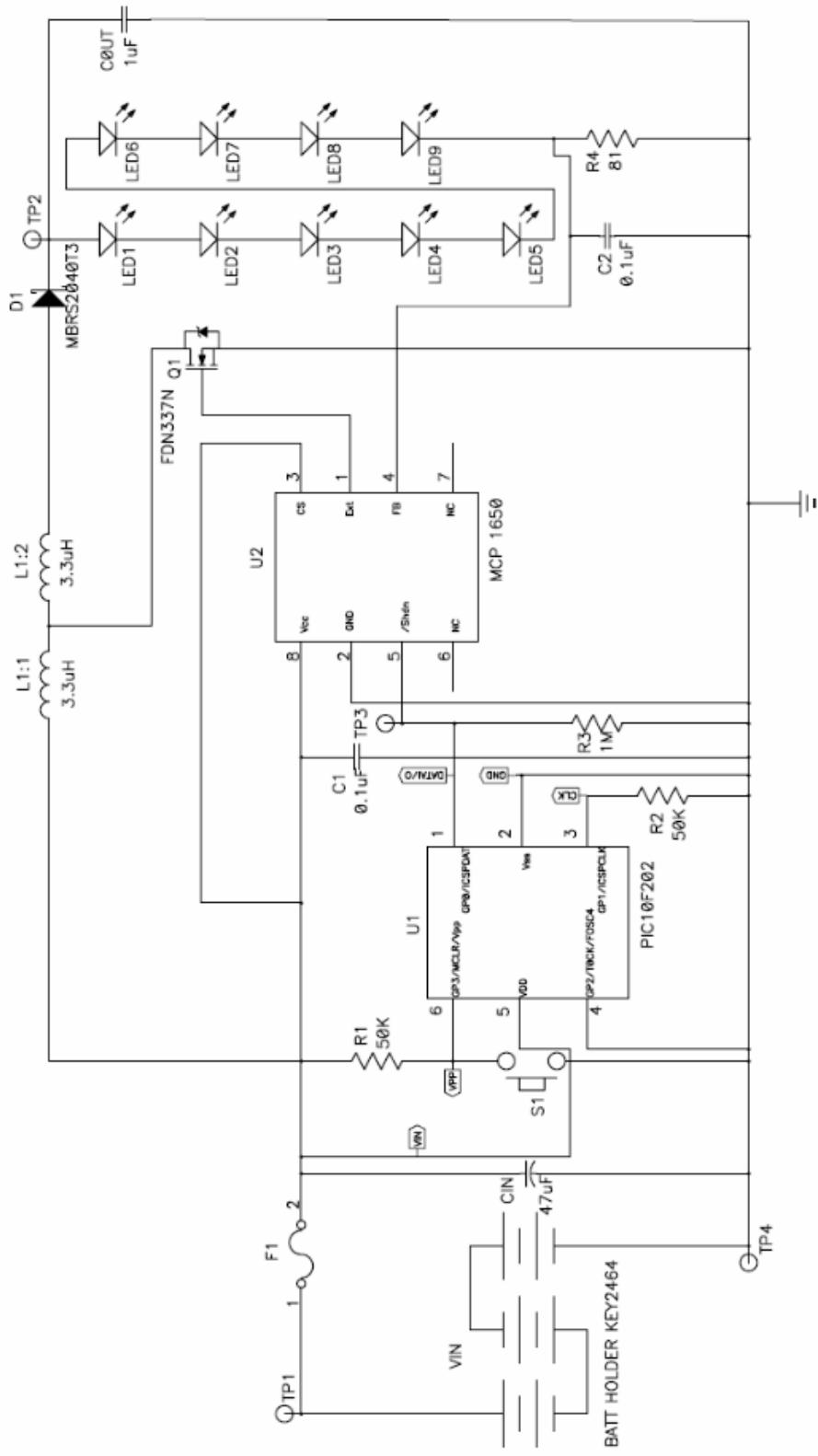
## CONTROLADOR BÁSICO DE ILUMINACIÓN INTELIGENTE

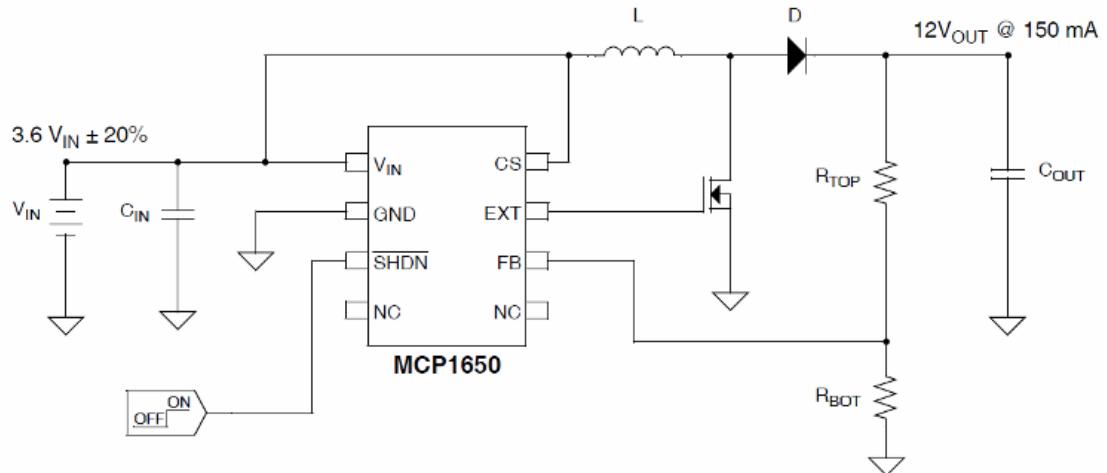


MCP1650/2 funciona a 750Khz



Características especiales del MCP1650/2.

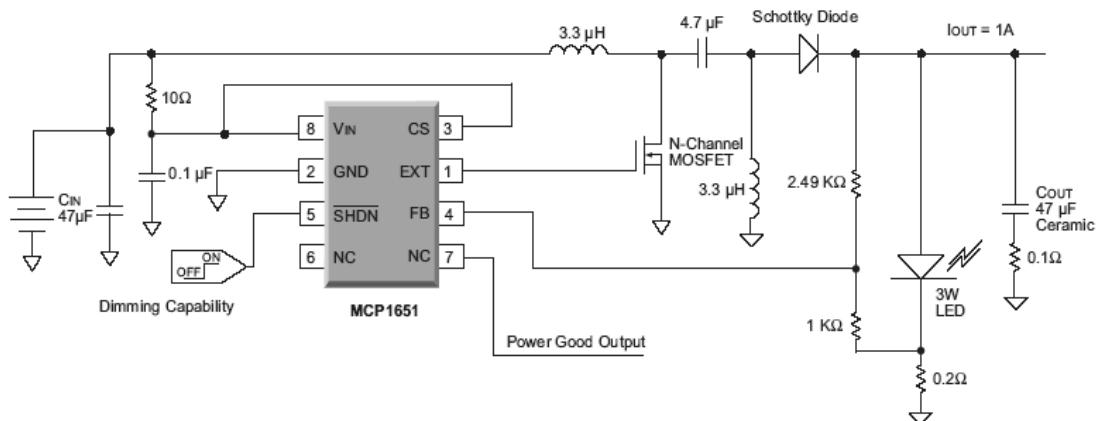




- Salida de potencia con capacidad mayor a 5W
- Capacidad de tensión de salida desde 3,3V a 100V
- Frecuencia de la compuerta del oscilador switching 750 kHz

### SINGLE-ENDED PRIMARY INDUCTIVE CONVERTER (SEPIC)

Las configuraciones SEPIC son útiles cuando existe la necesidad de elevar o disminuir la tensión cuando varia la de la batería



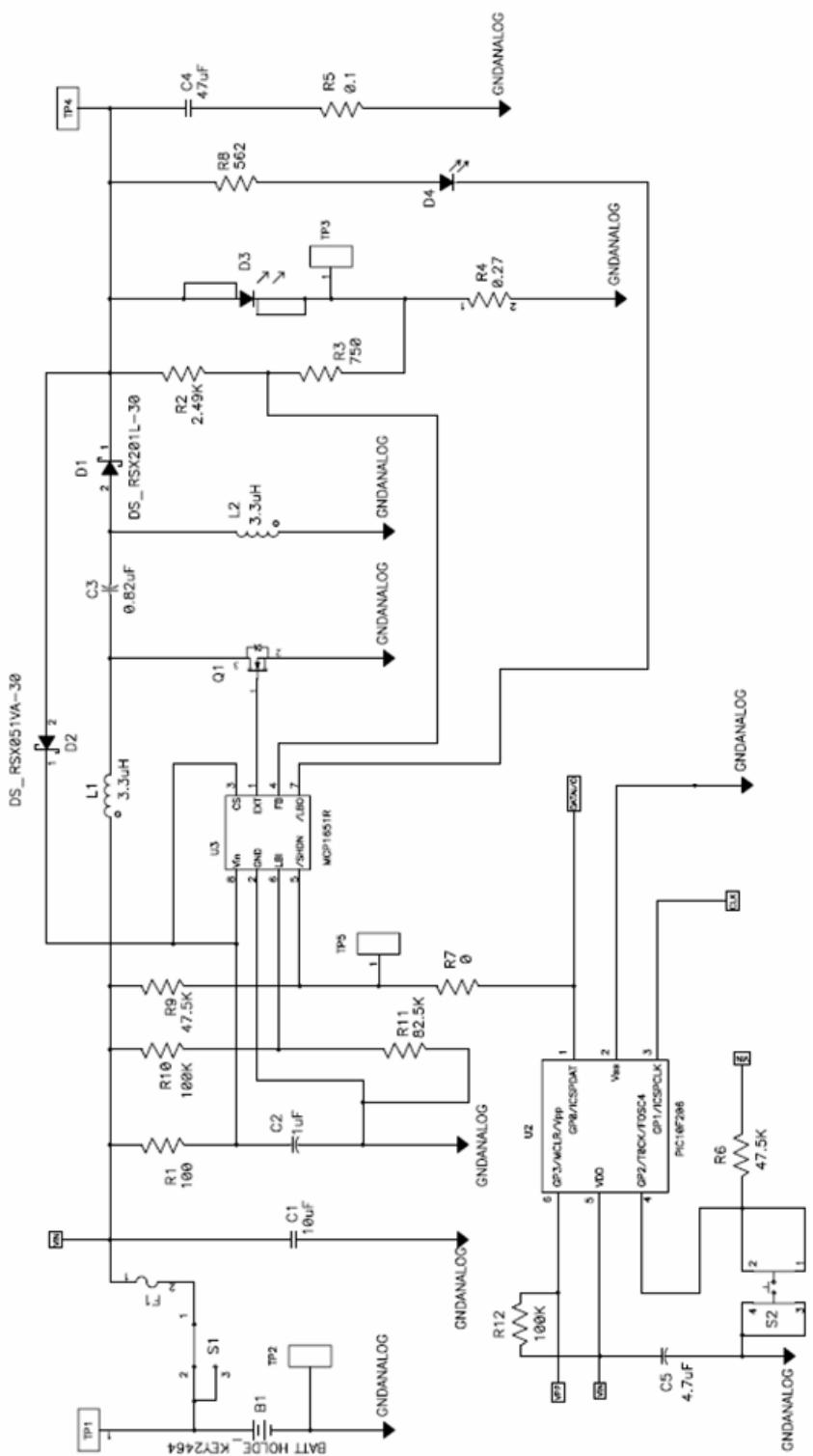
## **MCP165X 3W White LED**

### **Demo Board (Rev. 1) Configuracion SEPIC**

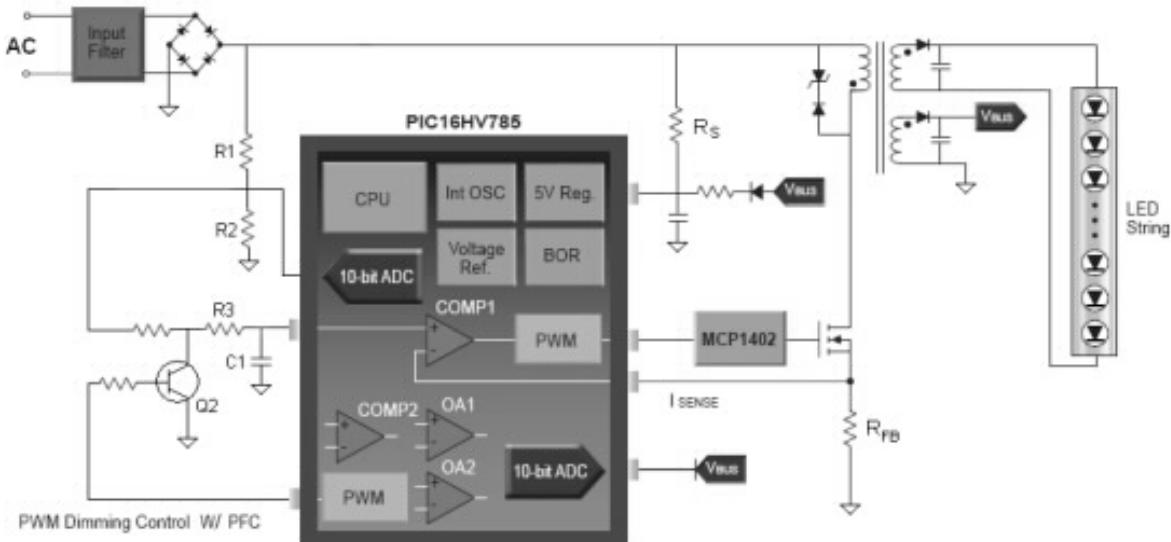
Esta placa nos permite un manejo de LEDs con Regulador SEPIC, esta topología de regulador de convertidor de inductancia primaria simplemente terminada (SEPIC) utiliza un inductor solamente como complemento, posee las siguientes ventajas cuando tenemos baterías como fuente de energía:

- El circuito puede trabajar como “Buck” o “Boost” según cambie la tensión de entrada.
- La topología del circuito provee protección contra corto circuito en forma inherente debido al uso de un capacitor de acoplamiento.

Si en la entrada del MCP1651 pin5 desconectamos el pic y colocamos un interruptor, podremos encender y apagar el led. Este circuito posee un led que nos indicara cuándo la batería cae por debajo de cierto valor para poder proceder a su reemplazo.

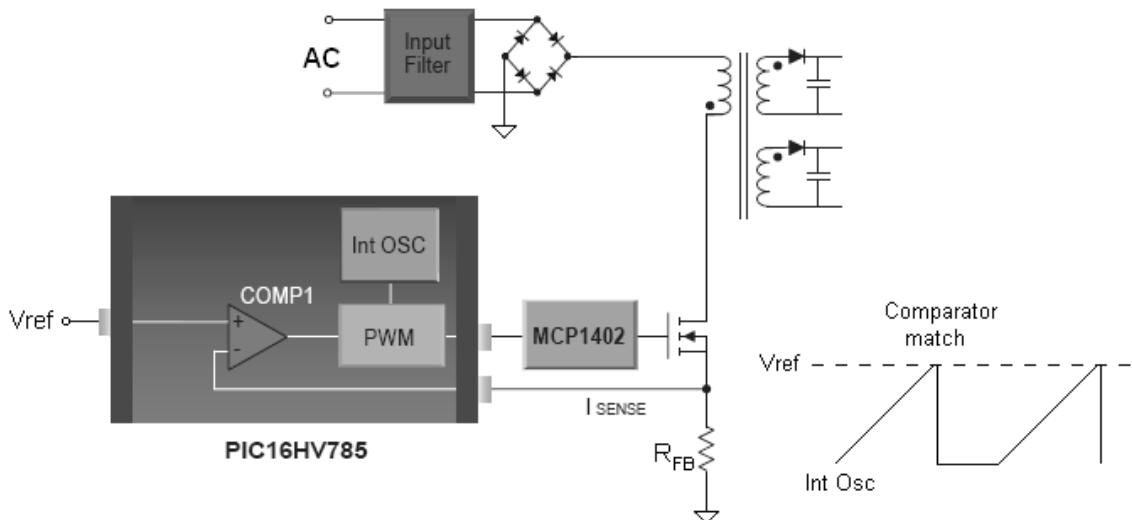


## FLYBACK

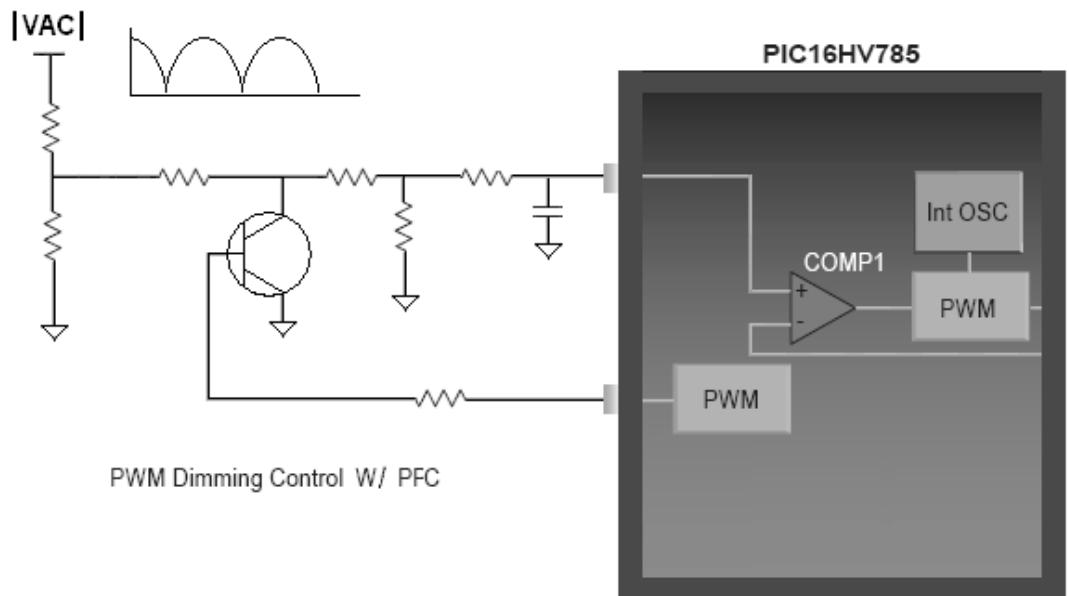


Este circuito es conectado directamente a la línea de tensión (220v) y mediante un transformador podemos obtener las bajas tensiones que necesitamos para nuestro LED. En el secundario, vemos un bobinado el cual se utiliza como realimentación del circuito de control.

### Funcionamiento Básico



## CONTROL DE BRILLO



CCP es usado para controlar la Vref y la potencia de salida

## RESUMEN DE TOPOLOGÍAS

Costo →

	Buck	Boost	Buck-Boost	SEPIC
Rango de Vin	$> V_{OUT}$	$< V_{OUT}$	$< V_{OUT} <$	$< V_{OUT} <$
Eficiencia	Alta	Alta	Med	Med
Aislación	Si	No	Si	Si
EMI	Alta	Bajo	Alta	Bajo
Respuesta Transitoria	Alta	Med	Med	Med

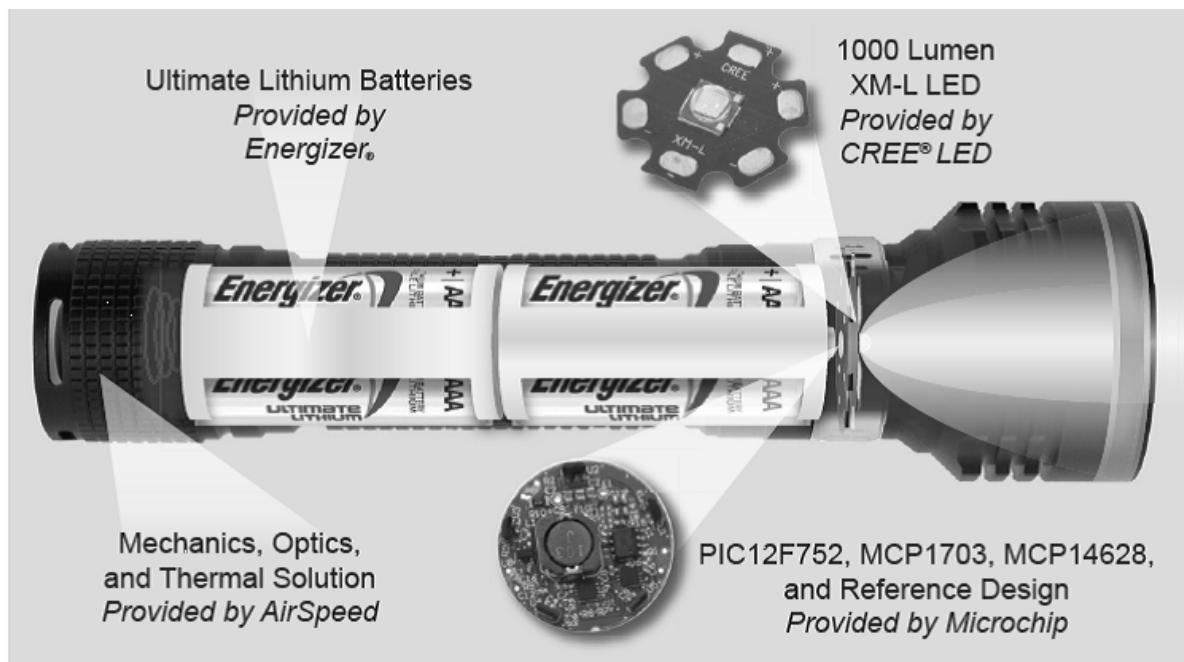
## EJEMPLO DE UNA LINTERNA DE POTENCIA A LED 10W

Muchos de estos diseños utilizan circuitos con resistencias que contribuyen a un mal rendimiento del LED, disminución de funcionalidad, y una menor vida útil.

El LED posee una duración mayor y mejor eficacia que una lámpara incandescente, éste se enciende de forma inmediata, sin necesidad de calentamiento, proporciona colocación precisa de la luz, no produce radiación de calor y pueden funcionar bien en aplicaciones de interior y exterior.

Si al LED le agregamos un MCU que lo alimente y controle, también podremos proporcionar un control de iluminación personalizado, patrones/secuencia, control de la interfaz de usuario y una mayor eficacia ( $lm/W$ ), lo que se traduce en una mayor duración de la batería y en una mayor salida de lúmenes.

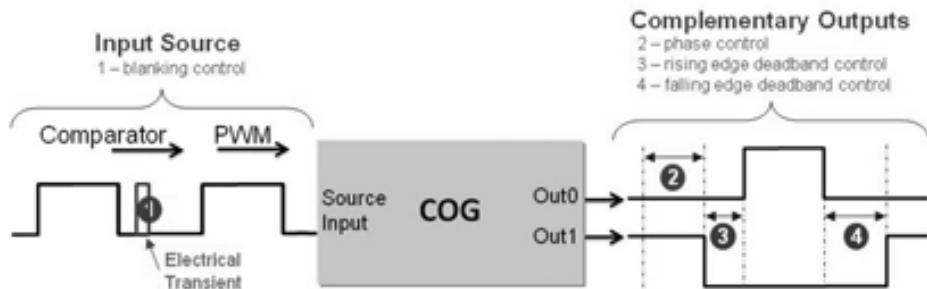
Al utilizar un MCU se puede modificar fácilmente la funcionalidad cambiando el programa.



Utilizaremos en el diseño en PIC12F752, el cual posee un modulo PWM, un comparador que nos permitirá controlar la corriente por el LED.

La salida del microcontrolador, deberá ser conectada a un driver MOSFET, el cual permitirá el control de la corriente por el LED por medio de 2 transistores de potencia en configuración puente T. Este driver, posee una entrada PWM y dos salidas complementarias para los transistores de potencia. Como característica debemos mencionar que entre las dos señales

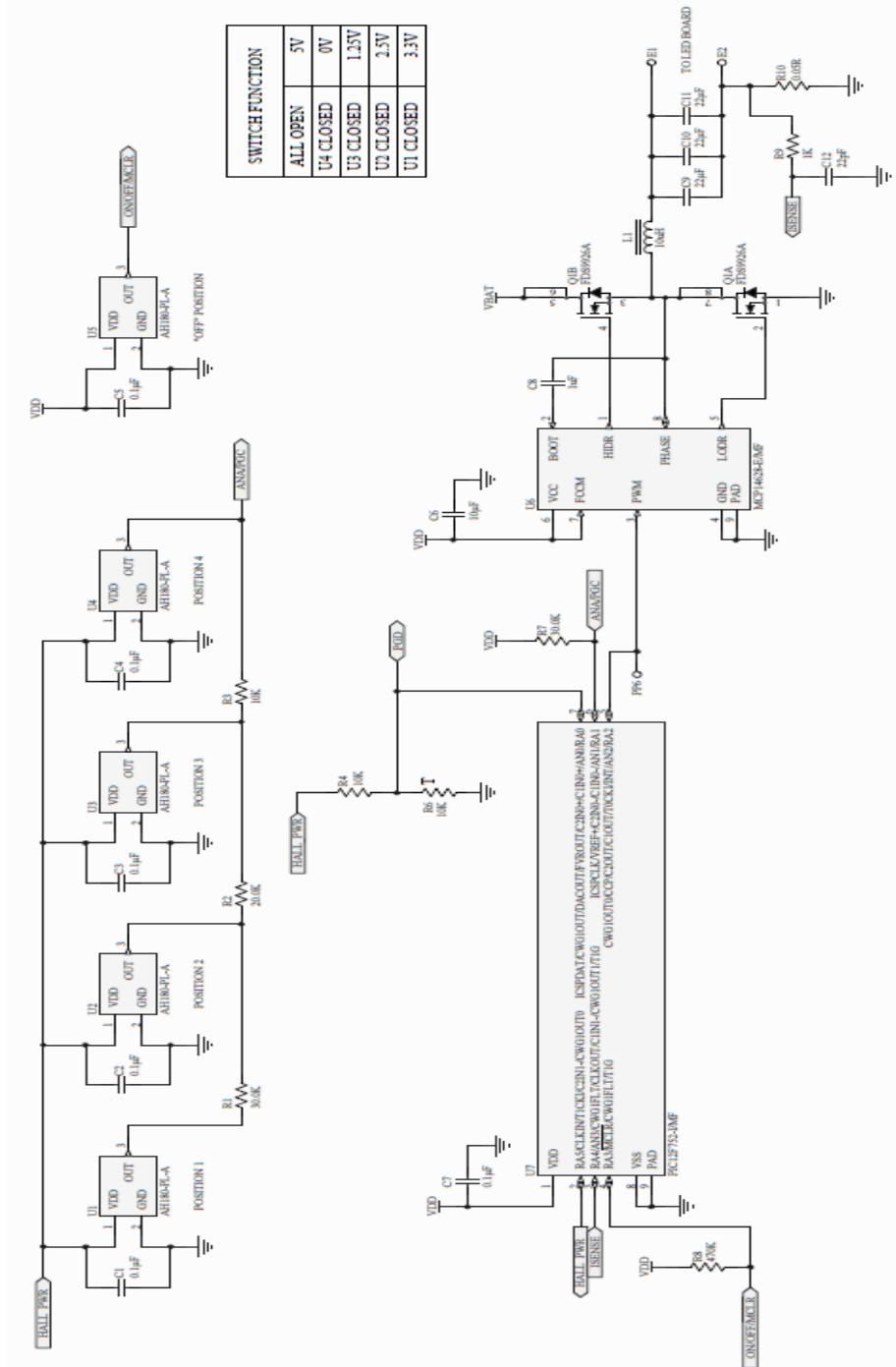
complementarias se generara en forma automática tiempos muertos para evitar inconvenientes en el puente T.



Se utiliza un puente T, en vez de la alternativa con un diodo, para evitar la disipación de potencia en éste y pérdidas de energía, ya que la fuente es una batería y se desea optimizar el tiempo de funcionamiento.

Como vemos en el diagrama en bloques el convertidor utilizado es del tipo Buck CC/CC, logrando una eficiencia del 90%. La eficiencia lumínica es superior a 100 lm/W, y podremos monitorear la temperatura. Al estar implementado el diseño con un microcontrolador, podemos lograr varios modos de operación, de baja potencia (60 lumens @ 0.6W), potencia media (200 lumens @ 2W), alta potencia (1000 lumens @ 10W) y otras funciones que el usuario desee implementar.

La autonomía lograda es de 31 hs @ 0.6W (baja potencia), 8.5 hs @ 2W (potencia media) y 1.5 hs @ 10W (máxima potencia)



## PROGRAMA DEL MICROCONTROLADOR

```
*****
*main.c
*Description: Main program loop
*Hardware: PIC12F752 fixed off time current source implementation
*
* Resources: PICC 9.83 / MPLAB 8.80
*
* CODE OWNERSHIP AND DISCLAIMER OF LIABILITY
*
* Microchip Technology Incorporated ("Microchip") retains all ownership and * intellectual
property rights in the code accompanying this message and in * all derivatives hereto. You may
use this code, and any derivatives created * by any person or entity by or on your behalf,
exclusively with Microchips * proprietary products. Your acceptance and/or use of this code
constitutes * agreement to the terms and conditions of this notice.
*
* CODE ACCOMPANYING THIS MESSAGE IS SUPPLIED BY MICROCHIP "AS IS". NO
* WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT
* LIMITED TO, IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY AND
* FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS CODE, ITS INTERACTION WITH
* MICROCHIPS PRODUCTS, COMBINATION WITH ANY OTHER PRODUCTS, OR USE IN ANY
* APPLICATION.
*
* YOU ACKNOWLEDGE AND AGREE THAT, IN NO EVENT, SHALL MICROCHIP BE LIABLE,
* WHETHER IN CONTRACT, WARRANTY, TORT (INCLUDING NEGLIGENCE OR BREACH OF
* STATUTORY DUTY), STRICT LIABILITY, INDEMNITY, CONTRIBUTION, OR OTHERWISE,
* FOR ANY INDIRECT, SPECIAL, PUNITIVE, EXEMPLARY, INCIDENTAL OR CONSEQUENTIAL
* LOSS, DAMAGE, FOR COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE
* CODE, HOWSOEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE
* POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWABLE
* BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS
* CODE, SHALL NOT EXCEED THE PRICE YOU PAID DIRECTLY TO MICROCHIP SPECIFICALLY
* TO HAVE THIS CODE DEVELOPED.
*
* You agree that you are solely responsible for testing the code and
* determining its suitability. Microchip has no obligation to modify, test,
* certify, or support the code.
*
* Author      Date    Ver     Comment
*~~~~~*/
```

\* Mihnea Apr 2012 1.00 Release  
\* Rosu-Hamzescu

#include "Hardware.h" // disponible en el DVD

```

__CONFIG(FOSCO_INT & WDTE_OFF & PWRTE_ON & MCLRE_OFF & CP_OFF & BOREN_EN &
WRT_OFF & CKLOUTEN_OFF);

void main()
{
    Init_Registers();
    Init_Vars();
    Delay_ms(3000);
    while(1)
    {
        if(TOIF)
        {
            /// Decrement button update counter and light fade counters
            TOIF = 0;
            bcnt--;
            fcnt--;

            /// Read LED current
            AD_SET_CHAN(ISENSE_CHAN);
            _delay(10);
            ad_res = 0;
            for(temp = 0; temp < AD_SAMPLES; temp++)
            {
                AD_CONVERT();
                ad_res += ADRESL;
                ad_res += (ADRESH << 8);
            }

            /// If LED supposed to be on and current below threshold activate missing load protection
            if(!poff && !TRIS_PWM)
            {
                if(ad_res < NOLOAD_PROT) no_load--; else
                    no_load = PROT_COUNT;
                if(!no_load)
                {
                    no_load = PROT_COUNT;
                    cprot = 1;
                    PWM_OFF();
                }
            }
        }

        /// If missing load protection activated go to sleep to avoid damaging circuit.
        if(cprot)
        {
            Shutdown_Peripherals();
        }
    }
}

```

```

    Sleepy_Micro();
}

/// LED dimming using PFM
pfm += pfm_inc;

if(pfm >= PFM_MAX)
{
    pfm = PFM_MAX;
    if(!poff) PWM_ON() else PWM_OFF();
} else PWM_OFF();
}

/// LED fade and brighten can be configured on a separate counter
if(!fcnt)
{
    fcnt = FADE_TIME;
    fade_brighten();
    DACCON1 = dac_set;
}

/// Button reading routines and temperature protection
if(!bcnt)
{
    bcnt = BUTTON_TIME;

    mode = button_scheme();

    /// Read thermistor divider voltage
    AD_SET_CHAN(TEMP_CHAN);
    _delay(10);
    ad_res = 0;
    for(temp = 0; temp < AD_SAMPLES; temp++)
    {
        AD_CONVERT();
        ad_res += ADRESL;
        ad_res += (ADRESH << 8);
    }
}

/// If board temperature over some threshold block turbo mode for 2 minutes to allow
cooling
if(tcool)
{
    tcool--;
    if(mode == TURBO_10W) mode = DIM_2W;
} else

```

```

if(ad_res < TEMP_PROT) tcool = TEMP_COOLDOWN;

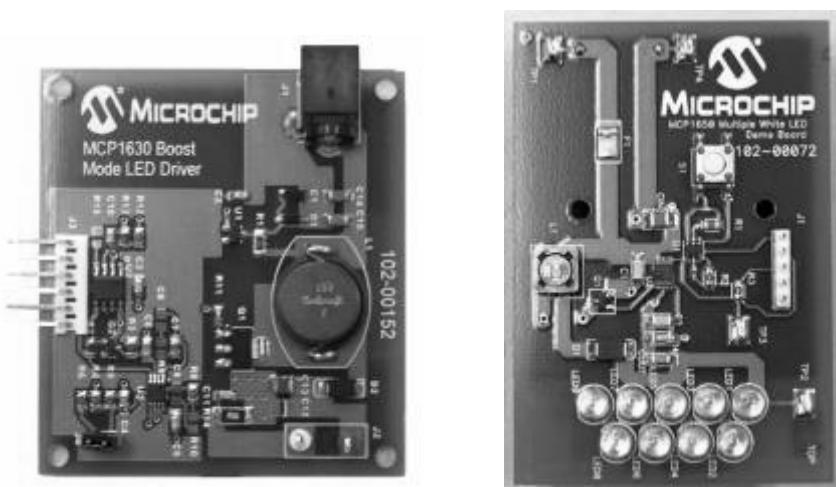
/// Customizable blinking sequence (in this case SOS)
if(mode == SOS_2W)
{
    if(blink_state == SOS_BLINKS) blink_state = 0;
    if(!blink_counter)
    {
        blink_counter = blink[blink_state];
        if(blink_state & 0x01) poff = 1; else poff = 0;
        blink_state++;
    } else
        blink_counter--;
} else
{
    poff = 0;
    blink_state = 0;
    blink_counter = 0;
}

/// Sleep mode will shut down all peripherals to reduce current consumption
/// Wake-up configured on IOC positive edge
if(mode == SLEEP)
{
    Shutdown_Peripherals();
    Sleepy_Micro();
    no_load = PROT_COUNT;
    bcnt = 255;
}
}
}
}

```

Con este programa podemos controlar la linterna con todas las prestaciones. El resto de las librerías se encuentran en el DVD para poder compilar el ejemplo.

## HERRAMIENTAS DE DESARROLLO



El manejo inteligente de LEDs de alta eficiencia, con sistemas de 9 a 13 Volts son fácilmente adaptables a un circuito de manejo inteligente de LEDs de alta potencia utilizando el MCP1702, MCP1652 y un PIC10F202. El MCP1702 conectado directamente a la fuente de 12 Volts crea una fuente de polarización de 5 Volts capaz de entregar una corriente de 250 mA al circuito de control Boost inteligente.

Los LEDs son alimentados por la fuente de tensión potenciada por el MCP1652, minimizando los requerimientos de corriente del sistema de 5V. Un PIC10F202 suma inteligencia al circuito proveyendo una forma de protección térmica, protección por desconexión y corto en la carga, y también la capacidad del usuario de controlar la intensidad lumínica (dimming) entre otras características.

La detección por Batería Baja permite al diseñador determinar el punto de disparo de una condición de batería baja para lograr ajustes “inteligentes” en las funciones con el PIC10F202. La indicación de “Buena” alimentación (Power Good) permite que el diseñador determine cuándo las condiciones de tensión son las correctas. Con estos circuitos el usuario puede realizar experimentos acerca de las distintas configuraciones de las fuentes para LEDs.



# **APLICACIONES GSM Y GPS**



## INTRODUCCIÓN

En este capítulo el objetivo es desarrollar un sistema de rastreo, telemetría y control remoto, que permita reportar los datos a través de la red celular en tiempo real.

Esta idea surgió a partir del crecimiento exponencial que tuvo la red celular, como es sabido esto nos permite estar comunicados prácticamente desde cualquier lugar. Entonces, ¿ porqué no utilizar la red celular para transmitir datos ?. Datos que en este caso son de geolocalización para determinar la ubicación de un objeto a distancia.

Realizar este tipo de telemetría utilizando la red celular, como red de transporte, es mucho más económico que utilizar, por ejemplo, un enlace satelital o una conexión dedicada donde habría que montar antenas propias.

La idea principal es que el cliente pueda rastrear lo que quiera, ya sea una flota de camiones o algo tan pequeño como una mascota, es por eso que el nombre comercial del dispositivo es TrackMe (GPS) y su slogan: rastreá lo que quieras en tiempo real.

Es posible conocer la posición y diversos datos como velocidad de movimiento, aceleración y temperatura utilizando GoogleMaps. Además, el sistema de comunicaciones es bidireccional con lo que es posible enviar comandos al dispositivo. La red ideal para este objetivo es la red celular de telefonía móvil combinada con internet, ya que es la red más ampliamente difundida a nivel mundial.

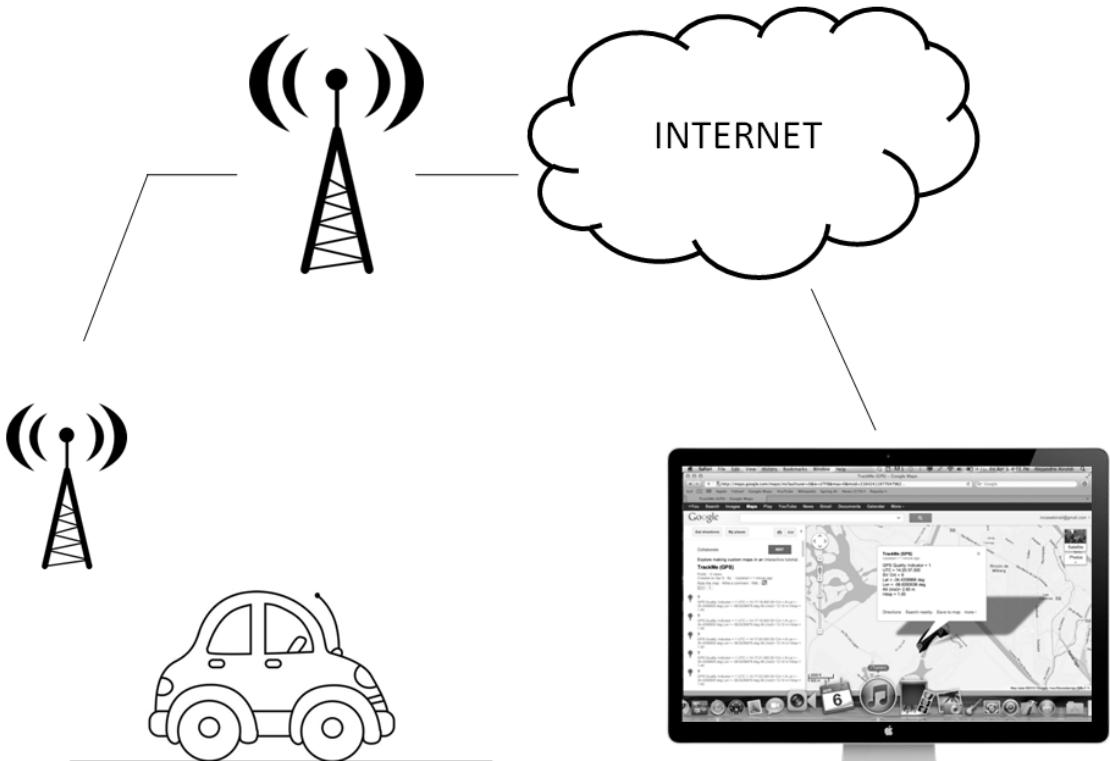
En este capítulo veremos cómo realizar un sistema de telemetría y control remoto a través de la red celular. Para ello aprenderemos a establecer la comunicación entre un microcontrolador PIC y módulos GSM y GPS. Además se explicarán los comandos básicos de operación de un módulo GSM y la trama de un módulo GPS.

También veremos que un producto exitoso debe contemplar el diseño del packaging, el soporte al cliente, regulaciones internacionales y buenas prácticas.

## ESQUEMA GENERAL

En este esquema se intenta mostrar los componentes principales del sistema. Por un lado tenemos el ente a monitorear, ya sea una persona, una mascota, un automóvil o cualquier otro objeto. Por otro lado necesitamos un terminal donde presentar la información, en este caso utilizamos los mapas de GoogleMaps, con lo que se puede acceder desde cualquier computadora conectada a Internet o bien desde un teléfono celular.

También se puede ver en el diagrama la red de transporte, que es una combinación de la red celular (GPRS) e internet.

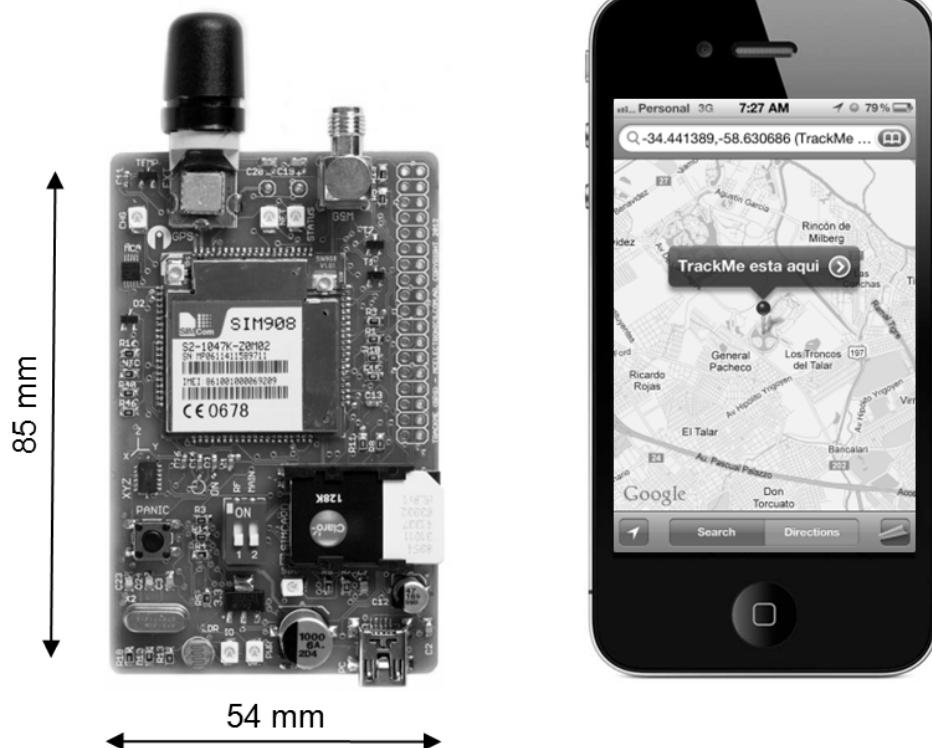


Objeto que se quiere rastrear o controlar a distancia a través del terminal.

El terminal puede ser un teléfono celular con acceso a internet o bien una computadora.

## PLACA DE DESARROLLO

El nombre comercial de la placa es TrackMe (GPS), la misma incluye módulos GSM y GPS, sensores de aceleración, temperatura y luminosidad, cargador de baterías y conexión USB. La idea de este capítulo es que el lector adquiera las herramientas para construir su propia placa tomando como modelo este desarrollo. Dada la complejidad del circuito tuvimos que fabricar el PCB con un tamaño de pista de 4 mils (0.1016 mm). En esta figura es posible comparar el tamaño de la placa TrackMe (GPS) versión 2013 con un teléfono celular.

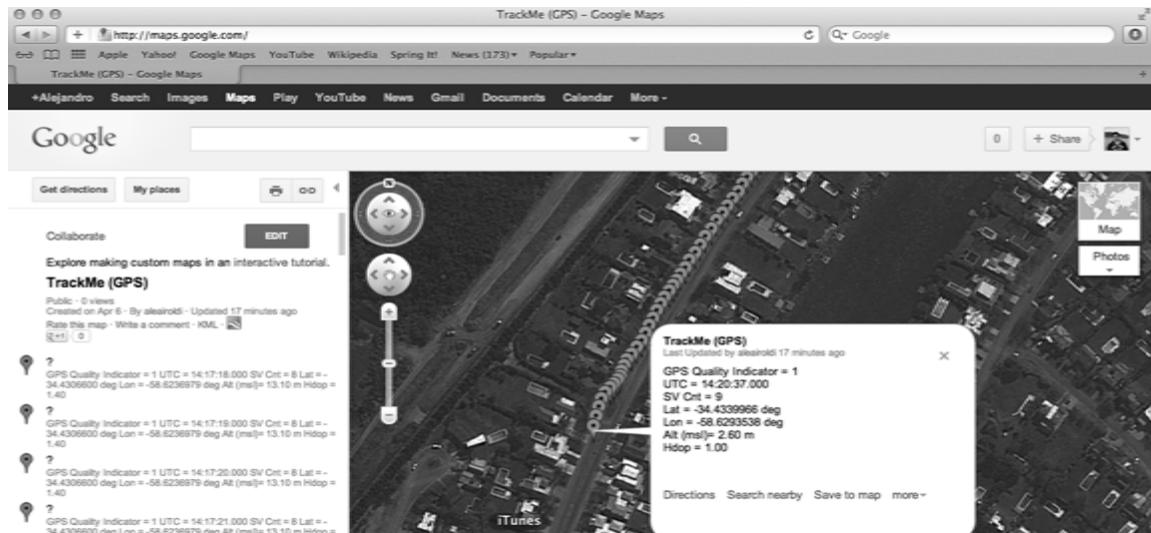


**Comparativa de tamaño con un teléfono celular. Placa TrackMe (GPS) v2013 fabricada por mcelectronics.**

## APLICACIONES

Si bien existe una infinidad de aplicaciones para este tipo de dispositivos, a continuación se mencionan algunas implementaciones que realizaron nuestros clientes:

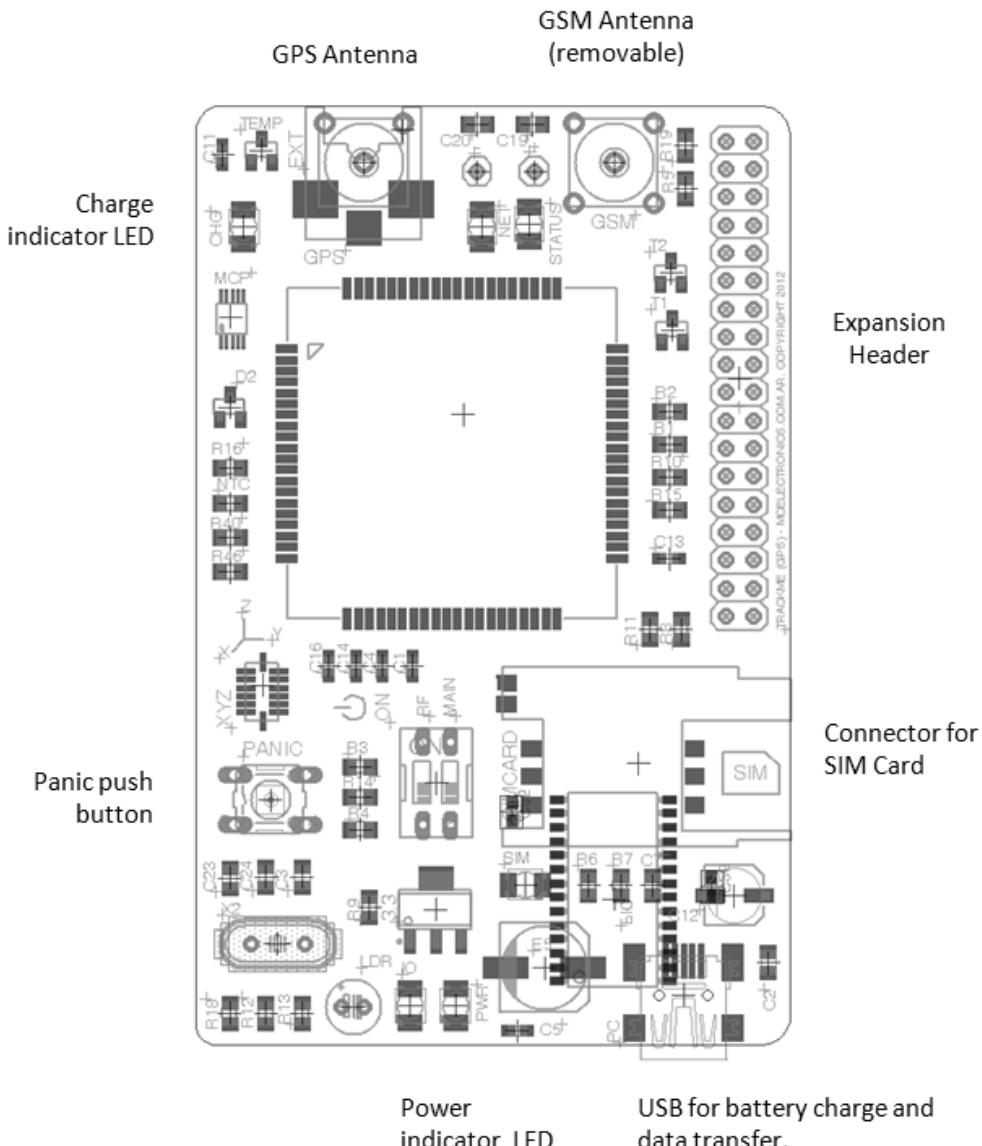
- Collar inteligente para mascotas.
- Alerta temprana de niebla.
- Monitoreo de envíos.
- Rastreo y monitoreo de flota.
- Indicadores en el transporte público de pasajeros.
- Ayuda a personas con dificultades de orientación.
- Sistema de alarma comunitaria.
- Aplicaciones en domótica.



**Placa TrackMe (GPS) reportando las coordenadas en tiempo real.**

## LAYOUT DE COMPONENTES

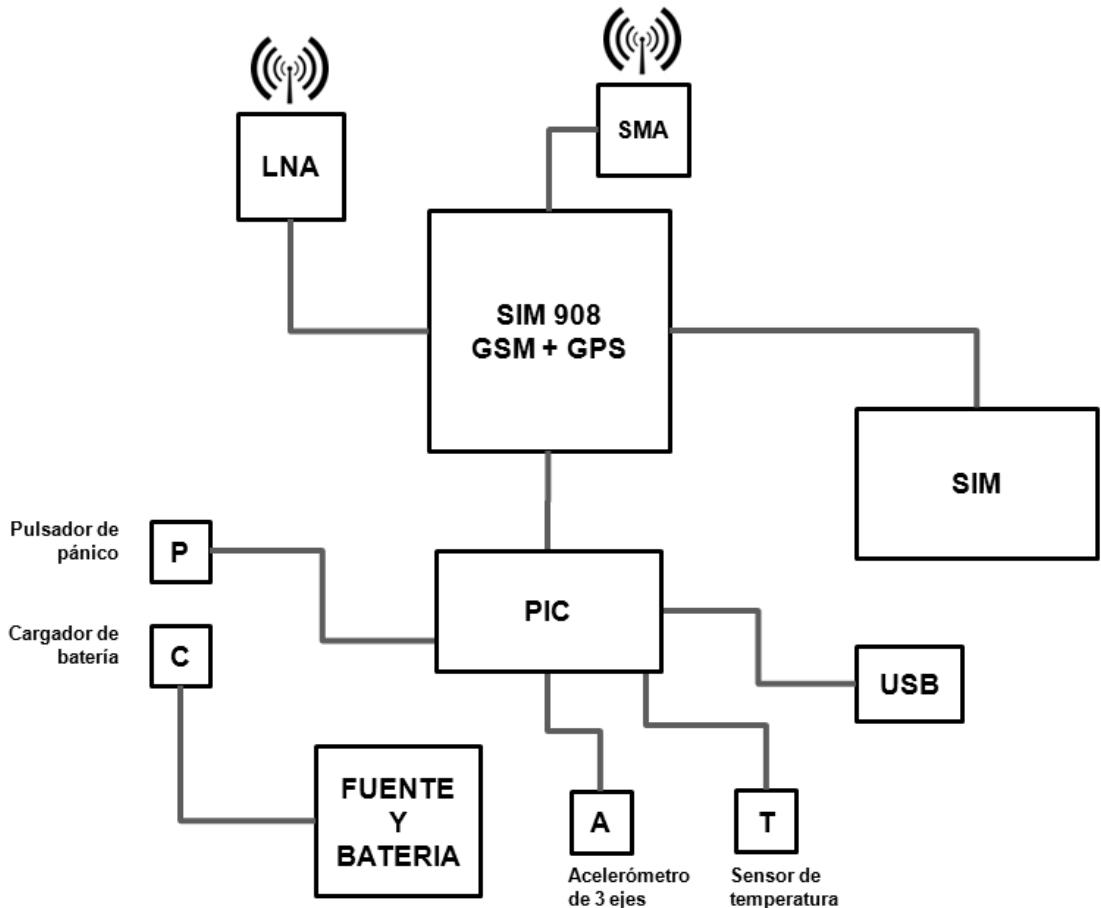
El esquemático completo puede encontrarse al final del capítulo. A continuación se describen los componentes principales. Como veremos la placa tiene una batería, similar a la que se encuentra habitualmente en los teléfonos celulares y que se puede conseguir en el mercado local. Esta batería tiene una autonomía de 72 hs en stand-by y se encuentra del lado de abajo de la placa es por eso que no aparece en este diagrama.



## ESQUEMA DE LA PLACA

Básicamente la placa está compuesta por un módulo GPS (para obtener la posición) y un módulo GSM (para transmitir la información). Cada uno de estos módulos requiere su correspondiente antena, en el caso particular del GPS, por ser una señal muy débil, debemos colocar lo más próximo posible a la antena un LNA (Low Noise Amplifier).

Todo el sistema es controlado por un microcontrolador PIC® de 8 bits. El programa de este micro está íntegramente programado en C y es el que se encarga de tomar los datos del GPS, procesarlos y transmitirlos al módulo GSM.

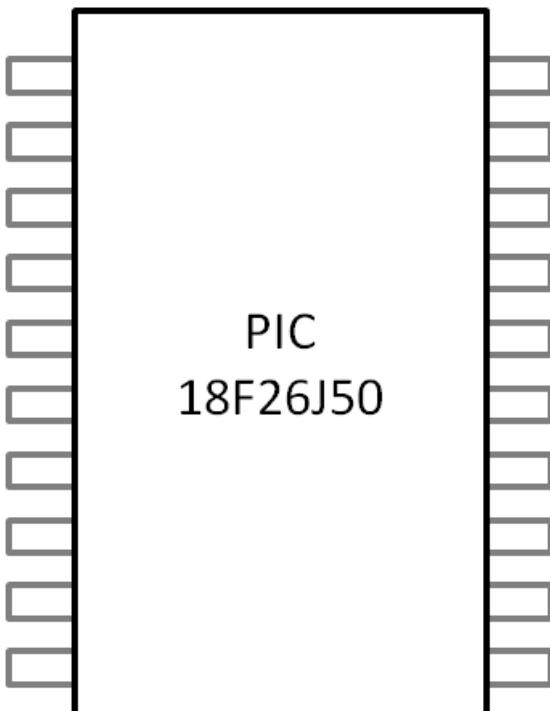


## EL MICROCONTROLADOR

Utilizamos el PIC18F26J50. Este es un microcontrolador de los más modernos de Microchip. Fundamentalmente es XLP, es decir Extreme Low Power, lo que significa que consume muy poca energía comparado con las familias anteriores, esto es crítico en este tipo de sistemas alimentados a batería.

Además posee 2 USART (\*) y 1 puerto USB para conectarlo directamente a la computadora. Por una cuestión de espacio se optó por la versión SMD, pero las características no varían según el tipo de encapsulado.

Este microcontrolador puede ser programado en lenguaje C y con cualquier compilador compatible, lo que le da una gran flexibilidad y la posibilidad de adaptarlo a diversas aplicaciones.

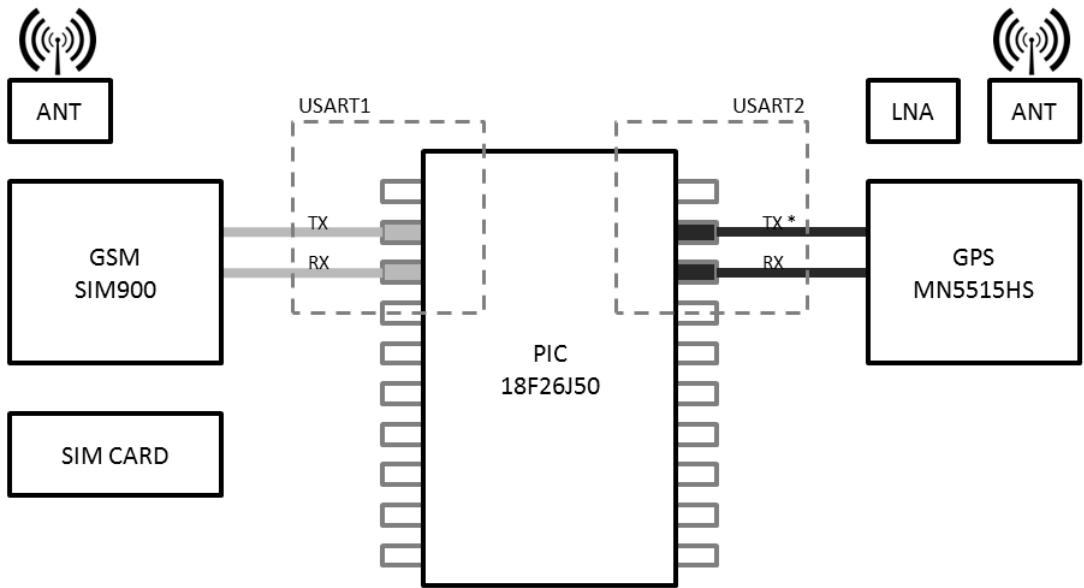


*Características destacables del PIC18F26J50: posee pines de comunicaciones remapeables, velocidad de operación: 12 MIPS, 64 KB de memoria Flash y 3.8 KB de RAM*

*(\*) Las 2 USART están implementadas por hardware.*

## CONEXIONES DEL MICROCONTROLADOR

Veamos en este esquema como está conectado el microcontrolador a los módulos GSM y GPS. Ambos módulos se conectan a través de la interface serie USART.



(\*) TX del PIC en el módulo GPS se utiliza para solicitar otras tramas o variar el intervalo de recepción.

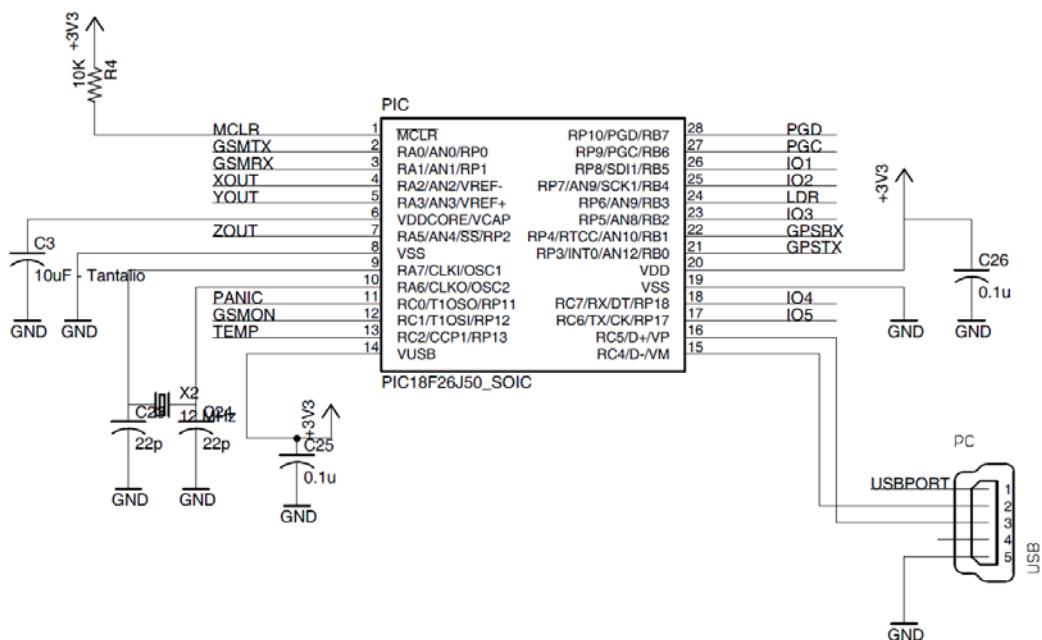
## CONEXIONES DEL MICROCONTROLADOR

Se muestra a continuación el esquema eléctrico del PIC, aquí se ven todas las conexiones a los distintos dispositivos de la placa. El PIC se alimenta con 3.3 v, es conveniente colocar un capacitor de .1 uF en paralelo lo más próximo posible al pin de alimentación.

Por otro lado están las conexiones SIMTX y SIMRX al módulo GSM y los pines GPSTX y GPSRX que proveen la comunicación serie al módulo GPS.

Los pines XOUT, YOUT y ZOUT son entradas analógicas que reciben la señal de los 3 canales del acelerómetro.

Finalmente están los pines digitales, estos son PANIC (entrada digital para el pulsador de pánico) y RELAY (salida digital para activar un relay). El PIC se conecta directamente a USB por medio de los pines D+ y D- como está indicado en el esquemático.



**Botón de pánico PANIC (PIN 11):** es un pulsador al que se le puede dar prioridad según la aplicación. El sistema es flexible y puede no habilitarse el botón de pánico si no se lo necesita o bien generar la interrupción correspondiente.

## MÓDULO GSM Y GPS SIM908

Para el sistema de comunicaciones celular se utilizó el módulo GSM SIM908 de SIMCOM, el mismo permite realizar llamadas, enviar mensajes de texto y realizar comunicaciones de datos a través de GPRS. Además incorpora el módulo GPS en el mismo encapsulado, estudiaremos este módulo más adelante. Se controla a través del microcontrolador por medio de comandos denominados AT. Es muy sencillo controlar un módulo GSM ya que los comandos se envían a través de la USART (interface serie del micro).



### Módulos SIM908 de SIMCOM. Medidas: 30x30x3,2 mm

Cuatribanda Mundial: GSM850, EGSM900, DCS1800, PCS1900

GPRS Multi-Slot Clase 10 y Clase 8.

Supresión de ruido.

Cancelación de eco.

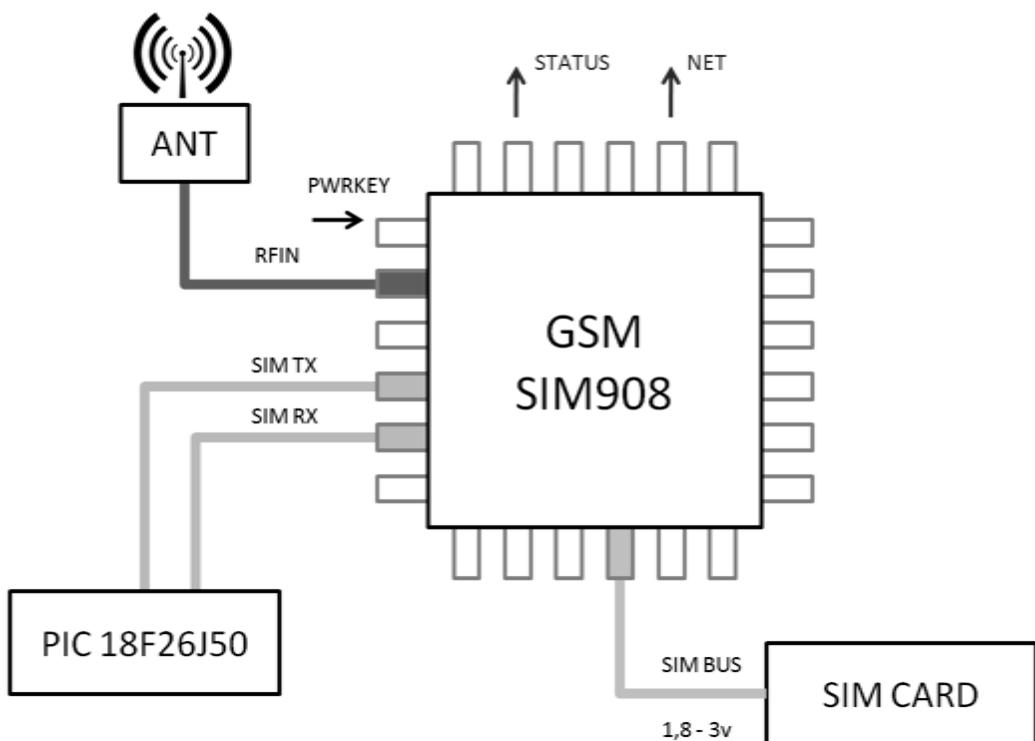
Tensión de alimentación entre 3,2 y 4,8 v.

Consumo de corriente en modo sleep: 1.5 mA

Puede encontrar más información sobre este módulo y descargar la hoja de datos completa desde la web del fabricante: [www.sim.com](http://www.sim.com)

## CONEXIONES DEL MÓDULO GSM

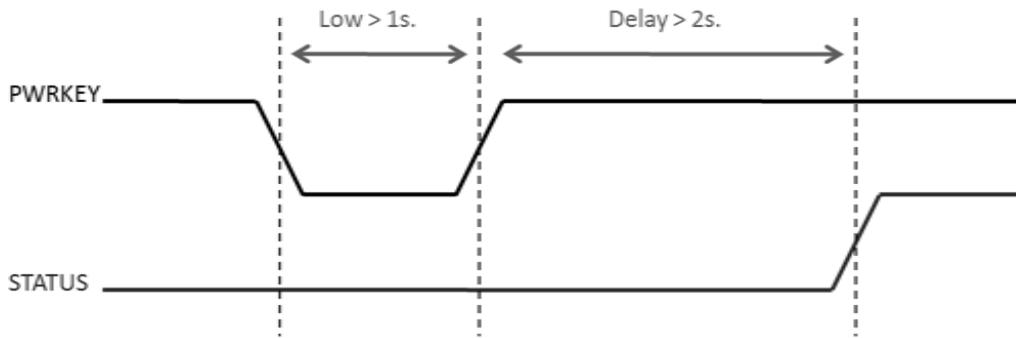
El módulo SIM908 se conecta al PIC a través de los pinos SIM TX (**RP0**) y SIM RX (**RP1**). Además necesita el conector de la antena y la tarjeta SIM para operar. Los pinos de Status y NET nos dan una idea del estado del módulo. Es conveniente conectar LEDs indicadores en ambos pinos. Cuando el módulo está listo para operar el PIN Status se coloca en 1, esto ocurre generalmente después de un reset. Por otro lado el LED conectado al PIN NET parpadea cada 2 segundos cuando el modulo está conectado y registrado a la red celular.



*Conexiones para GSM del modulo SIM908*

## ENCENDIDO DEL MÓDULO GSM

El módulo SIM908 necesita un 0 en el pin PWRKEY para encender. En este caso la placa tiene un transistor en emisor común (inversor), con lo debemos poner el pin RC1 (**GSMON**) del PIC en 1 para encender el módulo.



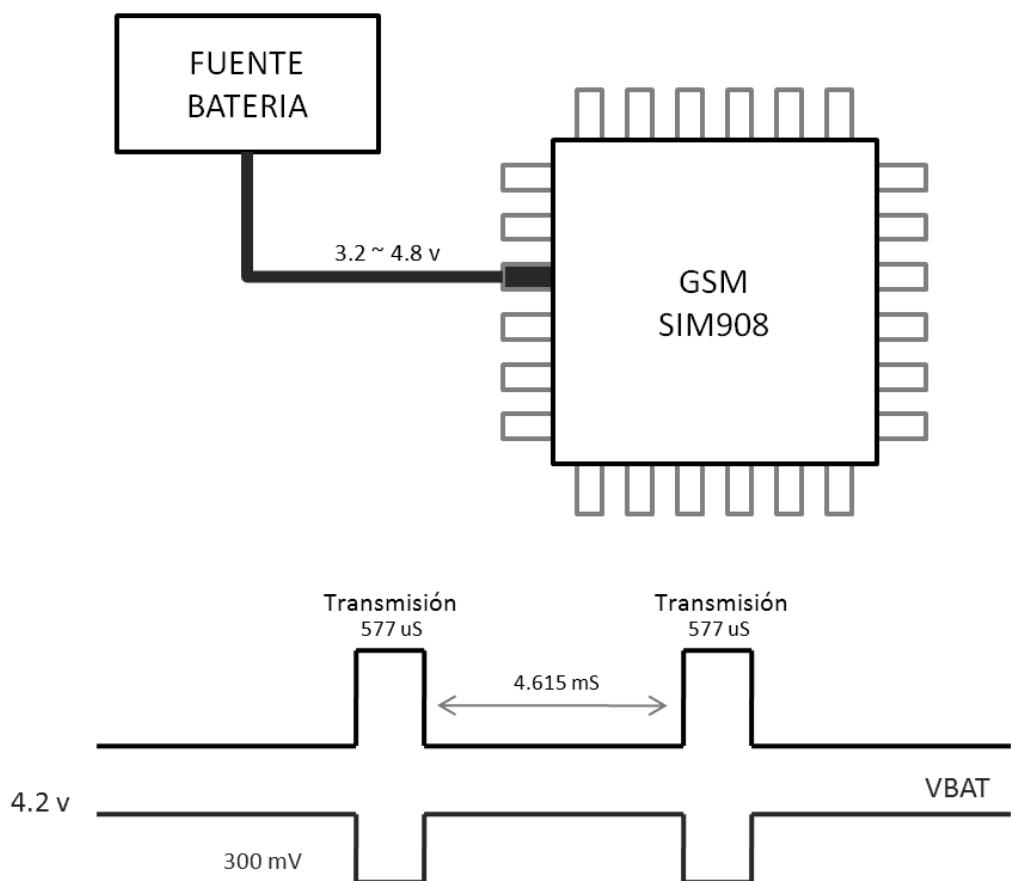
*El módulo se apagará automáticamente (STATUS = low) cuando se detecte tensión de alimentación fuera del rango  $3.3 < \text{VBAT} < 4.6$  o cuando la temperatura esté por encima de  $85^\circ\text{C}$ . Sin embargo 0.1v antes de los límites envía warnings por la USART.*

## CONSIDERACIONES IMPORTANTES

Debemos tener especial cuidado al diseñar la fuente de alimentación del módulo SIM908 ya que el módulo se reinicia automáticamente ante una caída de tensión mayor o igual a 300 mV.

La fuente debe proveer una tensión estable entre 3.2 y 4.8 v y además debe poder entregar 2A durante los periodos de transmisión. Es importante destacar que el módulo consume picos de hasta 2A de amplitud durante una llamada telefónica o una comunicación de datos, si la fuente no es capaz de entregar esta corriente, la tensión va a caer y en consecuencia se va a reiniciar el módulo.

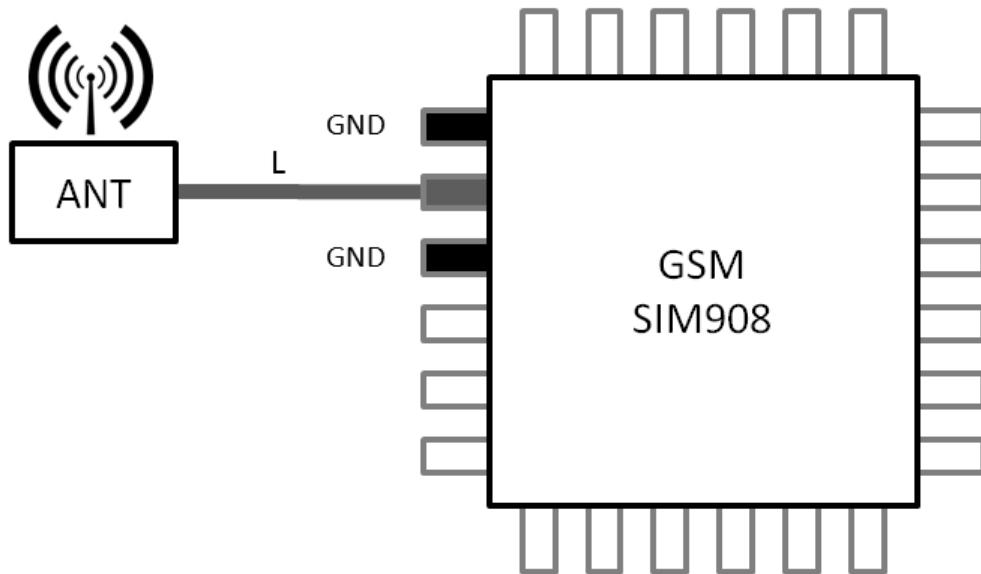
Para evitar esto podemos utilizar un regulador de tensión que soporte hasta 2A pero por lo general son costosos y relativamente grandes. Otra alternativa es colocar un capacitor de aproximadamente 1000uF SMD en paralelo con la fuente de alimentación para que este absorba el pico de corriente.



## CONSIDERACIONES IMPORTANTES

Otra cuestión fundamental es el diseño de la línea de transmisión que comunica el módulo SIM908 con la antena GSM. Para que las impedancias estén adaptadas, esta línea debe presentar una  $Z_0 = 50$  ohm para la frecuencia de trabajo (Bandas celulares que oscilan entre 850 y 1900 MHz).

Hay diversos programas de diseño electrónico que nos permiten calcular el ancho de la pista a partir de la impedancia deseada y otros parámetros como el espesor de la placa, el espesor del cobre y la constante dieléctrica del material. También es importante mencionar que si logramos hacer  $L \ll \lambda$  donde  $L$  es el largo de la pista y  $\lambda$  la longitud de onda a la frecuencia de trabajo, los efectos de atenuación se hacen despreciables.



*La línea de transmisión desde la antena al PAD del módulo debe tener una impedancia de 50 ohm.  $\lambda = 16$  cm,  $L = 0.5$  cm*

## CÁLCULO DE LA LINEA DE TRANSMISIÓN

Las líneas de transmisión que se realizan con el mismo sustrato que las pistas sobre el PCB se llaman microstrip. En este caso utilizamos una aplicación que se puede descargar desde <http://www.eeweb.com/toolbox/microstrip-impedance> para calcular el ancho de la pista adecuado y lograr una impedancia de aproximadamente 50 ohm.

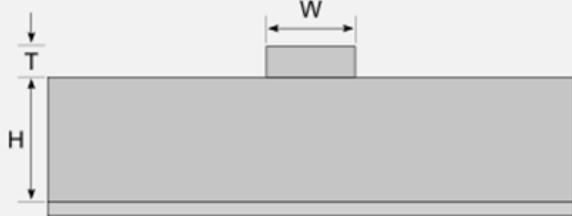
EEWeb

Microstrip Impedance Calculator

**Choose Type**

- Microstrip
- Embedded Microstrip
- Symmetric Stripline
- Asymmetric Stripline
- Wire Microstrip
- Wire Stripline
- Edge-Coupled Microstrip
- Edge-Coupled Stripline
- Broadside-Coupled Stripline

**Microstrip Impedance Calculator**



**Inputs**

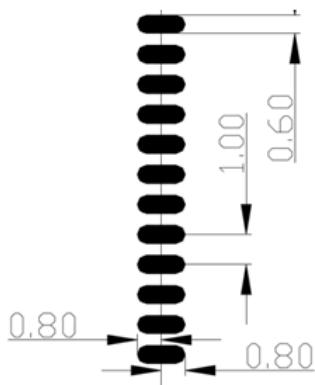
Trace Thickness	T	0.4	mm
Substrate Height	H	1.2	mm
Trace Width	W	1.65	mm
Substrate Dielectric	Er	4.7	

**Output**

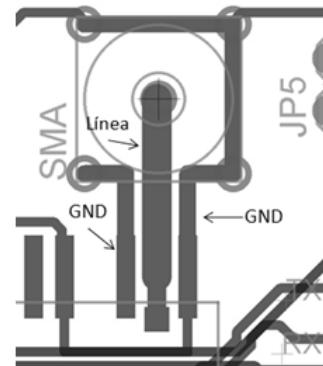
Impedance (Z): 50.3 Ohms

## CÁLCULO DE LA LINEA DE TRANSMISIÓN

Como se puede ver en la aplicación el ancho de la pista adecuado es de 1,65 mm, sin embargo llevar este ancho a la práctica es imposible ya que la separación de los pinos del módulo GSM es de 1,00 mm.



(1) Diagrama del modulo GSM



(2) Implementación en el PCB

Para solucionar esto hay 2 alternativas, una es fabricar un PCB multicapa, donde el plano de masa es una de las capas interiores y así la separación entre la línea y GND es mucho menor permitiendo un ancho de pista de aproximadamente 0.7 mm para la misma impedancia. Sin embargo el costo de fabricar un PCB multicapa es más del doble un PCB doble faz.

Otra alternativa es la que se ve en el esquema (2) donde la pista se ensancha a medida que se aleja del módulo GSM. De todas maneras no se llega al ancho de 1,65 mm, pero es una solución de compromiso ya que este método no incrementa el costo de la placa.

En este caso se eligió la segunda opción, además debemos tener en cuenta que la longitud de la pista es mucho menor que la longitud de onda a la frecuencia de trabajo (alrededor de 10 veces menos) lo que reduce la atenuación.

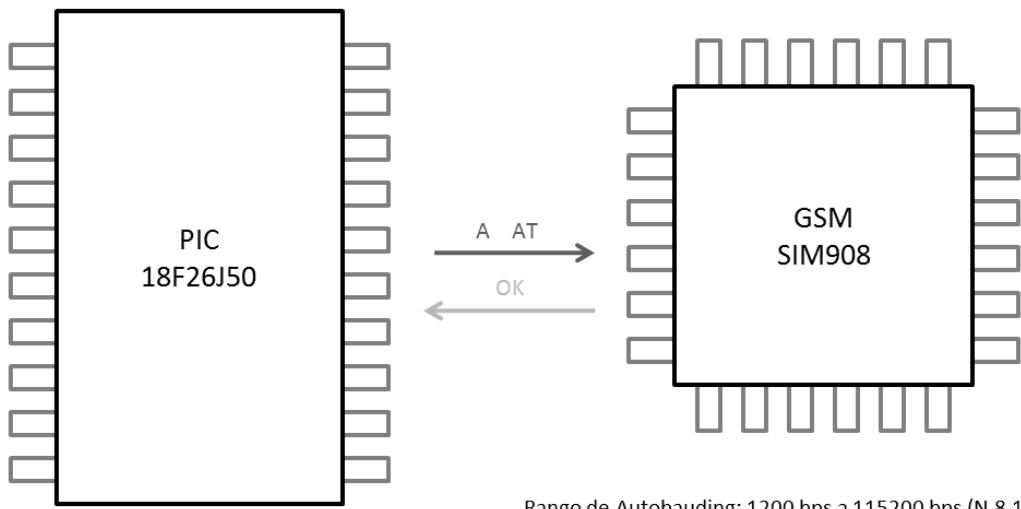
## COMUNICACIÓN CON EL MÓDULO

El módulo GSM SIM908 se maneja por medio de comandos a través de la USART. Estos comandos se denominan comandos AT.

Los comandos AT permiten realizar llamadas, enviar SMS, consultar los parámetros del modulo, medir la señal celular y realizar todas las configuraciones del SIM908, de esta manera es posible seleccionar el medio de comunicación que mejor se adapte a la aplicación correspondiente.

La comunicación es tipo conversación. Siempre se debe esperar la respuesta antes de enviar el próximo comando.

Por defecto esta autobauding habilitado (capacidad que tienen los sistemas de comunicaciones de trabajar a cualquier velocidad dentro de un rango preestablecido). Para sincronizar la velocidad del PIC con el SIM900 se debe enviar A y esperar de 3 a 5 segundos. Luego enviamos el comando AT antes de iniciar la comunicación. Una vez sincronizados el SIM908 responde OK.



## COMANDOS AT BÁSICOS

Estos comandos pueden enviarse a través del HyperTerminal (utilizando el driver USB-GSM diseñado especialmente) o bien a través del microcontrolador sin utilizar la computadora.

Por ejemplo:

```
ATI  
SIMCOM_Ltd  
SIMCOM_SIM908  
Revisión:1008B10SIM900S32_(SPANSION)SIMCOM_Ltd
```

AT+GSN // número IMEI del SIM900

**355117001566879**

**OK**

AT+CSQ // Nivel de señal [rss,ber]

rss (0= -113dBm o 1= -111dBm o 2 a 30= -109dBm a -53dBm o 31= -51dBm  
o >/99 no se conoce). ber (En % bit error rate)

**[17,0]**

ATD51065802; Realizar una llamada - SIM908 disca 51065802

AT+CLIP=1 // Mostrar el numero de la llamada entrante (Caller ID)

**OK**

**RING**

+CLIP: "51065802",129 // Número de llamada entrante 51065802

**OK**

ATH // Colgar

**RING**

ATA // Atender una llamada entrante

**OK**

## COMANDOS AT BÁSICOS

Estos comandos pueden enviarse a través del HyperTerminal (utilizando el driver USB-GSM diseñado especialmente) o bien a través del microcontrolador sin utilizar la PC.

Por ejemplo:

Envío de SMS al 51065802

```
AT+CMGF=1 // Seleciono modo texto
AT+CMGS="51065802" <ENTER> // Número de teléfono y 0x0D
> SU MENSAJE <CTRL+Z> // Escriba el mensaje y 0x1A
Devuelve la posición en la memoria de enviados
OK
+CMGS: 123
```

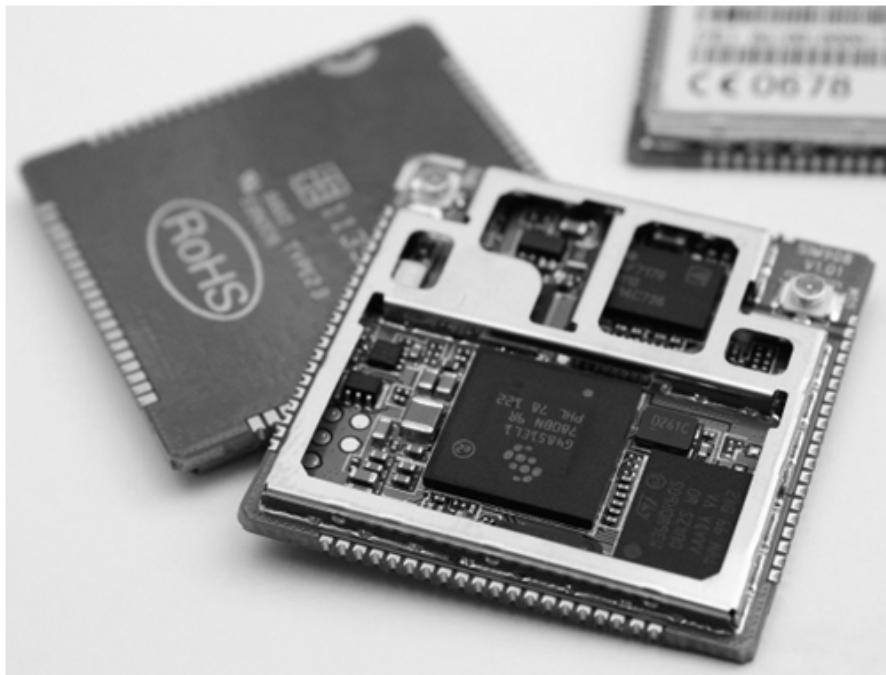
Recepción de SMS

```
AT+CMGF=1 // modo texto
AT+CNMI=2,2,0,0,0 // no lo guarda en memoria lo envia por USART
Al llegar un mensaje lo envía por TX automáticamente.

+CMT:"12345678","","28/05/10,11:20:28+32"
Hola Mundo
```

## MÓDULO GPS

Como mencionamos el módulo SIM908 incluye un receptor GPS: Las características más sobresalientes son: receptor GPS basado en el SIRFstarIII - 42 canales. Consumo menor a 77 mA. Protocolos NMEA o SIRF. Adquisición rápida de satélites. 30 s. Alimentación: 3.2~4.8V. Sensibilidad de tracking: -160dBm



*El GPS es compatible con el estándar SIRIII lo que nos da una precisión de 2.5m al determinar la ubicación.*

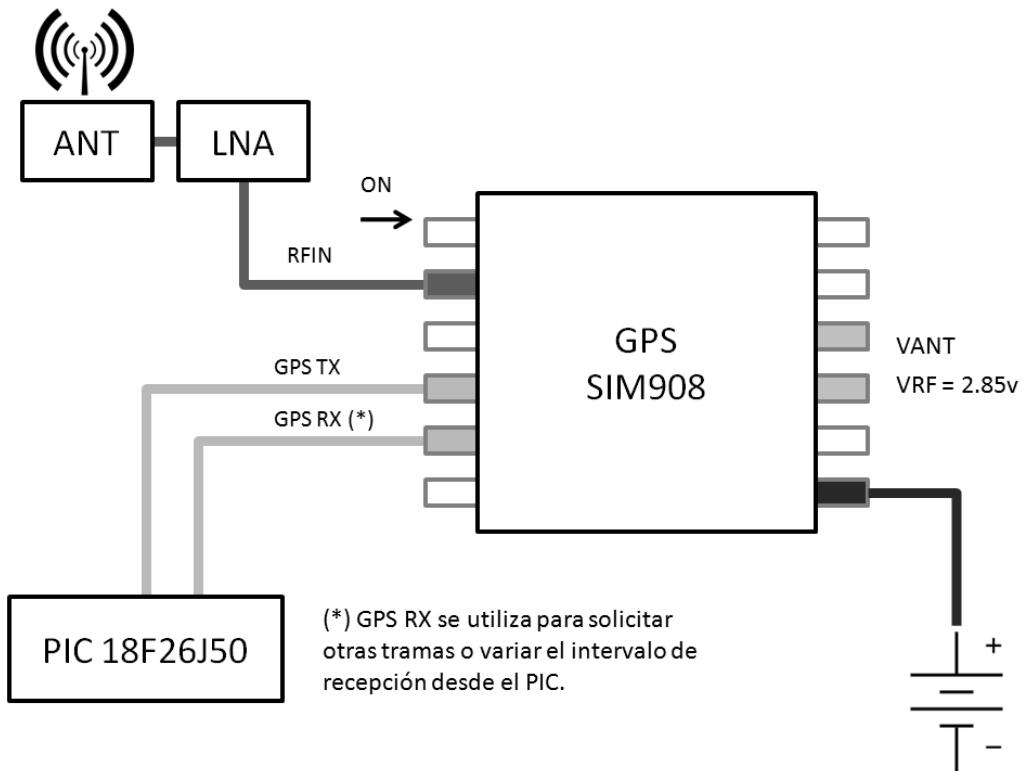
## CARACTERISTICAS PRINCIPALES DEL MÓDULO GPS

El consumo está en el orden de los 77 mA durante la adquisición de satélites y baja levemente en tracking. Un parámetro que no pudimos comprobar fue el tiempo de arranque en frío, según esta tabla extraída de la hoja de datos es de 30 s (es el tiempo que tarda en determinar la ubicación luego del encendido). En la práctica nunca logramos bajar de los 60 s. a cielo abierto.

Parameter	Description	Performance				Unit
		Min	Typ	Max		
<i>Horizontal Position Accuracy<sup>(a)</sup></i>	<i>Autonomous</i>		2.5			<i>m</i>
<i>Velocity Accuracy<sup>(b)</sup></i>	<i>Speed</i>	-	0.01	-		<i>m/s</i>
	<i>Heading</i>	-	0.01	-		°
<i>Time To First Fix<sup>(c)</sup></i>	<i>Hot start</i>	-	1	-		<i>s</i>
	<i>Cold start</i>	-	30	-		<i>s</i>
<i>Sensitivity</i>	<i>Autonomous acquisition</i>		-143			<i>dBm</i>
	<i>Tracking</i>		-160			<i>dBm</i>
<i>Receiver</i>	<i>Channels</i>		42			
	<i>Update rate</i>		1			<i>Hz</i>
	<i>Altitude</i>			18288		<i>km</i>
	<i>Velocity</i>			1850		<i>km/h</i>
	<i>Tracking L1, CA Code</i>					
	<i>Protocol support NMEA, OSP</i>					
<i>Power consumption<sup>(d)</sup></i>	<i>Continuous tracking</i>		76			<i>mA</i>
	<i>acquisition</i>		77			
	<i>Power down current</i>		0.03			<i>uA</i>

## CARACTERISTICAS PRINCIPALES DEL MÓDULO GPS

El módulo GPS se conecta al PIC a través de los pines GPS TX (**RP3**) y GPS RX (**RP4**). Además necesita la antena y el LNA (Low Noise Amplifier) para operar.



*Conexiones básicas del GPS.*

## TRAMA NMEA DEL MÓDULO GPS

NMEA 0183 es un protocolo a través del cual pueden comunicarse los instrumentos de navegación marítima y terrestre. Ha sido definido, y está controlado, por la organización estadounidense National Marine Electronics Association.

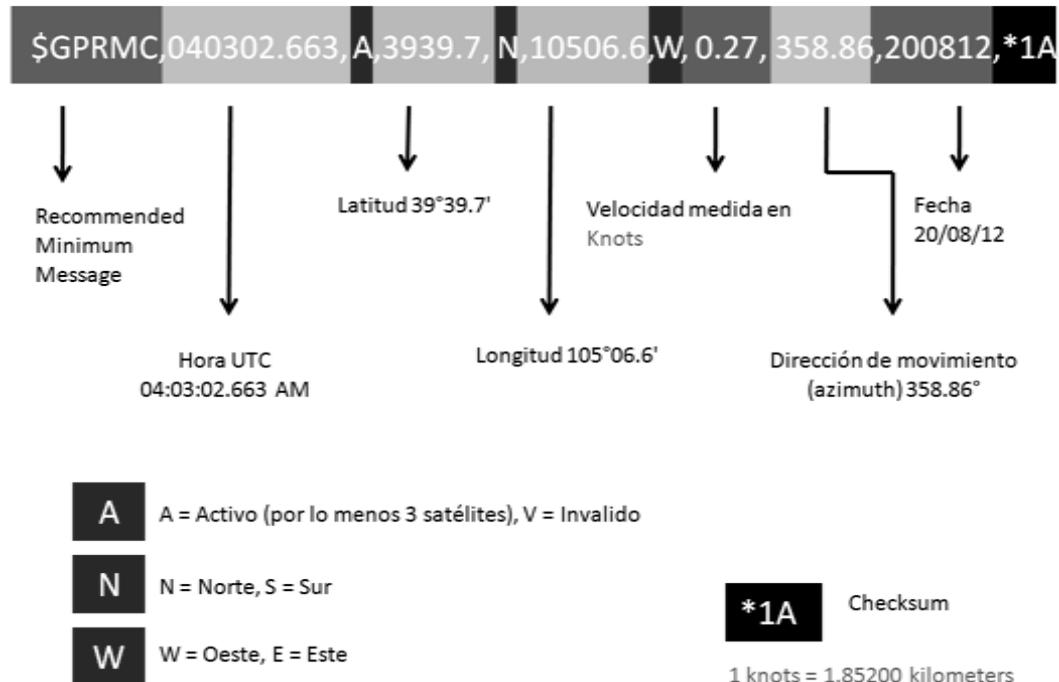
En la trama NMEA tenemos información como posición, velocidad, dirección de desplazamiento, hora UTC, posición de los satélites, intensidad de la señal que se recibe y datos de control.

```
$GPRMC,031015.593,A,3435.7639,S,05823.0162,W,0.32,154.54,250510,,A*68
$GPGGA,031016.593,3435.7640,S,05823.0162,W,1,04,2,6,83.8,M,14.6,M,,0000*64
$GPGSA,A,3,20,07,08,04,,4,5,2,6,3,7*3A
$GPGSV,3,1,11,25,79,292,,13,61,184,,27,58,311,,07,58,317,43*70
$GPGSV,3,2,11,23,47,127,,20,43,046,42,04,36,266,36,02,27,244,20*7E
$GPGSV,3,3,11,08,21,321,26,16,20,123,,10,11,225,21*44
$GPRMC,031016.593,A,3435.7640,S,05823.0162,W,0.31,152.77,250510,,A*61
$GPGGA,031017.593,3435.7640,S,05823.0162,W,1,04,2,6,83.7,M,14.6,M,,0000*64
$GPGSA,A,3,20,07,08,04,,4,5,2,6,3,7*3A
$GPRMC,031017.593,A,3435.7640,S,05823.0162,W,0.33,155.66,250510,,A*65
$GPGGA,031018.593,3435.7640,S,05823.0161,W,1,04,2,6,83.5,M,14.6,M,,0000*64
$GPGSA,A,3,20,07,08,04,,4,5,2,6,3,7*3A
$GPRMC,031018.593,A,3435.7640,S,05823.0161,W,0.34,156.14,250510,,A*68
$GPGGA,031019.593,3435.7636,S,05823.0162,W,1,04,2,6,82.5,M,14.6,M,,0000*66
$GPGSA,A,3,20,07,08,04,,4,5,2,6,3,7*3A
$GPRMC,031019.593,A,3435.7636,S,05823.0162,W,0.29,152.87,250510,,A*69
```

*Trama NMEA obtenida del módulo GPS.*

## TRAMA NMEA DEL MÓDULO GPS

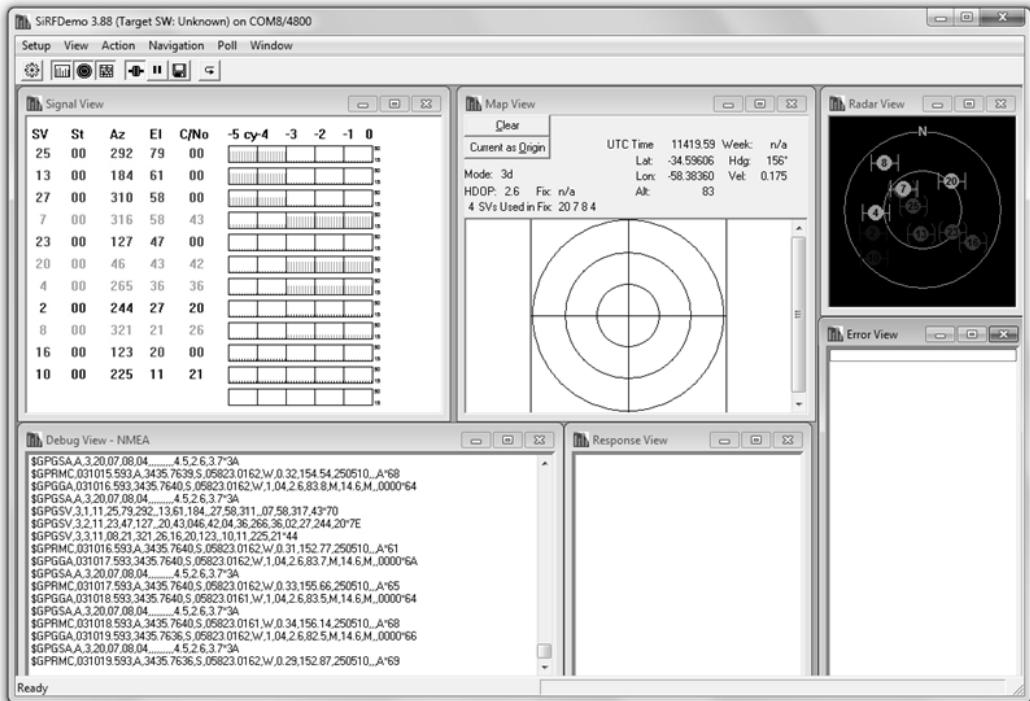
De las diversas tramas que envia el GPS, la trama \$GPRMC contiene la información básica de rastreo y es la que vamos a utilizar para determinar la posición y demás datos como velocidad, dirección y hora UTC. Veamos cómo está compuesta la trama GPRMC:



*Trama GPRMC con la información básica de localización.*

## TRAMA NMEA DEL MÓDULO GPS

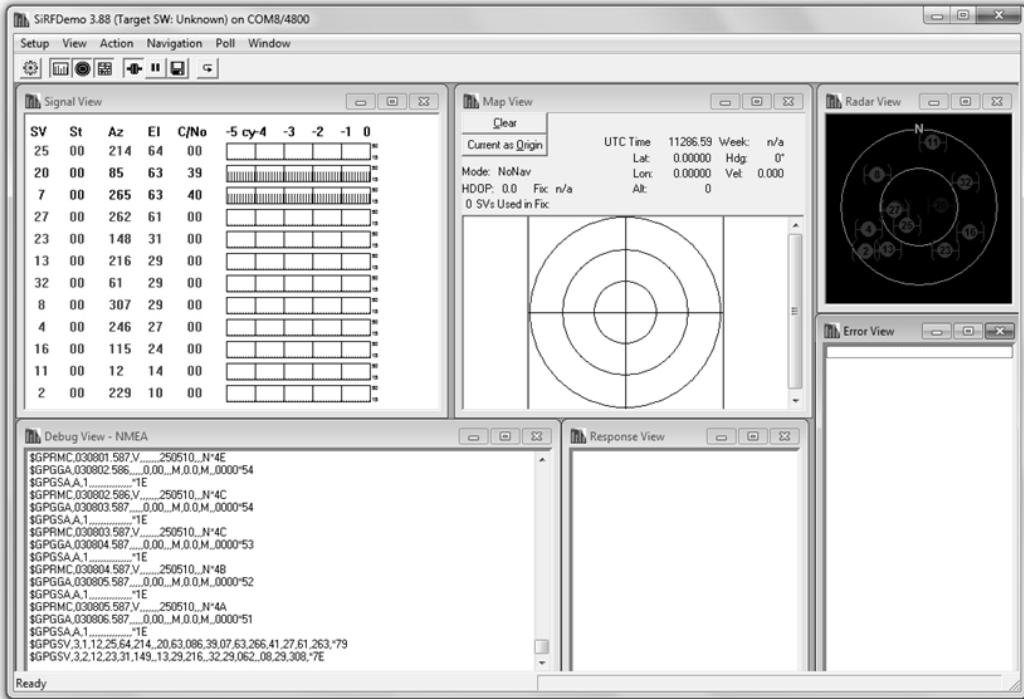
La placa TrackMe (GPS) incluye un programa cargado en el microcontrolador PIC que convierte la información proveniente del módulo GPS a USB, de esta forma es posible conectar la placa a una computadora. Para decodificar la trama NMEA del GPS y mostrar la información en forma gráfica es necesario un software llamado SIRFDemo que corre bajo Windows. Este programa es compatible con todos los receptores GPS que trabajen bajo las normas NMEA o SIRF.



*Captura de pantalla del software conectado a la placa TrackMe y recibiendo las tramas NMEA por USB.*

## TRAMA NMEA DEL MÓDULO GPS

Se requieren por lo menos 3 satélites para determinar la posición. En este caso todos los satélites están en rojo por lo que aun no hay señal suficiente. Cabe desatascar que el GPS debe tener una vista clara del cielo para funcionar correctamente.

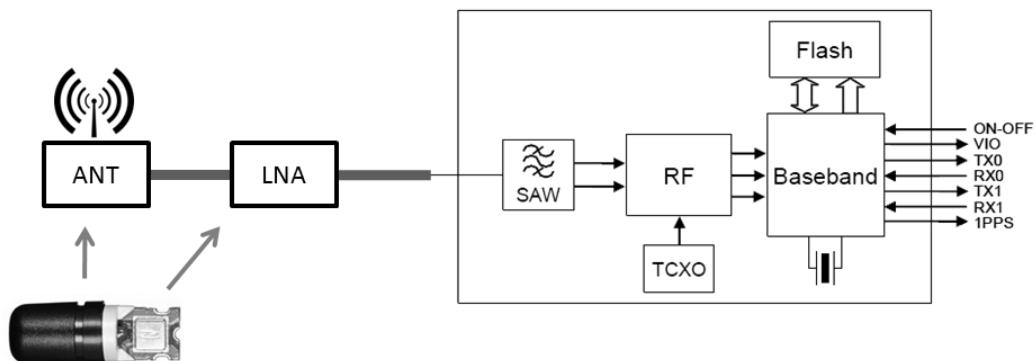


*En este caso no se llega a determinar la ubicación por la falta de visibilidad de satélites. Es fundamental contar con una vista clara del cielo para que el sistema GPS trabaje adecuadamente.*

## CONSIDERACIONES IMPORTANTES

Al igual que con el módulo GSM en este caso es crítico el diseño de la pista que vincula el módulo GPS con la antena, pero además debemos colocar un LNA, para filtrar y amplificar la señal del satélite.

Es necesario un LNA (Low Noise Amplifier) próximo a la antena para amplificar la señal del GPS. 1.57542 GHz



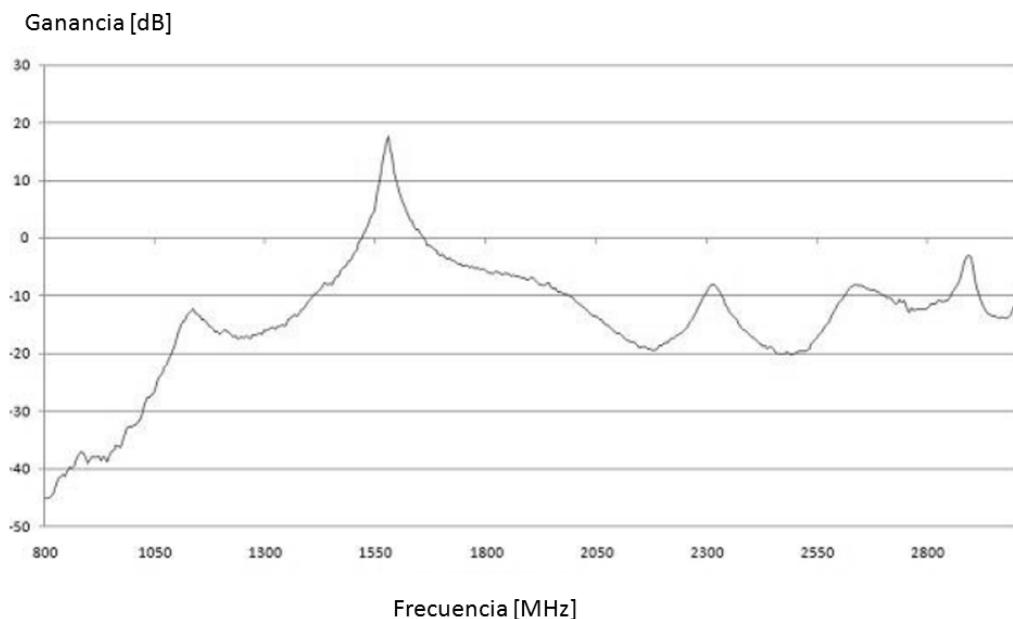
*La antena SL1204 de Sarantel incluye el LNA (es una antena activa). Las antenas activas deben ser alimentadas para energizar el amplificador. El módulo GPS del SIM908 puede alimentar la antena a través del mismo pin por el que recibe la señal (RFIN)*

## AMPLIFICADOR DE BAJO RUIDO

La señal que recibimos de los satélites es muy tenue y puede contener un alto nivel de ruido. Más aun si el receptor GPS se encuentra próximo a módulos de comunicaciones GSM como en este caso.

Por eso, para amplificar la señal del GPS y optimizar la recepción, se hace indispensable contar con un LNA o Amplificador de bajo ruido. Se llama de bajo ruido, porque introduce muy poco ruido a la señal original.

Veamos cual es la respuesta en frecuencia de un LNA:



Frequency (MHz)		$S_{12}$ (dB)
860	GSM 900	-40
		-37
1575.42	GPS (L1)	+18
1700	GSM 1800	-4
		-7
		-9
2450	Bluetooth/Wifi	-21

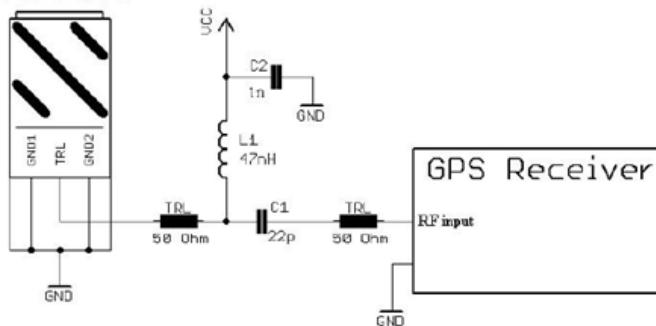
*Tabla de frecuencias y bandas del GPS.*

## AMPLIFICADOR DE BAJO RUIDO

Un LNA como todo amplificador requiere alimentación para funcionar, en esta caso el LNA integrado en la antena consume 3.3 mA y una tensión dentro del rango 1,8 v y 3,6v. Hay dos formas de proveer esta alimentación:

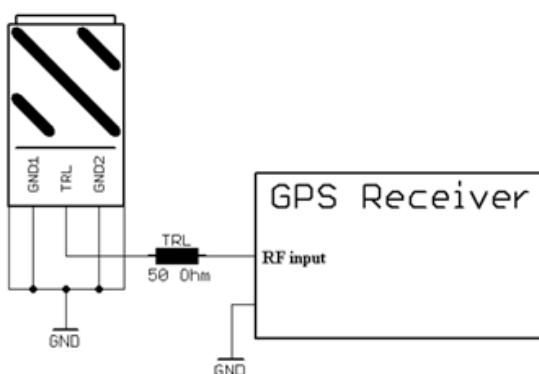
1 - Conectando la antena por medio de un inductor a una fuente externa.

En este caso la fuente entrega la alimentación necesaria pero habría que desconectarla cuando el modulo GPS no esté operando para ahorrar energía.



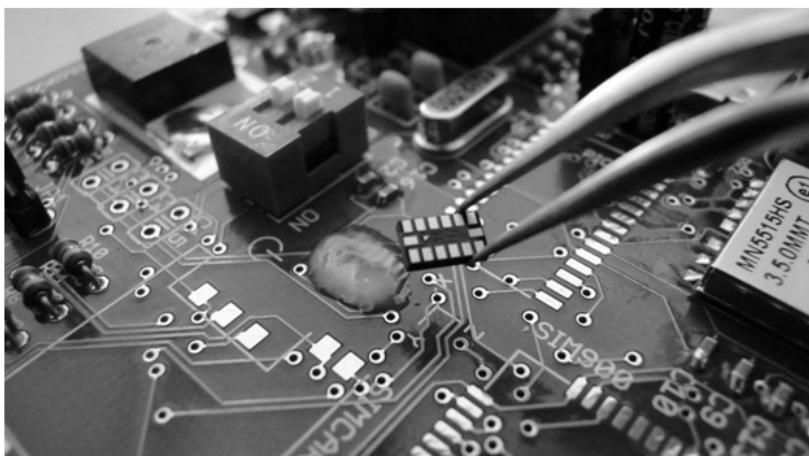
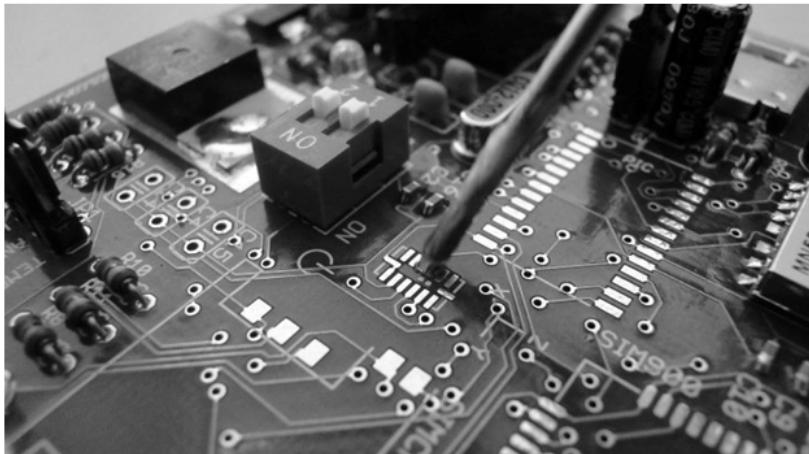
2 - Conectando la antena directamente a la entrada del módulo GPS

Los receptores GPS modernos cuentan con la posibilidad de alimentar la antena activa directamente a través del mismo PIN que reciben la señal. Esto ahorra componentes externos y desconecta el LNA cuando no se está utilizando el módulo receptor.



## OTROS DISPOSITIVOS QUE INTEGRAN EL SISTEMA

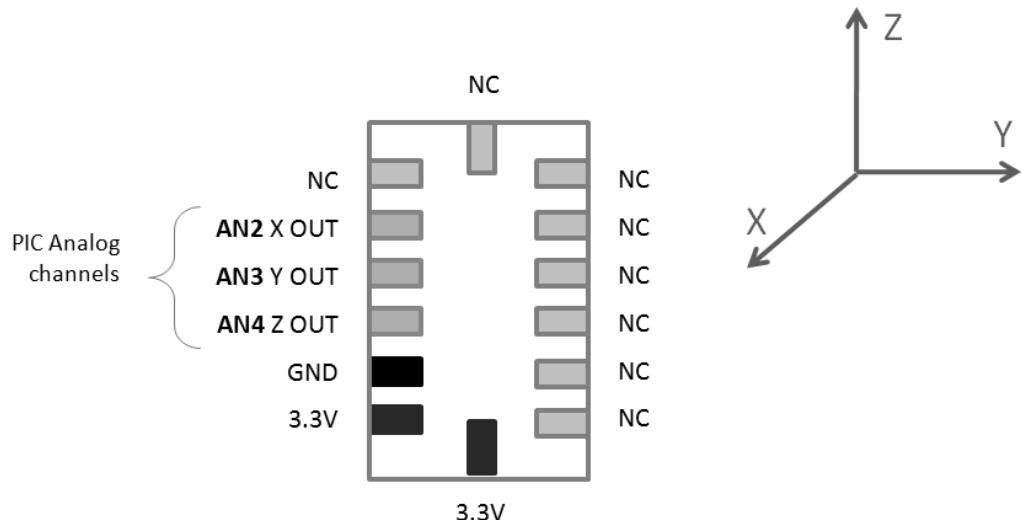
El acelerómetro es sin duda el dispositivo más complicado de montar, no solo por su reducido tamaño, sino por su encapsulado QFN y además porque cualquier inclinación daría como resultado una medición errónea.



*Para el proceso de soldado se utilizó aire caliente, en primer lugar se coloca flux sobre la placa y luego se calienta para hacer más receptivo al estaño. Una vez colocado el acelerómetro en posición se calienta nuevamente con un flujo de aire suave para no desplazarlo de su posición.*

## ACELERÓMETRO

Es un acelerómetro analógico de 3 ejes (X OUT, Y OUT, Z OUT). Están conectados directamente a los canales analógicos del PIC (ver esquemático).



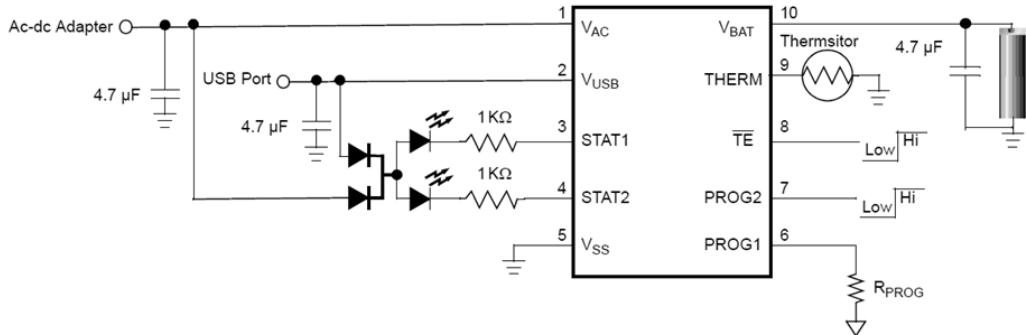
Freescale MMA7368L

**$\pm 1.5g$  Three Axis Low-g Micromachined Accelerometer. La tasa de cambio es de  $800mV/g$ .**

## CARGADOR DE BATERIA

La placa incorpora un cargador de baterías de Microchip especialmente diseñado para utilizar el puerto USB. Este cargador toma la alimentación del puerto USB de la computadora y limita la corriente a 500mA (máximo disponible por norma). Además cuenta con protecciones contra cortocircuito y exceso de temperatura.

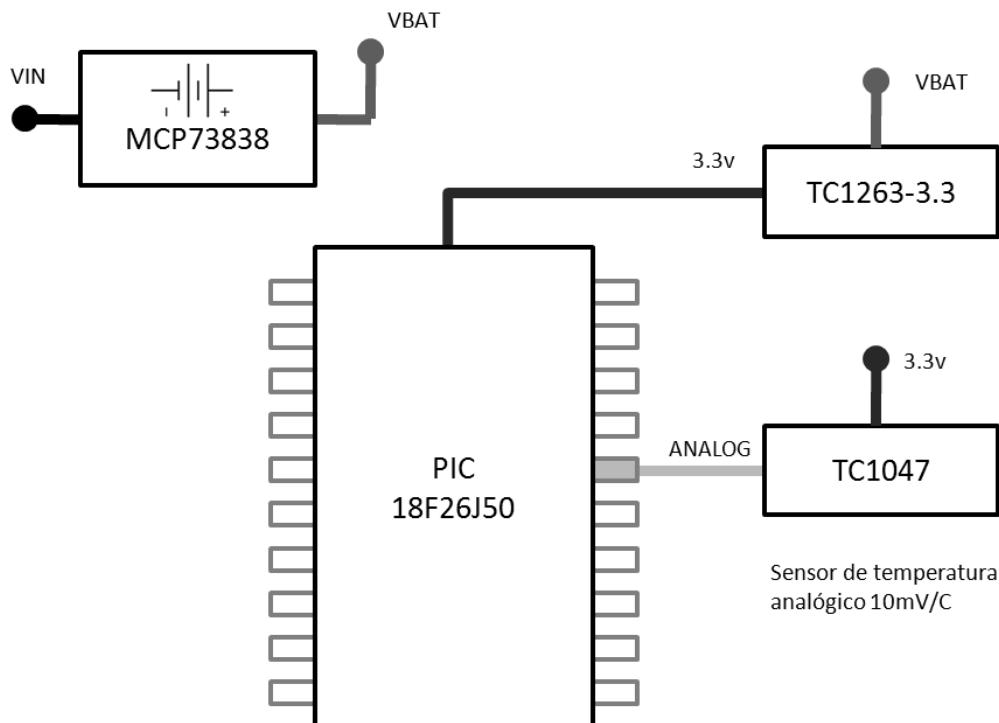
A continuación se puede ver el cargador MCP73838 y el circuito eléctrico necesario.



*Otra característica para destacar de este cargador es que puede aceptar simultáneamente alimentación del puerto USB y de la red eléctrica, si ambas fuentes están presentes, automáticamente se conecta a la red y no toma corriente del USB.*

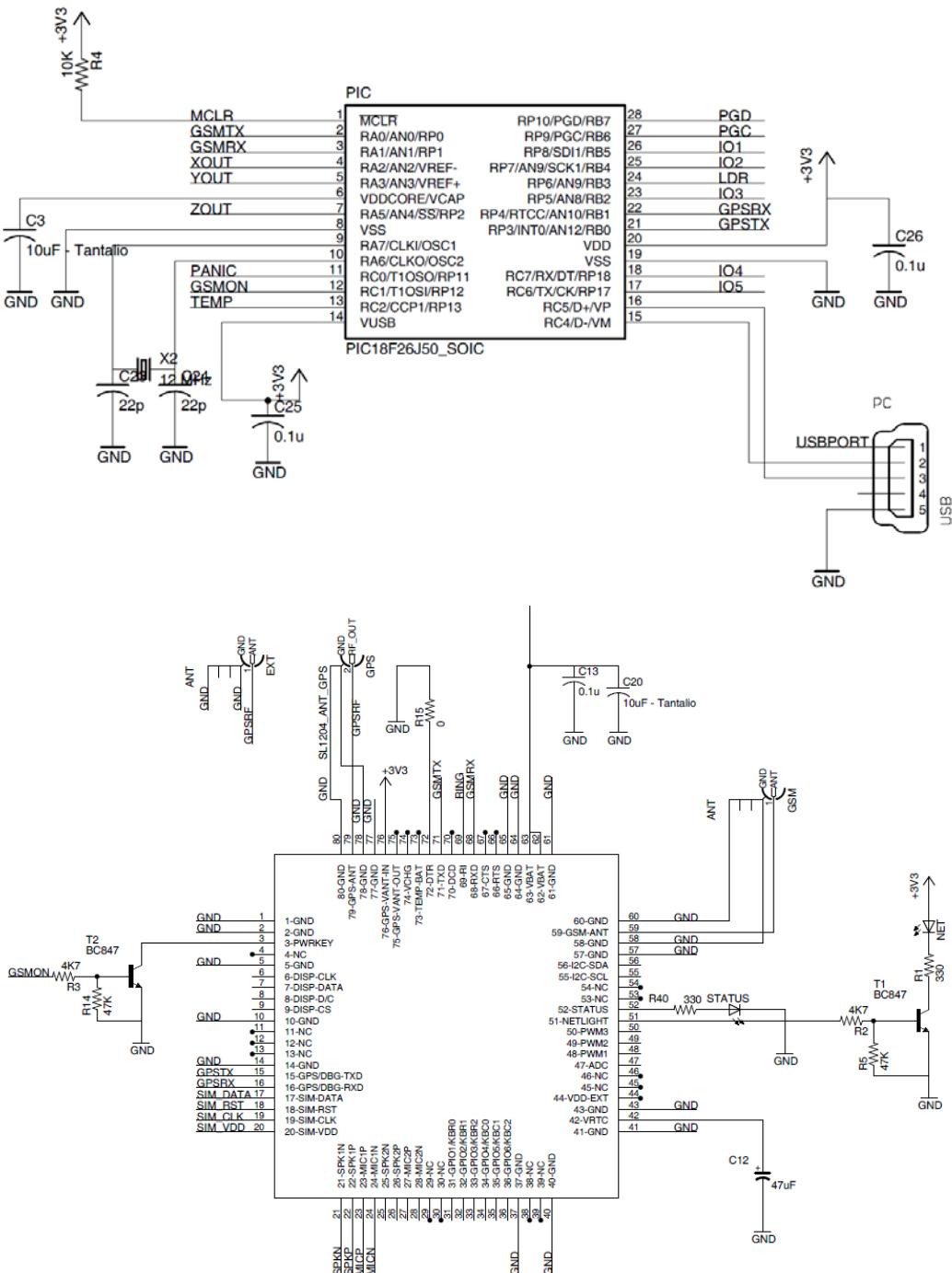
## ESQUEMÁTICO

La placa cuenta con un pulsador de Pánico que se conecta al PIC a través de PANIC. También es posible utilizar la entrada digital para otro dispositivo desconectado el Jumper JP3. En este tipo de aplicaciones alimentadas a batería es fundamental que el regulador sea LDO (low dropout), es decir que la diferencia de tensión entre Vin y Vout sea la menor posible. En este caso se utilizó el regulador TC1262-3.3 (3.3v) de Microchip que posee una tensión de funcionamiento de 200 mV a 500 mA. La corriente máxima es de 500 mA.

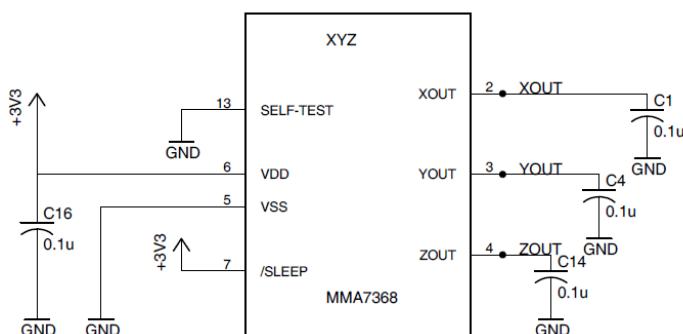
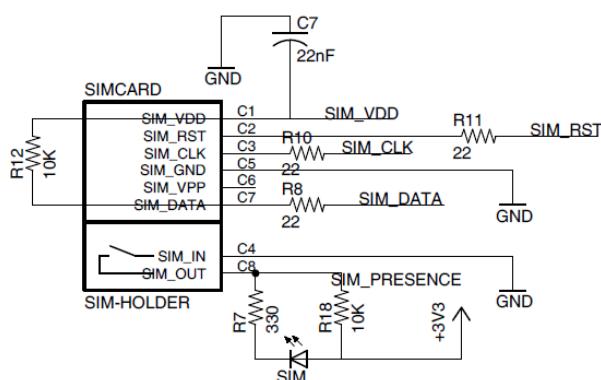
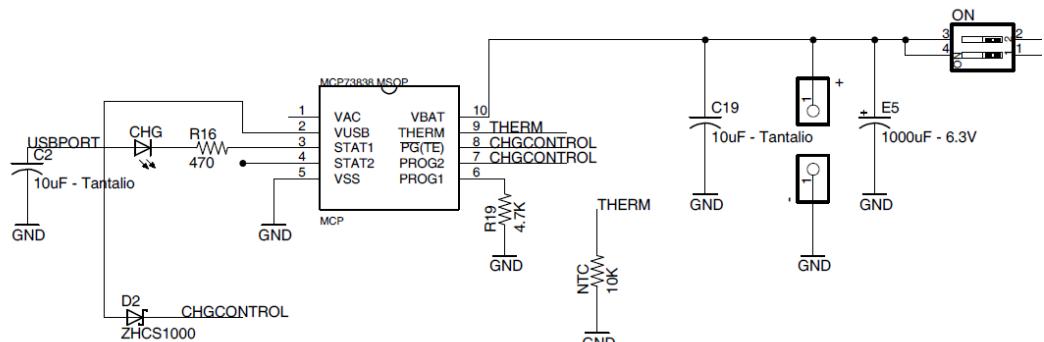


*El esquema muestra los dispositivos antes mencionados y su conexión al microcontrolador. Todos los componentes de la placa obtienen los 3.3v del regulador TC1262-3.3 a excepción del módulo GSM que está conectado directamente a la batería.*

## ESQUEMÁTICO COMPLETO

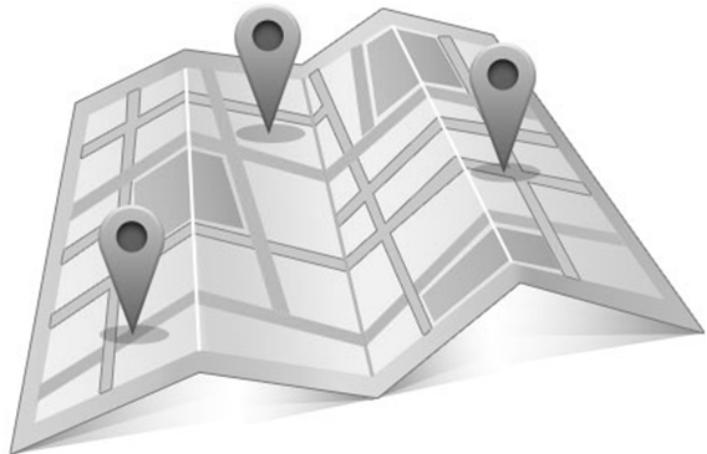


## ESQUEMÁTICO COMPLETO



**Cargador de baterías, sim holder y acelerómetro. Puede encontrar el esquemático en PDF en el DVD que acompaña este libro.**





# EJEMPLOS Y APLICACIONES



## EJEMPLOS Y APLICACIONES

### Inicialización del módulo SIM908

Es importante respetar los delays para inicializar el modulo correctamente. Una vez que el módulo está inicializado podemos realizar todo tipo de comunicaciones celulares.

```
void inicializacionSIM900(void)
{
    int cont;
    delay_ms(1000);
    puts("A",SERIAL_GSM); //para sincronizar el baudrate del SIM 900
    delay_ms(300);
    putchar(0xd,SERIAL_GSM);
    delay_ms(3000);

    for(cont=0;cont<5;cont++) // mando varios AT para el autobauding
    {
        puts("AT",SERIAL_GSM);
        delay_ms(300);
        putchar(0xd,SERIAL_GSM);
        delay_ms(200);
        output_high(LED);
        delay_ms(200);
        output_low(LED);
    }

    puts("AT+CMGF=1\r",SERIAL_GSM); // configuro para que trabaje en modo texto
    delay_ms(100);
    putchar(0xd,SERIAL_GSM);
    delay_ms(100);
    return;
}
```

*El código completo se encuentra en el DVD. Todos los ejemplos están programados en C para el compilador CCS.*

## EJEMPLOS Y APLICACIONES

### Leer canales analógicos

En este ejemplo simplemente se seleccionan los canales analógicos del acelerómetro y el sensor de temperatura y se guardan los valores en variables (no se muestra la definición en el recuadro)

```
#define X_ADC 2          // Defino el canal analógico
#define Y_ADC 3
#define Z_ADC 4
#define TEMP_ADC 11
set_adc_channel(X_ADC);      // Selecciono el canal con el multiplexor
delay_ms(3);                // Defino un delay de 3 ms para cargar el capacitor de lectura
eje_x=read_adc();           // Guardo el valor en la variable
set_adc_channel(Y_ADC);
delay_ms(3);
eje_y=read_adc();
set_adc_channel(Z_ADC);
delay_ms(3);
eje_z=read_adc();
set_adc_channel(TEMP_ADC);
delay_ms(3);
temperature=read_adc();
```

## EJEMPLOS Y APLICACIONES

### Enviar la temperatura por SMS

El ejemplo completo consiste en leer el sensor TC1047, convertir el valor de tensión en grados y enviarlo por SMS cuando se presiona el botón de pánico. En este fragmento vemos que los datos deben ser pasados como string, con lo cual convertimos la variable grados de float a string con sprintf. Lo mismo aplica para enviar las mediciones del acelerómetro.

```
sprintf(string_temp,"Temp=%1.2f C ",grados);
fprintf(SERIAL_GSM,"AT+CMGS=\"1512345678\"\r"); // aca poner el número de celular que
recibe el SMS
delay_ms(200);
fputs(string_temp,SERIAL_GSM);
delay_ms(100);
fprintf(SERIAL_GSM,"%c",0x1a);
output_high(led);
while(input(PANIC)==0);
```

## EJEMPLOS Y APLICACIONES

### Leer y almacenar la trama NMEA del GPS

La idea es leer y almacenar una trama válida en una variable, para luego procesarla y enviarla como un link de GoogleMaps.

```
if ( kbhit (SERIAL2) )           // Espero un nuevo dato en la USART2
{
    e=0;
    e=getchar(SERIAL2);          // En la variable e se guardan los caracteres
    switch(e)
    {
        case 10:                // Espero el new line feed
        if(cbuffGPS[0]=='$' && cbuffGPS[1]=='G' && cbuffGPS[2]=='P' && cbuffGPS[3]=='R' &&
           cbuffGPS[4]=='M' &&
           cbuffGPS[5]=='C' && cbuffGPS[18]=='A' ) //esta linea valida si el dato es valido (detecto
al      menos 3 satelites)
        {
            PIN_ON(LED);
            strcpy(cbuffGPS2,cbuffGPS);           // La trama $GPRMC completa queda
guardada en      cbuffGPS2
        }
        xbuff=0;
        flag=0;
        break;
        default:

            cbuffGPS[xbuff++]=e;
            if(xbuff>(lenbuffGPS-1))
                xbuff=lenbuffGPS;
    }
}
```

## EJEMPLOS Y APLICACIONES

### Convertir al formato de GoogleMaps

Debemos obtener la latitud y longitud de la trama \$GPRMC (cbuffGPS2) y además convertir los valores a coordenadas decimales. Luego los parámetros quedan en las variables latitud y longitud.

```
if(cbuffGPS2[0]=='$' && cbuffGPS2[1]=='G' && cbuffGPS2[2]=='P' && cbuffGPS2[3]=='R' &&
cbuffGPS2[4]=='M' && cbuffGPS2[5]=='C')
{
    strncpy(latg,&cbuffGPS[20],2);           // convierto en coordenadas googlemaps
    strncpy(latm,&cbuffGPS[22],7);
    strncpy(longg,&cbuffGPS[32],3);
    strncpy(longm,&cbuffGPS[35],7);

    latitudgoogle=(atof(latm))/60;
    longitudgoogle=(atof(longm)/60);

    sprintf(&latitud[1],"%.6f",latitudgoogle);
    latitud[0]=latg[0];
    latitud[1]=latg[1];

    sprintf(&longitud[2],"%.6f",longitudgoogle);
    longitud[0]=longg[0];
    longitud[1]=longg[1];
    longitud[2]=longg[2];
}
```

## EJEMPLOS Y APLICACIONES

### Enviar un link compatible con GoogleMaps

Simplemente incluimos las variables latitud y longitud, con el formato adecuado, dentro de la URL de GoogleMaps.

```
printf(SERIAL_GSM,"AT+CMGS=\"1512345678\"\r"); // aca poner el numero de celular que  
recibe el SMS  
delay_ms(200);  
printf("http://maps.google.com/maps?q=-%s,+-%s+(TrackMe esta aqui)",latitud,longitud);  
delay_ms(100);  
fprintf(SERIAL_GSM,"%c",0x1a);  
output_high(led);  
while(input(PANIC)==0);
```



*El usuario recibe en su celular un SMS con un link que le muestra la ubicación de la placa.*

## EJEMPLOS Y APLICACIONES

### Enviar datos por GPRS

Podemos enviar la información a GoogleEarth en tiempo real a través de GPRS. El método para hacerlo consiste en enviar la trama NMEA completa (sin procesar) a un servidor TCP. Luego en el servidor, que puede ser nuestra computadora, corremos una aplicación que nos permita retransmitir los datos recibidos por TCP a un Puerto COM Virtual. Por último abrimos GoogleEarth y seleccionamos ese puerto virtual como entrada de datos. De esta forma se puede ver en tiempo real, la posición y la velocidad de la placa en un mapa de GoogleEarth.

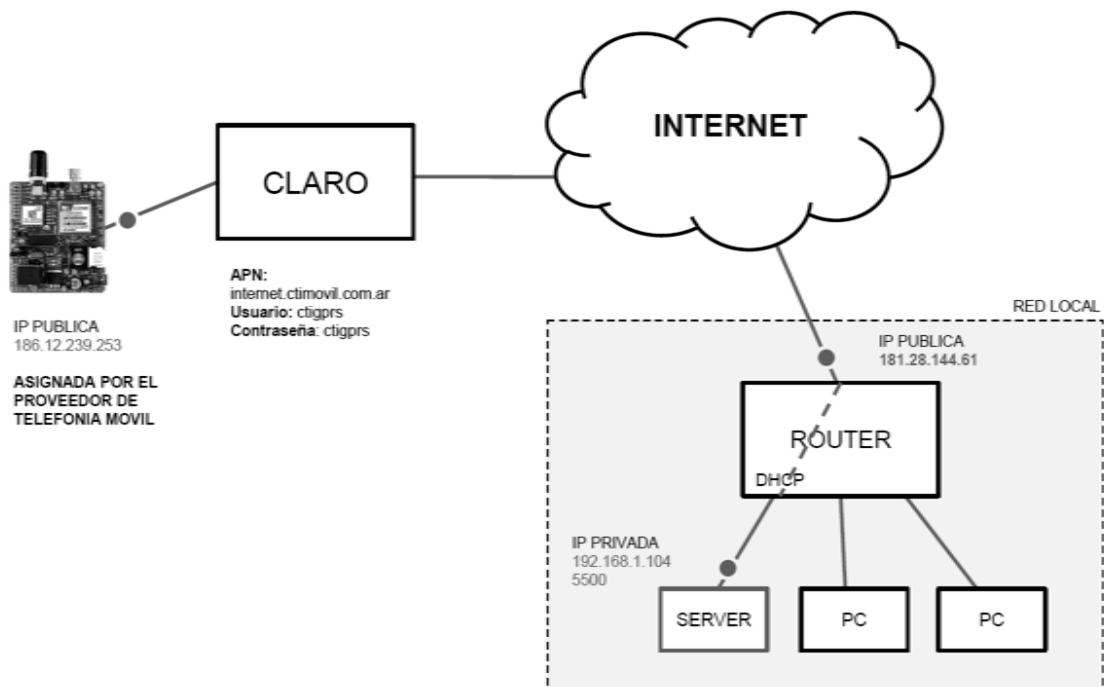


**También es posible almacenar la trama completa en la memoria de programa del PIC18F26J50 y luego convertir esa información a KML en la PC para abrirla con GoogleEarth o GoogleMaps.**

## EJEMPLOS Y APLICACIONES

### Comunicación por GPRS

En este esquema se representa el proceso de conexión. En primer lugar debemos obtener una dirección IP del proveedor de telefonía celular, para esto tenemos que logearnos en el servidor con usuario y clave según los datos del proveedor.



*Una vez obtenida la IP pública estamos en condiciones de consultar o enviar datos por GPRS. En el DVD hay un programa que nos permite crear un Servidor TCP en una computadora y de esa forma recibir la información del módulo.*

## EJEMPLOS Y APLICACIONES

### Comunicación por GPRS

Comandos para establecer la comunicación como cliente (la placa debe reportar a un servidor):

```
AT+CIPMODE=1 //GPRS EN MODO TRANSPARENTE
OK
AT+CGATT=1//ACTIVO EL GPRS
OK
AT+CSTT="internet.ctimovil.com.ar","ctigprs","ctigprs"//USUARIO Y CLAVE DEL
SERVIDOR GPRS
OK
AT+CIICR //ACTIVA CONEXIÓN GPRS
OK
AT+CIFSR //DIRECCIÓN IP LOCAL
IP
AT+CIPSTART="TCP","181.28.144.61","5500"
OK
CONNECT
```

### CONFIGURAR IP Y PUERTO SEGÚN SERVIDOR REMOTO

+++ Para regresar a modo comando.

ATO Para regresar a modo datos transparente.

*Puede encontrar un ejemplo para configurar el módulo SIM908 como servidor en el DVD del libro.*



# **APÉNDICE**

# **REGULACIONES**



## **REGULACIONES Y BUENAS PRÁCTICAS**

El diseño no termina en el hardware y el software sino que debe contemplar normas internacionales y buenas prácticas.

Al momento de considerar exportar un producto electrónico debemos tener en cuenta una serie de normas internacionales. Estas normas tienen como objetivo reducir el uso de sustancias peligrosas y tener un control sobre la emisión electromagnética.

### **Directiva ROHS:**

La directiva 2002/95/CE de Restricción de ciertas Sustancias Peligrosas en aparatos eléctricos y electrónicos, (**RoHS** del inglés "Restriction of Hazardous Substances"), fue adoptada en febrero de 2003 por la Unión Europea. Esta norma obliga a reducir el uso de las siguientes sustancias:

- Plomo
- Mercurio
- Cadmio
- Cromo VI (También conocido como cromo hexavalente)
- PBB
- PBDE

Los microcontroladores PIC son libres de plomo, adicionalmente se utilizó estaño libre de plomo en el proceso de soldado. Este estaño se funde a una temperatura de 420 grados centígrados. Muy por encima del punto de fusión del estaño convencional.

### **FCC-ID:**

La FCC es la Comisión Federal de Comunicaciones de los Estados Unidos. Uno de sus objetivos es controlar y certificar los dispositivos electrónicos que hagan uso del espectro radioeléctrico. En este caso la placa TrackMe, utiliza la red pública de telefonía móvil en las bandas 850, 900, 1800 y 1900, con lo cual el módulo de comunicaciones utilizado debe estar previamente certificado por la FCC. El fabricante del módulo GSM es la empresa China SIMCOM, con lo cual en la planilla de exportación de Fedex debemos aclarar que el módulo utilizado tiene su correspondiente FCC-ID y que no ha sido alterado su funcionamiento. De otra forma el producto no es aceptado en la aduana de Estados Unidos.

Algo similar ocurre a nivel local, el órgano contralor es la CNC (Comisión Nacional de Comunicaciones), en este caso el trámite de registro lo debe hacer quien importe el módulo al país.

## DISEÑO DEL PACKAGING

Si bien el packaging parece un dato menor e irrelevante para este tipo de productos, es fundamental si se pretende insertar el desarrollo en el mercado global. Como premisa el packaging debe proteger al producto en su interior. En este caso la protección no sólo es mecánica sino que también es estática.

### Protección estática:

Todas las placas se envían dentro de una bolsa antiestática y anti humedad para evitar daños eléctricos durante la manipulación al sacarlas de la caja. Además, por norma, estas bolsas llevan una etiqueta nomenclada indicando que el producto en su interior puede resultar dañado por estática.



Caja actual de mcelectronics.

### **Protección mecánica:**

La caja de cartón debe ser lo más resistente posible. En este caso se pensó el tamaño de la caja para que entre un número entero de ellas en un contenedor de FedEx. De esta forma las cajas no se mueven en su interior. La caja, además, debe indicar su contenido y en el caso de ser un producto que emita radio frecuencia, como en este caso, el correspondiente FCC ID para evitar demoras en la aduana.

### **Diseño eficiente y racional:**

Desde el punto de vista de la eficiencia está bien visto que el packaging sea lo más pequeño posible. Si bien una caja grande y colorida llama la atención en una góndola, este tipo de productos no se esperan encontrar en un supermercado. Con lo cual cuanto más pequeña sea la caja, menos se va a mover el contenido en su interior y más cajas van a caber un contenedor.



**Contenedores de FedEx de 10 y 25 Kg para exportación.**



# **WI-FI HECHO SIMPLE**



## WI-FI

Una conexión WIFI, es un mecanismo que permite el vínculo de dispositivos electrónicos en forma inalámbrica. Los dispositivos con Wi-Fi, tales como: una PC, una consola de videojuegos, un smartphone o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso (AP) tiene un alcance de unos 20 metros en interiores, y al aire libre una distancia mayor, casi 100m. Si se superponen los AP pueden cubrir grandes áreas.

Wi-Fi es una marca de la Wi-Fi Alliance (anteriormente la WECA: *Wireless Ethernet Compatibility Alliance*), organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados a redes inalámbricas de área local.

## HISTORIA

Esta tecnología surgió de la necesidad de establecer un mecanismo de conexión inalámbrica que fuese compatible entre los distintos dispositivos. Buscando esa compatibilidad las empresas 3com, Airones, Itersil, Lucent Technologies, Nokia y Symbol Technologies se reunieron para crear la Wireless Ethernet Compatibility Alliance, o WECA, actualmente llamada Wi-Fi Alliance. El objetivo de la misma fue designar una marca que permitiese fomentar más fácilmente la tecnología inalámbrica y asegurar la compatibilidad de equipos.

En abril del 2000 WECA certifica la interoperabilidad de equipos según la norma IEEE 802.11b, bajo la marca Wi-Fi. Esto quiere decir que el usuario tiene la garantía de que todos los equipos que tengan el sello Wi-Fi pueden trabajar juntos sin problemas, independientemente del fabricante de cada uno de ellos.

En el año 2002 la asociación WECA estaba formada ya por casi 150 miembros en su totalidad. La familia de estándares 802.11 ha ido naturalmente evolucionando desde su creación, mejorando el rango y velocidad de la transferencia de información, entre otras cosas.

La norma IEEE 802.11 fue diseñada para sustituir el equivalente a las capas físicas y MAC de la norma 802.3. Esto quiere decir que en lo único que se diferencia una red Wi-Fi de una red Ethernet es en cómo se transmiten las tramas o paquetes de datos; el resto es idéntico. Por lo tanto, una red local inalámbrica 802.11 es completamente compatible con todos los servicios de las redes locales (LAN) de cable 802.3 (Ethernet).

### Aplicaciones inalámbricas

- Sensores de humo y de CO
- Mediciones
- Apertura de garage
- Accesorios para Smartphone y tablets
- Monitoreo de la salud
- Juegos y entretenimiento

Hay cosas que debemos saber sobre WIFI, por ejemplo los tipos de red:

- 802.11 a/b/g/n
- Formas de proveer un dispositivo cliente
- Certificación
- Seguridad y autenticación
- Servicios y stack TCP/IP

### Tipos de red soportadas

Infraestructura:

Los nodos clientes se comunican vía un punto de acceso. La conexión más común es cuando conectamos nuestra PC a la RED del hogar.

Adhoc/Wi-Fi® Directa:

Este tipo de conexión permite un vínculo punto a punto. Esta es una gran ventaja frente a otros dispositivos.

SoftAP/LimitedAP:

El módulo puede actuar como un AP. El módulo es el coordinador de la red independientemente de la plataforma en donde está instalado

Los protocolos 802.11 a/b/g/n poseen distintas características como lo podemos visualizar en el siguiente esquema:

802.11 Protocol	Frequency	Modulation	Bandwidth	Data rates (Mb/s)	# MIMO streams	Comments
a	5GHz	OFDM	20 MHz	6, 9, 12, 18, 24, 36, 48, 54	1	High freq. reduces effective range
b	2.4GHz	DSSS	20 MHz	1, 2, 5.5, 11	1	Many IT departments are turning off "b" access points
g	2.4GHz	OFDM & DSSS	20 MHz	6, 9, 12, 18, 24, 36, 48, 54	1	Only universal module scheme. Access points auto-adjust rate to minimize packet error rate
n	2.4GHz & 5GHz	OFDM	20 MHz & 40MHz	7.2, 14.4, 21.7, 28.9, 43.3, 57.8, 65, 72.2 (per stream)	4	Must implement MIMO and 40 MHz bandwidth to get max. data rates (600Mb/s)

802.11a es utilizada para aislar redes y evitar la saturación del espectro de 2.4GHz

- Por ejemplo: hospitales y registros de pacientes

802.11g tienen un uso eficiente de energía y es totalmente compatible con redes 802.11n.

- Misma técnica de modulación

802.11n Es útil donde se requiere un alto rendimiento de transferencia de datos en aplicaciones como video de alta definición movimiento (Mbytes de datos vídeo y datos). Se aplican técnicas de Multiple Input Multiple Output (MIMO), para lograr tasas más elevadas de transferencia de datos. La mayoría de las aplicaciones cliente del dispositivo no necesita 802.11n

## SEGURIDAD Y AUTENTICACIÓN

### WEP: Wired Equivalent Privacy

Este tipo de seguridad es considerada obsoleta, estuvo vigente desde 1999 al 2003, pero desde 2008 fue prohibida por "Payment Card Industry Security Standards Council"

### WPAv1: Wi-Fi Protected Access (v1)

Es una versión recortada de la 802.11i, posee el mismo hardware que WEP, pero utiliza una encriptación extremo a extremo del tipo TKIP, las palabras claves que utiliza son mas largas y complejas (hexadecimal de 8 a 64 valores). Este método no es recomendado pero posee una seguridad razonable.

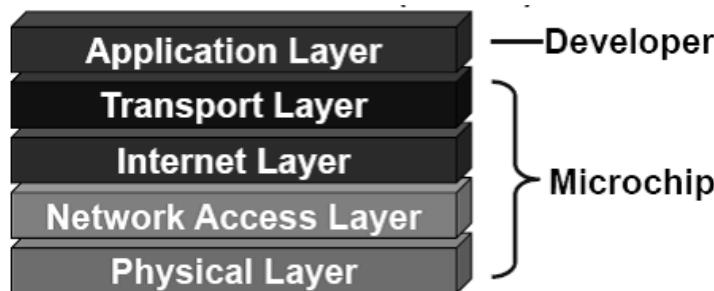
### WPAv2: Wi-Fi Protected Access (v2)

Se requiere una actualización del hardware y utiliza un algoritmo de 256bits lo cual la hace muy segura.

### WPA/WPA2 Enterprise:

Este método de seguridad adiciona un nivel de seguridad corporativo y se caracteriza por su compleja implementación. Todo esto hace que sea considerada muy segura.

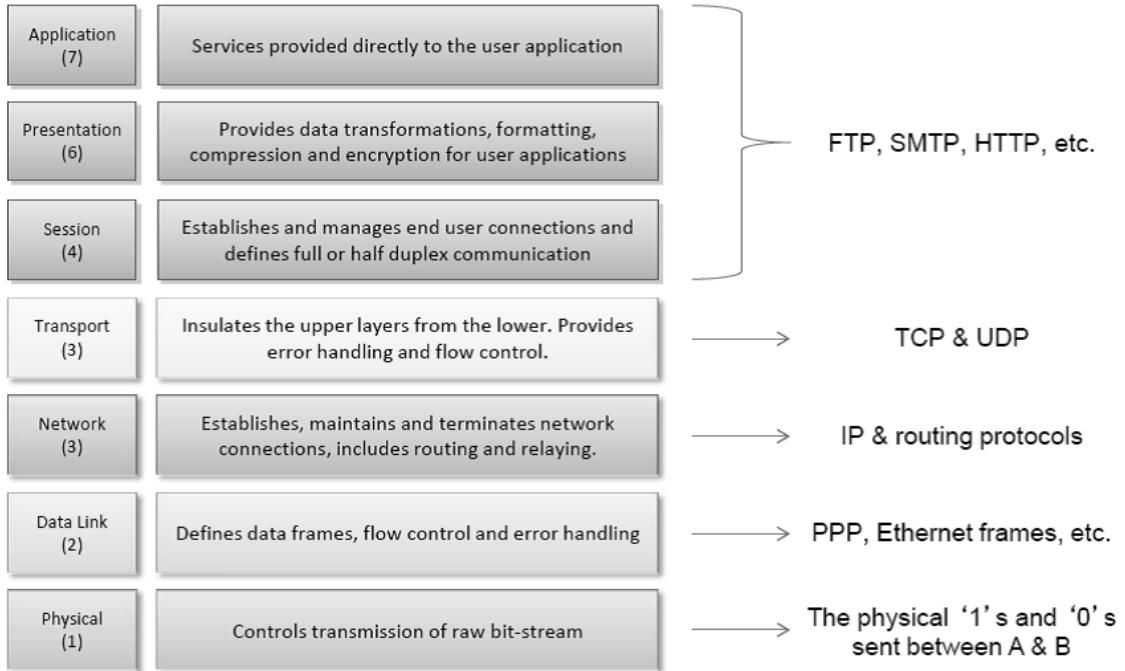
### Stack del protocolo de internet (TCP/IP)



Soporta múltiples sockets y transportes, multi interfase para un funcionamiento simultaneo de red cableada y WIFI.

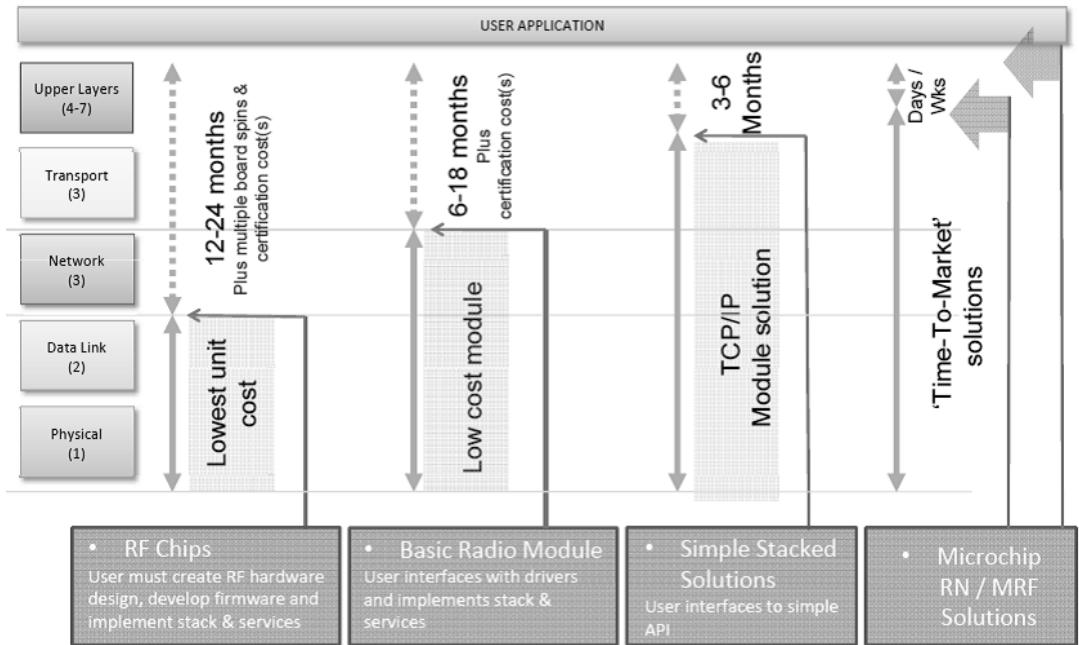
## Servicio y stack de WI-FI®

Cualquier solución WI-FI deberá tener la mayoría de las capas del stack WI-FI, las diferentes soluciones de los vendedores poseen algunas o todas las capas que se muestra a continuación.



## Inversión: Tiempo Vs. Dinero – Que elegir?

Esta ecuación debemos tenerla en cuenta al momento de encarar nuestro proyecto, podemos apreciar en el grafico siguiente los costos y las capas involucradas en cada implementación. Vemos que cuanto menor es el costo y la capa, mayor será el tiempo de desarrollo de nuestra aplicación por lo que nos demorara si deseamos encarar otro proyecto.



Microchip posee varios productos disponibles para los desarrolladores, la familia MRF y la RN son ejemplo de ello

Pero cuál de estos productos cubre nuestras necesidades y nos permite un desarrollo simple y de bajo costo. Analicemos las características más importantes de estas familias.

## FAMILIA MRF

En esta familia, el stack deberá ser cargado y ejecutado por el microcontrolador así como la aplicación. Esta es una gran carga que debe ser considerada al momento de dimensionar el microcontrolador de nuestro proyecto. El stack TCP/IP se encuentra disponible en el sitio web de microchip y viene preparado para alguno de los microcontroladores fabricados por esta firma. El modulo WI-FI soporta protocolos 802.11 b & b/g, posee un bajo consumo y un gran ancho de banda. El ambiente de desarrollo de microchip provee servicios IP complejos para nuestro desarrollo. La conexión entre el microcontrolador y el modulo se realizará por medio del puerto SPI.

## Familia RN

En esta familia, el stack se encuentra en el modulo el cual soporta 802.11 b/g. Como característica, este modulo posee un bajo consumo y una conexión muy rápida (<100ms). Al estar el stack en el modulo se desliga al microcontrolador de dicha tarea, con lo que se consigue que pueda funcionar con cualquier tipo de microcontrolador del mercado (debe poseer USART).

Al no tener stack, no son necesarias librerías especiales para el microcontrolador, con la interpretación de simples comandos, uno puede controlar todas las funciones, desde simples envíos de datos hasta sitios web, mails, etc.

Dependiendo desde qué punto uno desarrolle su proyecto, se deberá preguntar:

Está utilizando un PIC?

Conoce WI-FI?

Conoce TCP-IP?

Puede desarrollar implementaciones en RF?

Desea una solución a medida?

## MODULO DE RADIO MRF24WXXX

Este modulo posee un gran desempeño para los clientes de PICs, brinda servicios WI-FI para clientes desarrollados en las librerías de microchip. Los clientes pueden adquirir gran experiencia en desarrollos con este modulo, es de bajo consumo y gran ancho de banda. Ver capítulo *Ethernet y WI-FI embebido con PIC para aplicaciones completas con este módulo.*



Se utiliza con stack de microchip para PIC 8,16,32 bit, también permite compatibilidad para aquellos que deseen pasar de norma B a G.

Los servicios que dispone este módulo son:

Wi-Fi Directo / WPS / SoftAP / Server HTTP / SSL / IPv6, ect.

## MODULO DE RADIO RNXXX

Este modulo nos ofrece una solución en un solo chip, esto trae como beneficio la posibilidad de realizar desarrollos en corto plazo y así colocarlos en el Mercado fácilmente. Posee aplicaciones para la transferencia de datos, ultra bajo consumo (4uA sleep, 30mA TX).



Es ideal cuando se requiere un prototipo en corto tiempo, cuando uno no sabe sobre TCP/IP o WI-FI, cuando uno no conoce el stack de los MCU y que el mismo se encuentra embebido en el modulo, cuando uno utiliza MCU de terceros, cuando solo se desea agregar WI-FI a una aplicación desarrollada, para todo esto, podemos utilizar los servicios que nos brinda este modulo como ser:

- Cliente FTP
- TCP
- UDP
- WPS
- Soft AP
- DHCP
- Cliente DNS
- Telnet
- Cliente HTTP

Debemos hacer notar que también trae pines I/O para el usuario, conversores AD de 14 bits, timers, etc

Hay que destacar que este modulo posee una versión de firmware para WebScan™, o sea RTLS (Real-Time Location System).

Existe un firmware especial basado en el firmware standard de WIFLY, el cual esta diseñado para funcionar en todos los módulos Wi-Fi™ como el RN-171 y RN-131, este agrega la capacidad de un Sistema de Localización de Tiempo (RTLS). Este firmware esta diseñado específicamente para implementaciones de tracking donde se busca implementaciones de bajo costo y simplicidad.

El firmware esta armado para trabajar con tags, cuando el tag se despierta, webscan mide el nivel de señal RSSI de los puntos de acceso de la infraestructura local.

Los valores calculados de RSSI son devueltos al servidor activo en paquetes TCP. Utilizando simples cálculos de triangulación, un software de terceros puede fácilmente ubicar la posición con una precisión apreciable.

Los servidores utilizan esta información para mostrar la posición con registros de trayectorias o sistemas de mapeo con la capacidad de mostrar sobre una imagen la posición.

Los Tags pueden ser despertados utilizando varios mecanismos, incluyendo temporizadores, entradas de sensores y puntos de sensado de RFID

La utilización del modo dormido del modulo nos permite aumentar la autonomía de la batería así como su vida útil.

#### **Ejemplo:**

En este ejemplo, se monitorea la posición de la carga mediante el tag (etiqueta) colocado en el pallet, la carga llega a la bahía de entrada. Un operario con un elevador lo retira, en la bahía de carga una antena capta el modulo y envía datos grabados antes de su llegada al almacén, tales como camiones frigoríficos, temperaturas tomadas a intervalos de tiempo representada por la activación de los temporizadores, etc. Cuando la carga se desplaza, el modulo mide la señal de RSSI y la transmite a los servidores de la compañía por WIFI.

Vamos a analizar con más detenimiento el modulo de la familia RNXXX. Este modulo nos permite realizar una conexión a una red Ethernet por medio de WIFI utilizando el puerto serie de un microcontrolador e implementarlo por medio de comandos AT.

El modulo de RNXXX es un dispositivo standalone con conexión inalámbrica completa, el dispositivo tiene incorporado una pila TCP/IP y las aplicaciones, la configuración de hardware es muy simple, requiere sólo cuatro patas (alimentación, TX, RX, y masa).

Una vez que haya realizado la configuración inicial, el dispositivo automáticamente accede a una red wifi y envía/recibe datos en serie.

## Características del modulo

- certificados Wifi 2,4 GHz IEEE 802.11 b/g transceptor
- Ultra baja potencia de funcionamiento
- Administrador inteligente de energía incorporado con despierte programable
- Acepta 3,3v de alimentación o 2 a 3v a batería cuando se utiliza el regulador Boost.
- Consumos: 4 uA sleep, 35 mA Rx, 210 mW Tx at 18 dBm (Potencia de TX no configurable)
- Puede traer antena interna o conector para una externa
- Memoria flash de 9Mbit y 128Kbyte de RAM, 2Kbyte de ROM.
- 14 Pines de entrada salida de uso general para el RN171 y 10 pines para el RN131
- 8 entradas analógicas con conversores de 14bits y rango de 0 a 1,2v.
- RTCC para despierte

### Soporte de RED:

- Soporta Access Point (AP), ad hoc, y los modos de red de infraestructura.
- Botón para modos WPS de fácil configuración
- Stack TCP/IP incorporado
- Posibilidad de actualización del firmware por aire (FTP)
- Autenticación segura de WIFI vía WEP, WPA-PSK (TKIP), y WPA2-PSK (AES)
- Configuración por medio de comandos ASCII vía UART o WIFI
- Soporta: DHCP cliente, DNS cliente, ARP, ping ICMP, FTP cliente, TELNET, HTTP, UDP, y TCP

El módulo RNXXX tiene dos modos de funcionamiento: modo de datos y modo de comando. En la modalidad de datos, el módulo puede aceptar conexiones entrantes o iniciar las conexiones salientes. Para configurar los parámetros y/o ver la configuración actual, debe poner el módulo en el modo de mando (también llamado modo de configuración).

## Modo comando

De forma predeterminada, después de encender, el módulo está en modo de datos. Enviando la secuencia \$\$\$ hace que el módulo pase al modo de comandos. Hay que enviar \$\$\$ junto y con rapidez sin caracteres adicionales antes o después.

No se debe enviar un retorno de carro (CR) o un salto de línea después de \$\$\$ para entrar en el modo de comandos. El módulo responderá con CMD para indicar que está en modo de comandos. Una vez en el modo de comandos, se puede configurar el dispositivo con simples comandos ASCII; cada comando termina con un retorno de carro.

La mayoría de los comandos devuelven OK, los inválidos devuelven ERR. Para salir modo de comandos, enviar exit<CR>. El módulo responde con EXIT, lo que indica que se ha salido modo comando y se entró en modo datos.

Hay 250-ms de tiempo antes y después de la secuencia de escape para \$\$. Si no se entra en esa ventana, el modulo tratará todo como datos sin entrar en modo comando.

Se pueden visualizar diversos parámetros, tales como el SSID, el canal, la dirección IP, el puerto serie, etc. Todo esto es configurable desde el modo comando.

Es posible enviar comandos al módulo por la UART o remotamente a través de telnet. Cuando se utiliza la interfaz UART, la configuración debe coincidir con los ajustes guardados. El valor predeterminado es 9.600 baudios, 8 bits, sin paridad, 1 bit de parada y control de flujo de hardware.

Se puede entrar en el modo de comandos en la interfaz UART en cualquier momento independientemente de la conexión TCP activas.

Cuando el módulo WiFly enciende, se lleva a cabo el intento de auto-asociarse con el punto de acceso almacenado en sus parámetros de configuración, siempre que la característica de asociación automática esté activada. En la versión de firmware 4.0 o mayor, la característica de unión automática está desactivada de forma predeterminada. Para poder habilitarla se utiliza el comando ASCII **set wlan join 1**

Se puede deshabilitar la auto asociación utilizando el comando **set wlan join 0**. Este comando evita que el modulo trate de conectarse a una red que no existe y consume energía de la batería.

## Configuración remota Utilizando el modo Ad Hoc

Si utilizamos el modo ad hoc de configuración eliminamos la necesidad de que el modulo tenga que ser asociado a un punto de acceso en una RED. En el modo ad hoc , el modulo crea su propia red a la cual uno puede conectarse con una PC como se conectaría en una red WI-FI común. Para habilitar el modo ad hoc por hardware, se debe poner el pin GPIO9 en un nivel alto, 3.3 V al inicio.

Luego del arranque, con GPIO9 en nivel alto, el modulo WiFi creara una red ad hoc con la siguiente configuración:

SSID: WiFi-GSX-XX, donde *XX* son los dos bytes de la dirección MAC

Canal: 1

DHCP: desactivado

Direccion IP: 169.254.1.1

Mascara: 255.255.0.0

Desde su computadora, conéctese a la red WiFi-GSX-XX. Esta es una red abierta que no requiere clave. En la actualidad, los modulos WiFi solo soportan modo abierto par a las redes ad-hoc.

Luego de algunos minutos, Windows asignara un IP y se conectara a la red.

Se puede verificar el IP ejecutando el comando **ipconfig**.

Si está conectado, se mostrara la dirección IP y la máscara.

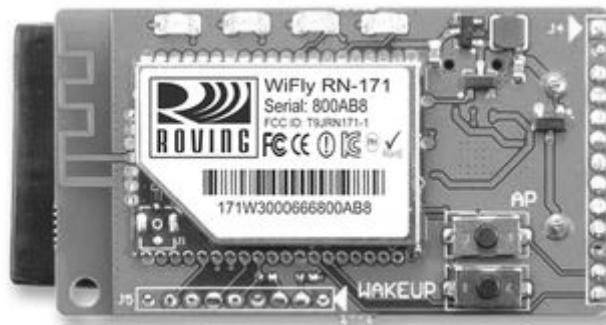
**Nota:** La dirección asignada por defecto es 169.254.x.y. Si su PC posee también una conexión por cable, ésta debe ser desactivada.

Una vez conectado, mediante el comando telnet es posible la configuración, el puerto por defecto es el 2000 (**telnet 169.254.1.1 2000**)

El modulo responderá con **\*HELLO\***. A partir de aquí es posible entrar en el modo comando utilizando la secuencia de escape **\$\$\$** y enviar los comandos de configuración deseados.

En versiones de firmware 2.28 y superiores, se puede deshabilitar la configuración remota para prevenir problemas mediante la utilización del bit 4 en el registro de modo TCP mediante el empleo del comando **set ip tcp-mode 0x10**

Comercialmente podemos adquirir un modulo de entrenamiento el cual posee dos pilas AAA, leds, pulsadores para poder realizar nuestras primeras practicas.



*Modulo WiFly RN-171 de Microchip*

Veamos puntualmente algunas funciones específicas paso a paso

### **Entrar en modo comando**

1. Desde la terminal del modo comando, hay que poner el modulo en modo comando tipeando **\$\$** en la ventana de la PC. Luego, con el comando **CMD** confirmamos que estamos en el modo comando.
2. Luego escribamos **show net** para mostrar la configuración actual de la red.

```
CMD
Show net
SSid=MiCasa
Chan=6
Assoc=OK
DHCP=OK
Time=FAIL
Links=1
<2.03>
```

Nota: Luego de que cada comando se complete, se verá el prompt que lucirá como <X.XX> donde X.XX indica la versión del firmware. Utilizando la función de búsqueda del modulo, se puede encontrar el nombre del (SSID) y canal de las redes disponibles.

Para iniciar el modo de escaneo de redes, deberemos estar en el modo comando y tipear scan, el modulo responderá con el prompt con una lista de redes WI-FI en el rango de alcance.

### Como asociarse a un Punto de Acceso

EL LED verde de la placa de experimentación WiFi GSX destellara si no se conecto a un punto de acceso. Si el punto de acceso al que se desea conectar posee una configuración abierta, se puede utilizar directamente el comando de unión para asociarse.

De la lista de escaneo se puede ver que la REv roving1 está abierta, si se escribe **join roving1** nos asociaremos a esa red.

Luego de reiniciarse el modulo se conectara a la red, adquirirá una dirección IP y el LED rojo se apagara. Si el LED verde destella lentamente es que está buscando IP.

También se puede indicar el numero de RED de la lista de escaneo utilizando el comando **join #1**. Si se conoce el nombre de la red a unirse, podemos configurar el nombre de SSID con el siguiente comando sin necesidad de utilizar el escaneo, **set wlan ssid access\_point\_name** y luego utilizamos el comando **join** sin argumentos.

Luego de configurar el modulo se deberá grabar la configuración para que la próxima vez, al iniciarse tome esta configuración. Para esto último utilizaremos el comando **save** y luego rebooteamos el modulo.

## **Envío de datos utilizando UDP**

UDP es un protocolo de conexión. Este protocolo posee como característica que no hay acuse de recibido del paquete del receptor. Esto hace que el protocolo UDP garantice que llega el paquete. Sin embargo, debido a su naturaleza sin conexión, UDP es ideal para aplicaciones que no toleran latencia excesiva pero pueden tolerar algunos errores en los datos. La transmisión de vídeo sería un buen ejemplo de aplicaciones UDP.

Para utilizar UDP con el modulo WiFi-GSX, se debe habilitar el protocolo utilizando el comando **set ip proto 1**, también es necesario configurar el IP del equipo remoto y el puerto local y remoto.

Los comandos para habilitar la transferencia de datos UDP serán:

### **Asociarse a una red**

```
set wlan ssid <string> // configure el nombre de la red  
set wlan phrase <string> // configure la clave para los modos WPA y WPA2
```

### **Configurar el protocolo y el puerto**

```
set ip proto 1 // habilita UDP como el protocolo  
set ip host <ip address> // configuro la dirección IP del equipo remoto  
set ip remote <port> // configuro el puerto remoto en el cual el host esta escuchando  
set ip local <port> // configure el Puerto por donde escucha el modulo  
save // graba la configuración  
reboot // reinicia el modulo para que los cambios tengan efecto
```

NOTA: Si intenta enviar datos físicamente escribiendo los caracteres en el teclado o si el microcontrolador no está enviando datos lo suficientemente rápido, el módulo WiFi enviará los paquetes con menos bytes de datos.

Para evitar este problema, defina el temporizador de limpieza a un valor superior. De forma predeterminada, se establece en 10 milisegundos.

Usted puede elegir deshabilitar el reenvío basado en temporizador de limpieza (deberá utilizar "set pers. tiempo 0 ") o si desea establecer un valor superior (por ejemplo, set pers. tiempo 2000).

Dado que UDP es un protocolo sin conexión, los datos comenzarán a fluir tan pronto como el módulo se reinicia. A diferencia TCP, no es necesario realizar una red "ABIERTA" para que se establezca la conexión. El modulo WiFly-GSX actúa como un puente, de modo que los datos ingresados por la UART se envían a través del link Wifi utilizando el protocolo UDP (en este caso) y los datos procedentes de la Wifi (mediante protocolo UDP en este caso) se enviarán a la UART.

### **Como hacer una conexión**

Para realizar una conexión, hay que abrir un socket y conectarse a una dirección IP. El telnet es la forma más simple, para probar esto. Desde el telnet escribamos open<dirección><puerto>, por ejemplo open 190.2.4.189 2000. Una vez abierto, los caracteres que ingresen por la UART, serán enviados y viceversa. Para realizar una conexión desde el modulo necesitaremos la dirección IP y puerto de nuestro servidor.

Un programa para realizar pruebas es el hércoles de la empresa HW group. Este programa abre un puerto IP y muestra en pantalla lo recibido y lo enviado.

También puede redireccionar el puerto e un COM de la PC.

Hay que recordar que para que esto funcione si tenemos un firewall o router se deben realizar configuraciones en este para que el tráfico llegue a nuestra PC.

Un ejemplo es definir nuestra PC como DMZ en el router o redireccionar los puertos.

### **Configurando conexiones automáticas**

A menudo se busca que al encender el modulo o en un despertador, este se conecte automáticamente a un servidor , envíe un dato y luego se desconecte. Todo esto se puede configurar para que ocurra automáticamente.

En el ejemplo que se muestra a continuación, se asume que el SSID de la red y la seguridad han sido configurados correctamente con anterioridad y la opción de autoconexión esta en 1(autojoin 1).

Esto también funciona en el modo ad hoc (autojoin 4), sin embargo hay una demora en la conexión a la red adhoc desde la computadora remota para garantizar una correcta unión a la red.

Una vez que el modulo se inicia o despierta, el temporizador de autoconexión, provoca que el modulo intente una conexión con la dirección IP y puerto configurados.

Mientras la conexión este abierta o fluyan datos, el temporizador no se decrementa. Una vez que los datos se detengan por 5 segundos, la conexión se cerrará, el temporizador sleep arrancará y se pondrá el modulo en sleep, finalmente el temporizador de despierte, comenzará el ciclo de nuevo. En el ejemplo podemos ver un caso de lo explicado anteriormente, hay que prestar atención a los dos temporizadores que se configuran como se el idle timer y el sleep timer.

**set ip host X.X.X.X** ( configura el IP de la PC remota )

**set ip remote\_port num** (configura el Puerto de la maquina remota)

**set sys autoconn 1** (conexión automática luego de encender o despierta)

**set com idle 5** (se desconecta luego de 5 segundos de no actividad de datos)

**set sys sleep 2** (duerma 2 seg antes de cerrar la conexión)

**set sys wake 60** (luego de 1 min de dormido se despierta)

**UART data TRIGGER mode.** Este modo realiza automáticamente conexiones TCP/HTTP

**set uart mode 2**

## Control de la conexión utilizando el PIO5 y PIO6

El pin PIO5 puede ser utilizado para controlar la conexión TCP. Una vez configurado el modulo y la función IO del pin, cuando esté en estado alto, se realizará la conexión almacenada, cuando esté el estado bajo se desconecta.

Con **set sys io 0x20** se configura el pin PIO5 para conectar y desconectar, es posible monitorear el estado de la conexión mediante el pin PIO6. Un estado alto indica una conexión abierta, un estado bajo indica que no hay conexión. Para habilitar las funciones del pin PIO6 se debe utilizar el siguiente comando.

**set sys io 0x40**

## Utilizando la configuración de DNS

El modulo WiFly contiene un cliente DNS, si la dirección IP del host no se especifica, por ejemplo si se configura en 0.0.0.0, significa que utilizamos el DNS. El modulo tratará de resolver automáticamente la dirección del host almacenada con el comando: **set dns name <string>** configura el nombre del host para conexiones TCP/IP.

Una vez que la dirección se resuelve, la conexión automática se establece.

Para ver la dirección de un host manualmente y probar que el cliente DNS se encuentra bien configurado, podemos utilizar el siguiente comando:

**Lookup <string>** string es el nombre del host.

### Enviar y recibir datos utilizando el servidor Web

El modulo WiFly puede ser configurado para descargar o enviar datos a un servidor Web.

Ejemplo: un usuario desea recuperar datos de un servidor web con el siguiente formato:

**http://www.servidorweb.com.ar/ob.php?obvar=WEATHER**

Configuración:

```
set dns name www.servidorweb.com.ar //nombre del servidor web
set ip address 0 // indicó que se utilice el DNS
set ip remote 80 // Puerto del servidor web
set com remote 0 // apaga el string remoto para que no interfiera
```

Para hacer la conexión con el modulo debemos escribir:

**open** o en una línea **open www.servidorweb.com.ar80**

El microcontrolador del usuario deberá enviar los datos:

**GET /ob.php?obvar=WEATHER \n\n**

Recordar que el \n es el fin de línea, en decimal es el 10 y en hexadecimal es el 0x0a, dos finales de línea seguidos son necesarios para que el servidor web sepa que la pagina esta completa.

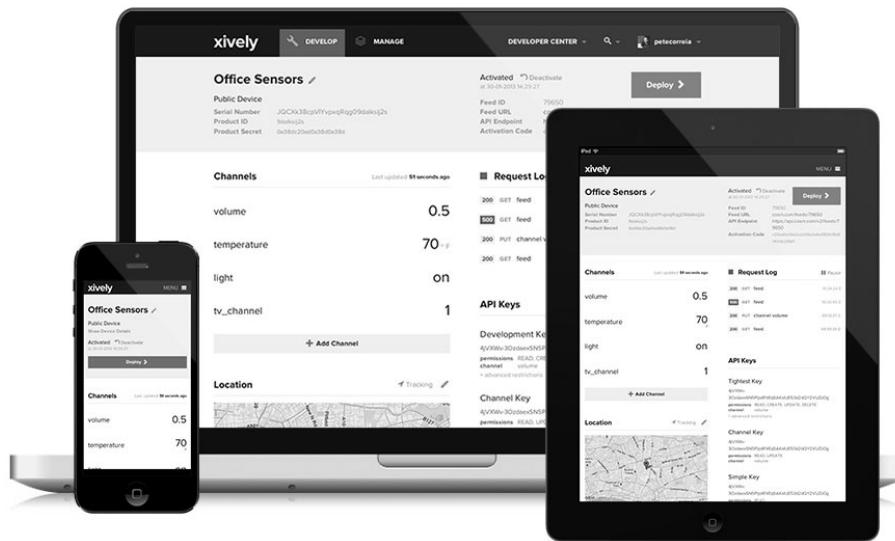
## ENLACE DE DATOS REMOTO

Muchas veces para subir información debemos configurar un servidor con una base de datos que nos permita el almacenamiento y luego realizar páginas que nos muestren los datos, existen en internet servicios pagos y gratuitos para practicar o poner en funcionamiento proyectos de este tipo.

Combinemos ahora un modulo RN171, un sistema operativo ANDROID y el servicio provisto por <https://xively.com>

### Paso 1: crear un usuario en Xively

Crear un usuario en Xively. Nos permite obtener un sitio para guardar datos en un servidor y visualizarlos después de cualquier acceso a Internet, para esto hay que crear el usuario en Xively.



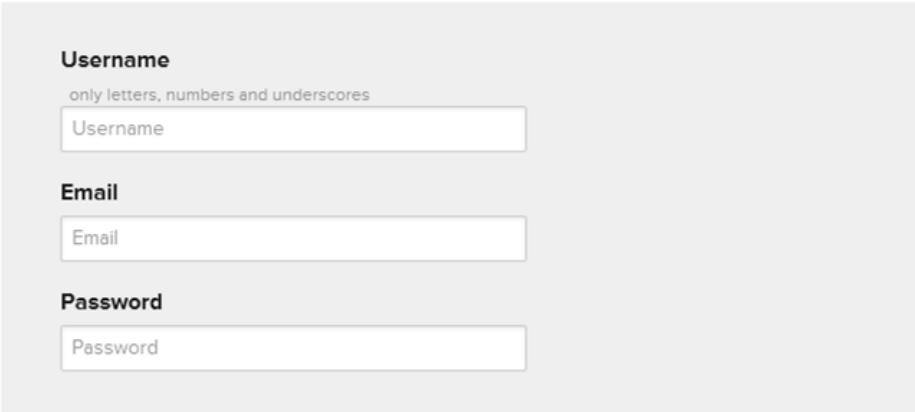
Después de haberlo generado, hay que elegir un password, un nombre de usuario y el número del mismo. Estos son los datos necesitaremos para acceder el "usuario". Hay que recordar que esto permite armar un sistema bidireccional. En la página web de la empresa hay gran

documentación de cómo realizar las implementaciones y aprovechar al máximo los servicios que brindan.

## Sign Up

For a free Developer Account

Looking for Commercial Service?



The screenshot shows a registration form with three input fields. The first field is labeled "Username" with the placeholder "only letters, numbers and underscores". The second field is labeled "Email" with the placeholder "Email". The third field is labeled "Password" with the placeholder "Password". Each field has a small explanatory text above it.

*Formulario de registro en xively*

### **Paso 2: Mediante la terminal GTK, conectar el modulo**

Debemos conseguir un circuito impreso o placa que nos permita montar el modulo. Se debe prestar especial cuidado a la alimentación del modulo ya que no funciona con 5V, el voltaje correcto es entre 2V y 3,3V así dos pilas de 1,5V funcionan perfecto.

Para comunicarse con el modulo RN171 hay que usar un programa terminal serial como el hyperterminal o similar.

Debemos implementar algún conversor serie (RS232) a niveles TTL, se podría utilizar el FT232RL. Cualquier convertidor es útil, un circuito sencillo es utilizar el PIC18F14K50, el firmware se baja de la web de microchip.

### Paso 3: configurar el RN171

Para conectarnos debemos configurar el hyperterminal en 9600N81.

\$\$\$ // entrar en modo comando LED parpadeando.

CMD // Resp:

scan // este comando retorna la redes de la zona y la dirección MAC

Resp:

<2.32>

SCAN:Found 1

Num SSID Ch RSSI Sec MAC Address Suites

1 AP MCE WPA2PSK d8:b3:77:14:20:04

set wlan ssid <string> // si el nombre lleva espacios al que reemplazar es por el caracter \$  
(AP\$MCE.....)

Resp:

AOK

<2.32>

set wlan auth 2 // 2 es por WPA2 como modo seguridad del AP.

Resp:

AOK

<2.32>

set wlan phrase <string> // password para WPA / WPA2 (10 números).

Resp:

AOK

<2.32>

set wlan join 1 // indico que la wlan se cargará automáticamente al encender.

Resp:

AOK

<2.32>

save // se guarda la configuración

Resp:

*Storing in config*

<2.32>

reboot  
Resp:  
*\*Reboot\*WiFly Ver 2.32, 02-13-2012 on RN-171*  
*MAC Addr=00:06:66:72:43:cd*  
*Auto-Assoc AP MCE chan=6 mode=WPA2 SCAN OK*  
*Joining AP MCE now..*  
*\*READY\**  
*Associated!*  
*DHCP: Start update data to*  
*DHCP in 455ms, lease=43100s*  
*IF=UP*  
*DHCP=ON*  
*IP=192.168.2.99:2000*  
*NM=255.255.255.0*  
*GW=192.168.2.1*  
Listen on 2000 // hemos hecho la conexión con el RN171.

exit  
Resp:  
*EXIT*

\$\$\$ // entramos otra vez en el modo de comando para configuración.  
Resp:  
*CMD*  
*set ip host 216.52.233.121 // ip api.com (Xively IP).*  
Resp:  
*AOK*  
*<2.32>*  
set ip remote 8081 // el puerto del usuario.  
Resp:  
*AOK*  
*<2.32>*  
set sys autoconn 1 // al arrancar se conecta  
Resp:  
*AOK*  
*<2.32>*  
save // guardar la configuración.  
Resp:  
*Storing in config*  
*<2.32>*  
exit  
Resp:  
*EXIT // al arrancar la próxima vez el RN171 se conectará a Xively.*

## CÓMO SUBIR LOS DATOS:

En este ejemplo cambio el valor de la variable temp a 30.

```
{  
  "method": "put",  
  "resource": "/feeds/XXXXXX", // XXXXX es el número de la variable.  
  "params": {},  
  "headers": {  
    "X-ApiKey": "el password personal optenido de Xively"  
  },  
  "body": {  
    "version": "1.0.0",  
    "datastreams": [  
      {  
        "id": "temp", // el nombre de la variable.  
        "current_value": "30"  
      }  
    ]  
  },  
  "token": "0x12355"  
}
```

## FTP

Estos módulos poseen la capacidad de un FTP cliente el cual viene en el firmware estándar. Esto permite al modulo obtener o enviar archivos al servidor, Antes se utilizaba este servicio únicamente para la actualización del firmware.

Ahora con la version2.22 o superior, es posible obtener o enviar archivos a un servidor FTP además de actualizar el modulo.

Configuración del modulo para una conexión a un servidor FTP

Debemos tener previamente configurado un servidor FTP, el modulo debe estar asociado a una red WIFI.

Para conectar el modulo a un servidor FTP es necesario realizar unos cambios en la configuración:

```
set ftp address <IP address> // se configure la dirección IP del servidor FTP  
set ftp dir <string> // configuración de la carpeta dentro del FTP, por defecto es publico  
set ftp user <string> //configure el nombre de usuario  
set ftp pass <string> //configura la clave del servidor FTP  
save  
reboot
```

### **Con estos comandos nos comentamos al servidor FTP**

Crear un archive en un servidor FTP

Una vez que el modulo se conectó al servidor FTP, podremos crear archivos en el servidor.

Para crear el archivo la secuencia de comando es:

```
ftp put <filename>
```

Este comando crea el archivo en el servidor FTP con el nombre , por defecto es OPEN, cuando aparece en la UART OPEN, se puede empezar a escribir datos en el archivo.

Hay dos formas de cerrar el archivo, una es enviando un string de cierre, por defecto es CLOS, el Segundo metodo para cerrar el archivo es utilizar el temporizador (set ftp timer <value>).

Una vez que uno deja de escribir en el archivo, el timer comienza a contar el tiempo en forma regresiva y luego cierra el archivo cuando llega a 0.

El temporizador es 1/8 del valor seteado, o sea, para 5 segundos el comando será:

```
set ftp timer 40
```

El comando de apertura y cierre será:

```
set comm open <string>
```

```
set comm close <string>
```

## **Leer un archivo desde un servidor FTP**

Para leer un archivo del servidor FTP, debemos utilizar los siguientes comandos:  
ftp get <filename>

Este comando enviará la sentencia OPEN por la UART y luego se comenzará a transmitir por la UART los datos del archivo. Una vez finalizado enviará la sentencia CLOSE, esto indica que la transmisión fue satisfactoria.

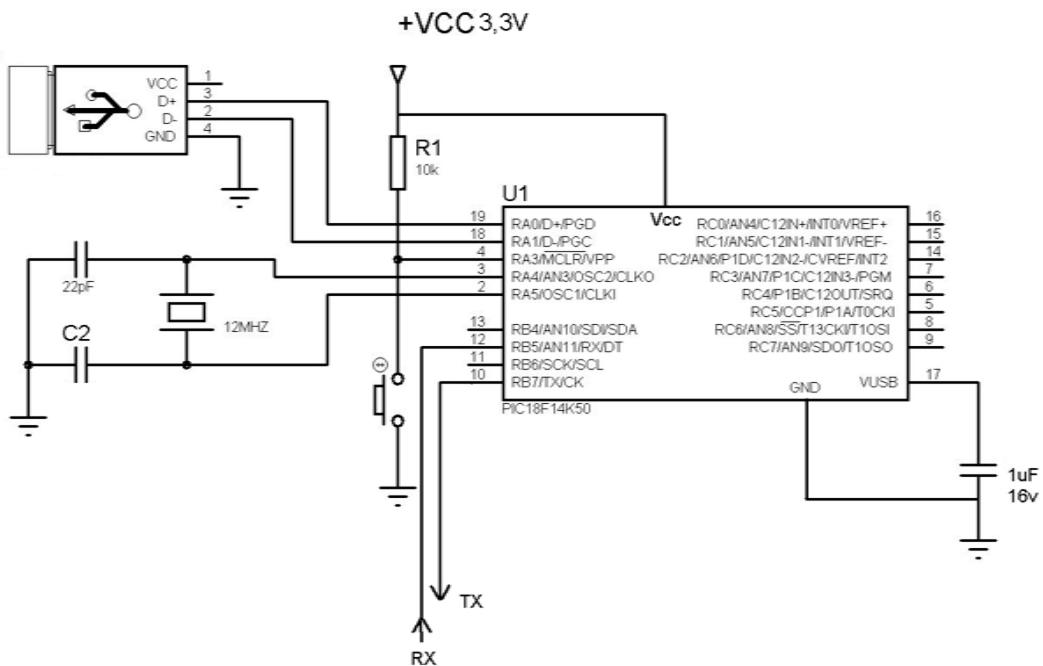
## **Actualizar el firmware**

Para actualizar el firmware del modulo WiFi, es necesario configurar el modulo para conectarse al servidor FTP de Roving Networks (configuración por defecto)

La configuración por defecto es:

FTP username = roving  
FTP password = Pass123  
FTP filename = wifly-GSX.img  
FTP directory = public (este parámetro no se puede modificar)  
FTP server = 208.109.78.34 (Roving servidor por defecto de actualización, puerto 21)

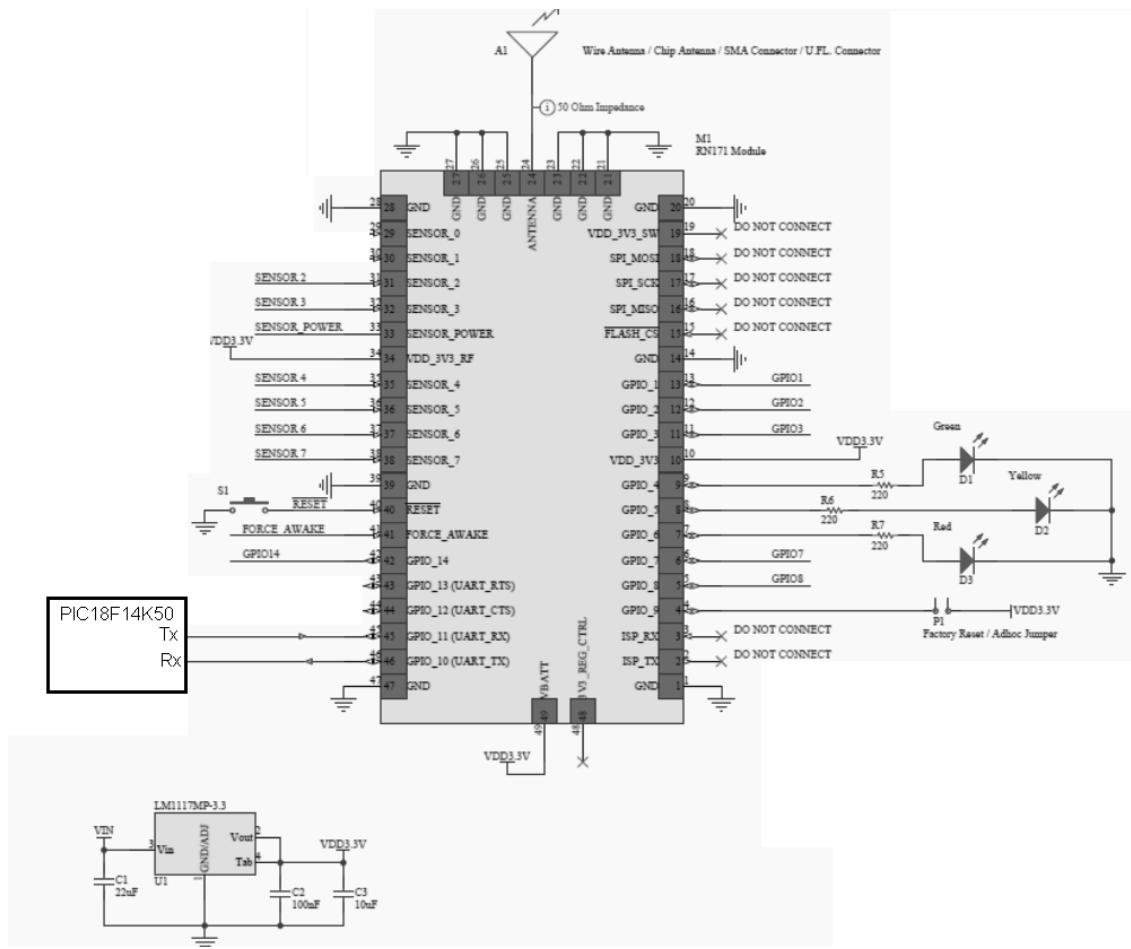
## CONEXIÓN DEL MODULO A UNA PC POR MEDIO DE UN MICROCONTROLADOR



*Esquema de conexión del microcontrolador. Se utilizan los pines TX y RX de la USART para controlar el modulo WiFi.*

Utilizaremos el PIC18F14K50 como puente entre el puerto USB de la PC y el puerto serie del módulo. En el circuito de la figura se muestra el circuito a implementar. Es importante que la fuente de alimentación sea de 3,3V ya que es la tolerada por el modulo.

El microcontrolador se vinculará al módulo mediante 3 pines, GND, TX y RX según el siguiente circuito.



*Esquema de conexión del modulo RN-171*

Una vez armado, deberemos programar el microcontrolador con el software que emula un COM Virtual en la PC. Esto nos permite controlar el modulo RN171 por medio del puerto USB. Se puede descargar desde la página de microchip en la siguiente dirección:

[http://www.microchip.com/stellent/idcplg?IdcService=SS\\_GET\\_PAGE&nodeId=1406&dDocName=en536385](http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en536385)

Bajo el nombre de Low Pin-Count USB Development Kit Project Labs. También se puede descargar el framework sobre USB y allí buscar dichas aplicaciones.

Al finalizar podremos controlar el modulo desde la PC y practicar los comando enunciados anteriormente. Una vez que el sistema este funcionando correctamente podemos prescindir de la computadora y grabar las instrucciones directamente en el PIC para crear un sistema autónomo.



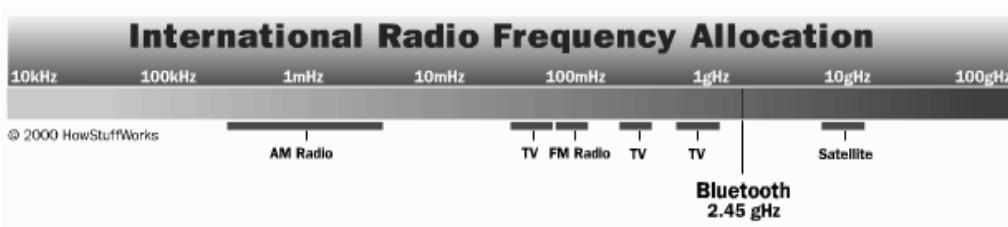
# **COMUNICACIONES BLUETOOTH**



## BLUETOOTH

Bluetooth es una tecnología inalámbrica de corto alcance que permite conectar computadoras, dispositivos de entrada, teléfonos móviles y dispositivos digitales portátiles entre sí sin cables. Cada dispositivo Bluetooth está equipado con un transceiver que transmite y recibe en una frecuencia de 2.4 Ghz, la cual está disponible en todo el mundo y no requiere de ninguna licencia.

Además de los canales de datos, están disponibles 3 canales de voz de 64 kbs. Cada dispositivo tiene una dirección única de 48 bits basada en el estándar de la IEEE 802.11 para redes de área local inalámbricas, que le permite formar temporalmente parte de una red. Las conexiones son uno a uno (punto a punto), con un alcance máximo de 10 metros, aunque mediante el uso de repetidores se puede lograr un alcance de hasta 100 metros con algo de distorsión.



El esquema de salto en frecuencia, permite a los dispositivos comunicarse incluso en áreas donde existe gran interferencia electromagnética, y además por cuestiones de seguridad se utilizan mecanismos de encriptación de 64 bits y autenticación para controlar la conexión y evitar que dispositivos no autorizados puedan acceder a los datos o modificarlos. El manejo del control de acceso y encriptación se hace a nivel de aplicación.

Los dispositivos pueden comunicarse entre sí e intercambiar datos de una forma transparente para el usuario. Hasta 8 usuarios pueden formar parte de una red de este tipo y pueden coexistir hasta 10 redes en la misma área de cobertura.

En cuanto a la interferencia que pudieran causar otros dispositivos, se debe tener cuidado con los que operan en la misma banda.

### Enlace de datos Bluetooth® con MCUs PIC®

Bluetooth..., Porque se lo debe considerar en su diseño?

- Sin costo: Banda no licenciada
- Millones de smartphones
- Solo se necesita diseñar un lado del link
- Interfaces de usuarios disponibles en teléfonos
- Desde su aparición la interface bluetooth a crecido notoriamente año a año

## EL NOMBRE BLUETOOTH

El rey Harald Gormsson fue el rey de Dinamarca y Noruega. Él ordenó muchos proyectos de obras públicas que unido a los dos países. A él le encantaba comer arándanos y tenía dientes manchados. Por lo tanto, Bluetooth (dientes azules).

Esta es una tecnología de intercambio de datos inalámbricos operando en la banda de 2.4 a 2.48 Ghz con 79 canales, Frequency Hopping Spread Spectrum (FHSS), fue creada por Ericsson en 1994 para reemplazar a la RS-232 cableada y así brindar un medio de intercambio de datos más versátil.

Actualmente es gestionado por un grupo de interés especial de Bluetooth (SIG). Para poder utilizar el logo se debe ser miembro del grupo SIG (sin costo).

La arquitectura por lo general consta en un maestro hablando con un esclavo permitiendo la transferencia de datos entre maestro y esclavo. El proceso de conexión consta en: descubrir, apareamiento, autenticación (opcional).

Los Rango/Clases nos muestran las potencias de cada clase y su alcance, existen módulos, de la firma Rowing Networks que cubre las clases existentes y son muy sencillos de utilizar.

Clase	Máxima Potencia		Rango (m)	RN Module
	(mW)	(dBm)		
Clase 1	100	20	~100	RN-41
Clase 2	2.5	4	~30	RN-42
Clase 3	1	0	~10	RN-42

Otro parámetro es la Versión/Estándar la cual nos definirá la velocidad de comunicación.

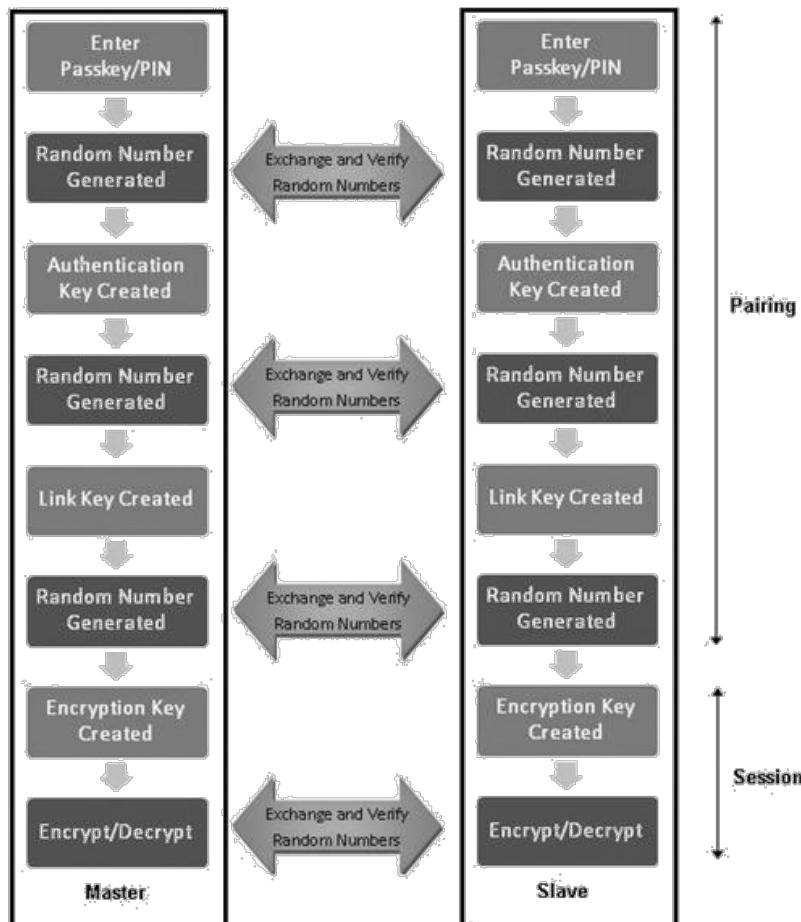
Versión	Velocidad	Vel. Max.
Versión 1.2	1 Mbit/s	0.7 Mbit/s
Versión 2.1 + EDR	3 Mbit/s	2.1 Mbit/s

Analicemos el proceso de conexión; Bluetooth® emplea una encriptación rápida y segura de 128 bits, (SAFER), esta se desarrolla en un proceso de tres pasos:

1-Describir el dispositivo

2-Asociación

3-Autenticación es opcional y es requerido si uno de los dos dispositivos demanda autenticación



En este grafico podemos observar los pasos de conexión entre dos dispositivos utilizando este tipo de conexión.

## Perfil Bluetooth®: dispositivo de interfaz humana (HID)

Típicamente utilizada en aplicaciones de teclados y mouse

- La computadora Host, carga el driver HID cuando la asociación es exitosa
- Omnipresente, interfaz estándar en todas las plataformas por lo que no se necesita la instalación de drivers adicionales

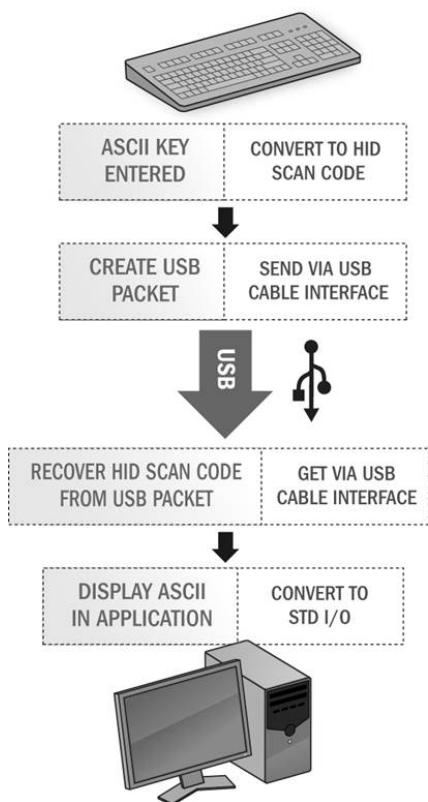
### Conexión de dirección simple

- Para requerimientos bidireccionales, se utiliza el puerto SPP

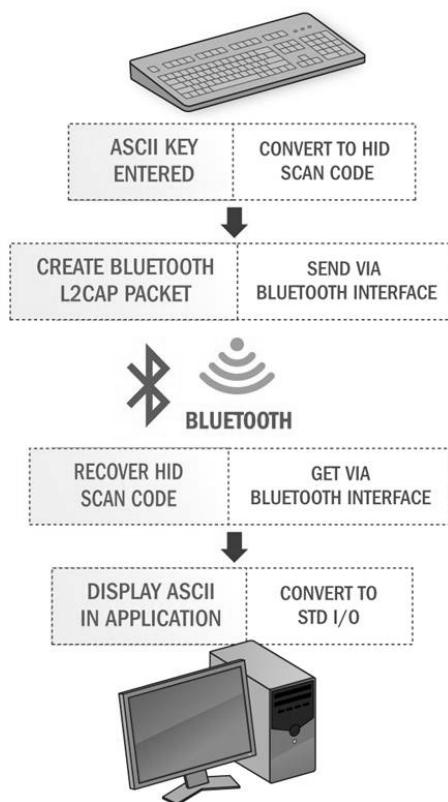
### El modulo RN posee muchas ventajas adicionales

- Traducción ASCII
- Reportes de consumo
- Control de teclado virtual Apple

#### TYPICAL HID CABLE ENVIRONMENT



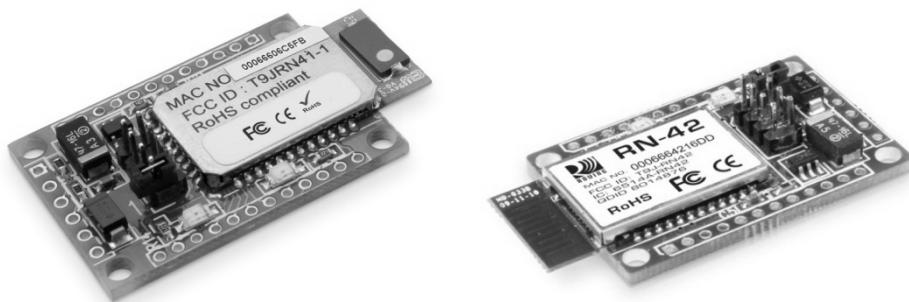
#### HID WIRELESS ENVIRONMENT



## DISPOSITIVOS DE MICROCHIP

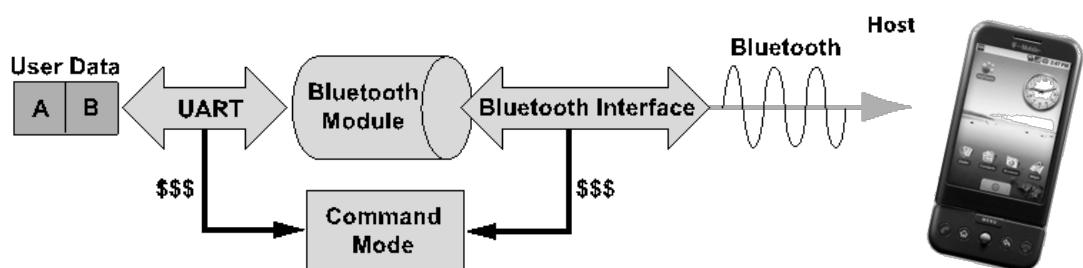
Para el desarrollo de aplicaciones utilizaremos los módulos Bluetooth® Microchip / RN, para facilitarnos la tarea de armado, existen kits de evaluación para esos módulos utilizando el perfil SPP – Bluetooth. La Placa de evaluación RN-42-SM, contiene el modulo RN-42, un convertidor RS-232, LEDs de estado, puentes para configuración, modos de auto – conexión, factory reset, baud rate (9600 o 115,200) y un regulador de tensión.

En la imagen siguiente vemos estos módulos. Estos módulos poseen modos de trabajo los cuales nos facilitan la configuración y la transferencia de datos en las aplicaciones que desarrollemos



## MODOS DE DATO & COMANDO

El modulo puede trabajar en dos modos distintos, un modo llamado comando el cual se accede con \$\$\$, en este modo, se pueden realizar configuraciones como por ejemplo el baud rate, nombre del dispositivo, código del pin, etc. Existe un temporizador de 60 seg el cual limita el tiempo configuración por bluetooth a una ventana de 60 segundos, el otro modo llamado datos (estado por defecto) el cual nos permite crear un túnel de datos UART <> Puerto COM siendo una conexión transparente para el usuario (dato escrito sobre la UART es enviado a Bluetooth, datos recibido sobre bluetooth se escribe sobre la UART)



Para poder configurar los parámetros de este equipo contamos con los siguientes comandos los cuales se enviaran por el puerto serie del microcontrolador hacia el modulo.

SC,<valor>	Configura el COD de MSW
SD,<valor >	Configura el COD de LSW
SH,<valor >	Configura registro HID
SM,<valor >	Configura el modo del dispositivo
SN,<valor >	Configura nombre del dispositivo
SP,<valor>	Configura el PIN de seguridad
SY,<valor>	Configura la Potencia de salida
S~,<valor>	Configura el perfil
SW,<valor>	Configura el modo de escucha (sniff)

Comando para la lectura de datos

D	devuelve la configuración básica
E	devuelve configuración extendida
GB	devuelve la dirección del dispositivo
GF	devuelve la dirección del dispositivo conectado
GK	devuelve el estado de la conexión

Algunos comandos de cambio /acción

\$\$\$	entra en modo comando
---	sale del modo comando
C	Se conecta a direcciones remotas almacenadas
C,<dirección>	Conecta a una dirección
R,1	Reinicio del modulo

I,<valor1>,<valor2>	<b>La búsqueda escanea por el valo1 segundos, utilizando el valo2 como COD del dispositivo, devolviendo la dirección y el nombre</b>
IN, <valor1>,<valor2>	igual que el anterior, pero no devuelve el nombre solo la dirección
IQ	El modulo escanea y devuelve el RSSI de los dispositivos locales
IS <valor1>	Inicia un escaneo con una mascara COD de 0x001F00 por valor1 seg.
IR <valor1>	Inicia un escaneo con un COD de 0x0055AA por valor1 seg.

## QUE ES EL COD ?

Un COD (Class Of Device) es un número que describe el dispositivo Bluetooth. Lo compone una palabra de 24bit con 3 campos, COD mayor, COD menor y Servicio COD como se muestra a continuación:

```
COD_MINOR_MASK = 0x0000FC  
COD_MAJOR_MASK = 0x001F00  
COD_SERVICE_MASK = 0xFFE000
```

Algunas clases de COD mayores

```
COD_MAJOR_MISCCELLANEOUS = 0x00  
COD_MAJOR_COMPUTER = 0x01  
COD_MAJOR_AUDIO = 0x04  
COD_MAJOR_UNCLASSIFIED = 0x1F
```

Por ejemplo es importante que en vez de aparecer en una búsqueda como una sucesión de números y letras posea un nombre más amigable. Para lograrlo, bastará con abrir el software de comunicación serie de nuestra preferencia (por ejemplo, el Hyperterminal) y escribir: \$\$\$, es decir, el símbolo de pesos, tres veces seguidas. Si el módulo lo recibe en forma correcta, devolverá las letras CMD seguido por un signo de pregunta, de igual modo que si estuviera esperando nuevas instrucciones. Luego de esto, escribimos SN, . Nosotros escribimos: **SN,MCEbluetooth.**

El módulo devolverá un AOK, indicándote que recibió la instrucción, y luego otro signo de interrogación. Este proceso se repite hasta que termines de enviar comandos AT de configuración al módulo. Al finalizar, se retira la alimentación, se reestableces y ¡listo! El módulo actuará según los comandos enviados de configuración.

*¡Atención! Existen dos tipos diferentes de comandos: unos determinan el funcionamiento del módulo (como los que te indicamos recién) y otros le indican acciones en tiempo real. En los primeros debes reiniciar la alimentación, mientras que para los segundos la acción se realizará en tiempo real, o sea, cuando tú envíes el comando AT.*

Configuración demostrativa HID

Utilizando el hyperterminal en la PC o laptop para hablar por serie al RN41EK

Pasos:

- Conectar a la tablet / phone

Creando un dispositivo HID

- \$\$\$
- SN, MCHIP\_KEYS <CR>

- SH, 0200 <CR>
- S~,6 <CR>
- R,1 <CR>

Bits del registro de banderas

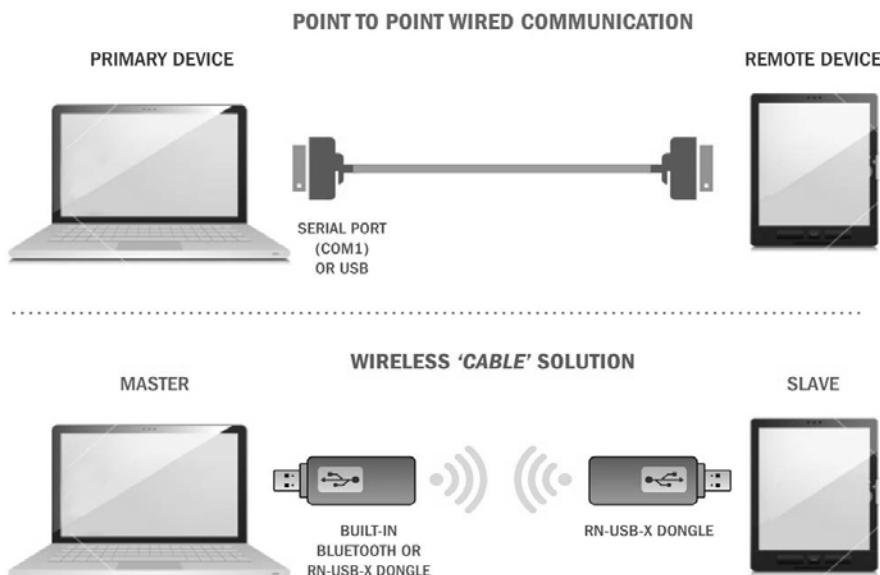
<b>9</b>	<b>8</b>	<b>7..4</b>	<b>3</b>	<b>2..0</b>
Fuerza al modo HID si el GPIO11 está en alto en el encendido	Cambiar el Teclado Virtual iOS	tipo 0000 = keyboard 0001 = game pad 0010 = mouse 0011 = combo 0100 = joystick 1xxx = reservado	Enviar una salida de reporte sobre la uart	# de los dispositivos apareados que el modulo puede conectarse

### Descubriendo y conectándose desde la PC

En su PC, tablet, teléfono, ir a Bluetooth®, administración y buscar por teclado, luego conéctese el equipo, y usted ahora tiene un teclado BT HID

#### Bluetooth® con perfiles: SPP

Emular un puerto COM sobre Bluetooth, Características: aplicaciones de reemplazo de cables, drivers Bluetooth en la computadora host y crea un puerto COM virtual luego de una asociación exitosa, cuando el puerto COM es abierto, la conexión Bluetooth del esclavo es abierta.



## CREANDO UN DISPOSITIVO SPP

\$\$\$  
SN, Explorer16 <CR>  
S~,0 <CR>  
R,1 <CR>  
\$\$\$ pone el modulo en modo comando  
SN permite al usuario configurar el nombre del dispositivo  
S~ configura el perfil del modulo a SPP  
R,1 reboots la unidad con la nueva configuración

## Configuración demo del SPP. Descubriendo y conectando a una PC



- Conecte el kit de evaluación al puerto USB de una PC
- Vaya al modo comando y resetee a los valores de fabrica al modulo
- Ingrese los comandos vistos anteriormente
- En su PC, ir al administrador de Bluetooth® y hacer click en “Add new device”

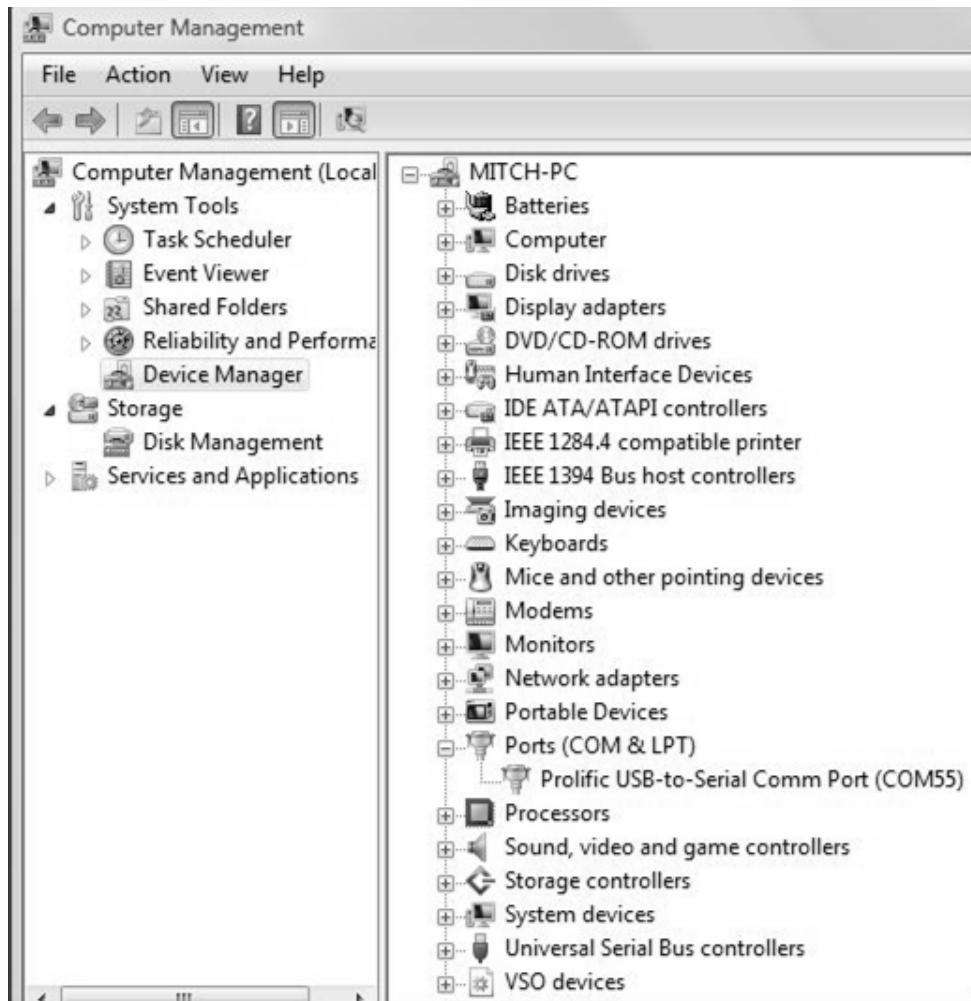
El dispositivo se mostrara como *RN42-ABCD*, cuando pregunte, ingrese el código 1234, un teclado Bluetooth será creado por Windows sin la necesidad de instalar drivers.

## Descubriendo y conectando desde una PC

Revise el puerto COM creado en administrador de programas de la PC  
Luego Navegue por los puertos (COM & LPT)

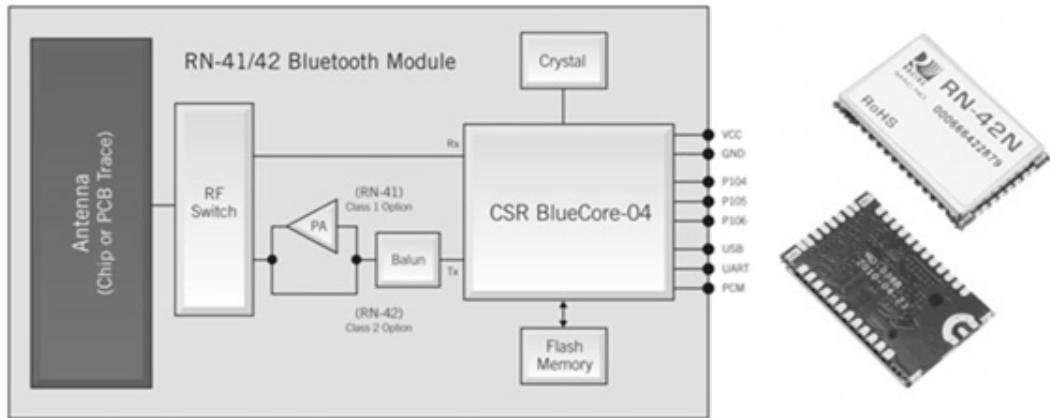
Mire donde diga “Standard Serial over Bluetooth Link”

**NOTA:** A veces, Windows creara dos puertos COM y los llamara “incoming” y “outgoing”. En este caso, utilice el “outgoing port”

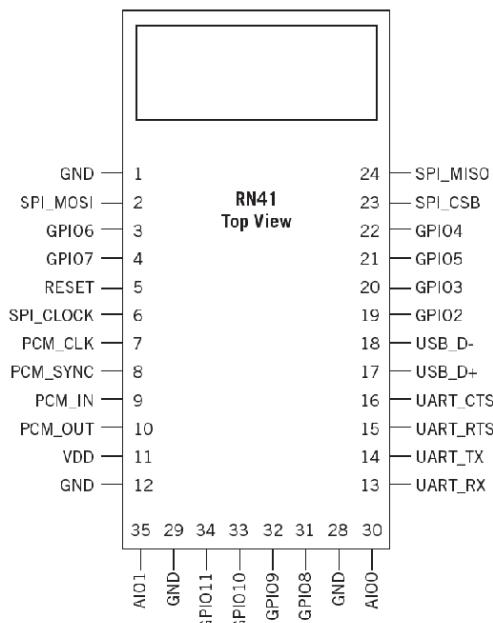


## CONSIDERACIONES DE HARDWARE

Como nos muestra el esquema, todo el hardware y stack bluetooth está en el modulo, esto nos permite utilizar microcontroladores de bajos recursos ya que no deberán cargar con la ardua tarea de soportar el código para el stack.



Es importante apreciar que el modulo posee conexión USB pines D+ y D- (18 y 17) y también conexión por medio de la UART, Rx, Tx (13 y 14 respectivamente). Solo una conexión es soportada en el momento de la implementación.



## **CONFIGURACIÓN DEL MODULO SOBRE EL PUERTO UART**

Conecte el modulo a su computadora, esto se realiza mediante un conector DB9 o por medio de un cable USB adecuado que por lo general contiene un conversor. Por ejemplo si utilizamos el Kit de evaluación RN-41-EK, se puede conectar utilizando un cable USB simple.

Una vez conectado el modulo y alimentado, se deberá ejecutar el programa emulador de una Terminal y abrir un puerto COM donde el cable fue conectado. La configuración del programa emulador deberá ser:

- Velocidad 115,200 kbps
- 8 bits
- Paridad No
- Bit de stop 1
- Control de flujo por Hardware Habilitado

Nota: Se puede utilizar la configuración local en cualquier momento cuando el módulo no tiene una conexión Bluetooth. Si el módulo está en el modo de configuración y se produce una conexión, el módulo sale del modo de configuración y los datos pasan a través del modulo y el puerto de trabajo

Una vez que se realiza una conexión, sólo se puede entrar en el modo comandos si el temporizador de configuración en el arranque no ha expirado (60 segundos). Para permanecer en el modo de configuración, debemos establecer la configuración de temporizador en 255. Si el módulo está en conexión automática en modo maestro, no puede entrar en el modo de comandos cuando se conecta a través de Bluetooth.

### **Configuración remota utilizando Bluetooth**

A menudo, es útil poder configurar el módulo remoto en forma remota a través de una conexión Bluetooth. Antes de realizar configuración remota mediante Bluetooth debemos previamente aparear el modulo con nuestra computadora

Para PCs con capacidad de Bluetooth y con sistema operativo Windows, hacer click en dispositivos Bluetooth de la bandeja de sistemas y con el botón derecho seleccionar agregar dispositivos Bluetooth, luego debemos seguir las instrucciones en la pantalla.

Una vez finalizada la configuración, se debe resetear el modulo o enviar el comando --- el cual produce que salga del modo configuración y permite que fluyan los datos normalmente.

## **Modo comando**

Para ingresar en el modo comando hay que cargar en la PC el emulador de terminal y activar al modulo preconfigurado (default).

### **Configuración del puerto serie**

Puerto COM, es el puerto donde se conecto el modulo

Velocidad 115200

Datos 8 bits

Paridad No

Bits de parada 1

Control de flujo No

Se deberá escribir \$\$\$ en la Terminal para entrar en modo comando.

El modulo devolverá la sentencia CMD, la cual indica que la configuración de la terminal y el modulo son las correctas.

Mientras se está en el modo comando, el modulo acepta bytes ASCII como comandos para ejecutar ordenes y realizar configuraciones.

Cuando se ingrese un comando valido el modulo devuelve AOK, si el comando es erróneo devolverá ERR y para un comando no reconocido, ?.

Si escribimos h <cr> veremos una lista de los comandos disponibles por el modulo. Una forma rápida de confirmar que se está en el modo comando es escribir X <cr> antes de tratar de ingresar al modo comando, este comando la configuración actual del modulo en forma completa como el nombre del modulo, la clase de dispositivo, la configuración del puerto serie.

Para volver al modo de datos, se debe escribir --- <cr> o resetear el modulo y reconectarnos.

## Modo de operación

El modulo posee varios modos de funcionamiento, el cual puede ser configurado utilizando el comando SM.

Nota; En todos los modos maestro, el modulo no puede ser descubierto o configurado en forma remota.

*Modo esclavo (SM,0)*— Es el modo por defecto, otros dispositivos Bluetooth pueden descubrir al modulo y conectarse a el. También se pueden realizar conexiones salientes en este modo.

*Modo maestro (SM,1)*— En este modo de baja velocidad, el modulo realiza la conexión cuando un comando de conexión ( C ) es recibido. Este comando también puede contener la dirección Bluetooth del dispositivo remoto. Si esta no se especifica, el modulo utilizará los datos que tenga almacenados. Las conexiones pueden ser cortadas en cualquier momento si se envía un carácter especial de corte o una palabra, para configurar este comando, se utiliza el la instrucción SO. Este modo es útil cuando se desea que le modulo inicie la conexión, en este modo el modulo no se puede descubrir.

*Modo Trigger (SM,2)*—En esta conexión de baja velocidad el modulo realiza la conexión automáticamente cuando se recibe un carácter en el puerto serie (UART). La conexión se mantiene mientras se reciban caracteres. El modulo posee un cronómetro que puede ser configurado para que se desconecte luego de un determinado tiempo de inactividad (Se utiliza el comando ST con tiempos de 1 a 255), también se puede configurar un carácter de corte que al llegar abre la conexión.

*Modo Maestro de Auto-Conexión(SM,3)*— en este modo, el modulo inicia la conexión automáticamente al encender y la reconecta si esta se pierde. Esta configuración puede ser establecida mediante comandos o por medio de interruptores como los que posee el modulo de desarrollo. También puede ser configurado estableciendo un estado alto en el pin GPIO6.

Si no hay direcciones almacenadas, el modulo comienza un proceso de consultas y el primer dispositivo encontrado que concuerda con el COD es almacenado. En este modo, los datos a alta velocidad son pasados sin ser interpretados, por lo tanto, la conexión no puede ser rota por medio de comandos o caracteres de interrupción por software. Si se produce una desconexión, el módulo intenta volver a conectar hasta que tenga éxito.

*Modo DTR Auto conexión(SM,4)*—Este modo debe ser configurado por medio de comandos. Funciona como el modo *Maestro de Auto-Conexión*, excepto que podemos controlar la conexión y desconexión por medio de GPIO6. Con un nivel alto en este pin, iniciamos el proceso de auto conexión, con un nivel bajo en dicho pin causa la desconexión.

*Modo ANY Auto conexión (SM,5)*— Este modo debe ser configurado por medio de comandos. Este modo funciona como el modo DTR auto conexión con la excepción de que cada vez que el pin GPIO se establece, se realiza una búsqueda y el primer dispositivo que se encuentra es conectado. La dirección almacenada no es utilizada y la dirección encontrada nunca es almacenada.

*Modo de asociación (SM,6)*- En este modo, el modulo trata de conectarse con el dispositivo remoto que concuerda con los datos almacenados, se debe configurar la dirección utilizando el comando SR

## PINES GPIO PARA CONFIGURACIÓN

El modulo Bluetooth posee pines que pueden ser utilizados para configurar el modulo:

Función	Pin GPIO	Configuración (OFF=0, ON=3Vcc)
Reseteo de fabrica	GPIO4	Off=deshabilitados, On=Habilitado Si se configura este pin al alimentar el modulo, se activa el reset.
Auto descubrimiento/asociación	GPIO3	Off=deshabilitado/On=habilitado Se utiliza en conjunto con GPIO6, si GPIO6 esta activado, comienza una búsqueda, una vez encontrado se almacena la información. Si GPIO6 no esta activado, el modulo entra en modo esclavo y espera ser encontrado por un maestro.
Auto conectar	GPIO6	Off=deshabilitada/On=habilitado Este modo es equivalente al modo maestro con auto conexión en software. El modulo se conecta a la dirección almacenada.
Velocidad	GPIO7	Off=Almacenada(115K), On=9600

## Conexión bluetooth

De fabrica, el modulo actúa como un esclavo y la PC o smartphone es el maestro.

El proceso de conexión independiente del dispositivo es muy similar:

- Descubrir—En la fase de descubrimiento, el modulo difunde su nombre. Los otros dispositivos deben estar preparados para conectarse a él. Este modo solo está disponible en modo esclavo.
- Asociarse – Durante este proceso, el modulo es validado con el maestro mediante el pin. Si el pin se valida en forma exitosa, se intercambiaran las claves de seguridad y la secuencia de salto pseudoaleatoria del canal. El sincronismo se logra cuando el modulo y el maestro establecen las claves de conexión.
- Conexión— Antes de la conexión, los dispositivos Bluetooth deben estar sincronizados correctamente. El maestro inicia la conexión, el maestro y el esclavo validan las claves y la conexión bluetooth se establece.

## Descubrir

Cuando se enciende el modulo Bluetooth, este puede ser descubierto. En la PC, en dispositivos Bluetooth, seleccionar agregar un nuevo dispositivo. En el administrador de dispositivos aparecerá el modulo. Este será visto como un servicio Serial Port Profile (SPP) **FireFly-ABCD**, donde **FireFly** es el tipo de modulo de Roving Networks y ABCD son los últimos numero de la dirección MAC. Este nombre se puede cambiar



## Asociación

Para asociar el modulo, hay que hacer doble click en el nombre del modulo (ejemplo: **FireFly-XXXX**) sobre la lista que aparece.

El firmware automáticamente almacena hasta 8 asociaciones, una vez alcanzado este número a medida que se ingresan nuevos módulos, los más viejos son eliminados. Seleccione código de asociación e ingrese el código configurado de fábrica 1234.

Cuando el administrador de dispositivos bluetooth completa la asociación, enviará un mensaje informando en qué puerto se encuentra mediante la leyenda COMX donde X es el número de puerto asignado.

Select a pairing option

- Create a pairing code for me  
The device has a keypad.
- Enter the device's pairing code  
The device comes with a pairing code.  
Check for one on the device or in the device manual.
- Pair without using a code  
This type of device, such as a mouse, does not require a secure connection.



FireFly-9959

Si el dispositivo bluetooth remoto no necesita autenticación, la conexión puede ocurrir sin el proceso de asociación. Sin embargo, las especificaciones de Bluetooth requieren que si

cualquiera de los dispositivos involucrados en el proceso de asociación requiere una autenticación, el otro dispositivo debe participar para asegurar un enlace seguro.

Los módulos de Roving Networks por defecto están en modo abierto, de tal manera que el módulo no requiere autenticación. Sin embargo la PC requiere autenticación por lo que se necesita las claves.

Una vez conectados, el modulo estará en modo datos permitiendo el flujo en ambas direcciones como si el puerto serie del modulo estuviera conectado localmente en la PC

### **Asociando con una computadora o smart phone**

El módulo puede utilizar una asociación segura simple (SSP) si es que se intenta asociarse con dispositivos que admiten la especificación Bluetooth versión 2.1 + EDR. SSP no necesita

### **Conectando**

En la mayoría de los casos, nos conectamos desde otro dispositivo al módulo como una conexión Bluetooth saliente. También se puede realizar una conexión entrante en el que la placa de evaluación inicia la conexión con el dispositivo remoto.

Para establecer una conexión Bluetooth salientes desde un PC al módulo, abra el puerto COM de salida con una aplicación o un emulador de Terminal. El modulo permanecerá conectado hasta que se cierre el puerto o se quite la alimentación del modulo.

Una vez conectado, el modulo estará en modo datos permitiendo el flujo en ambas direcciones. Solo un cliente se puede conectar al modulo esclavo a la vez. Este modulo no soporta un modo maestro multi punto.

### **Conexiones entrantes**

Para una conexión entrante debemos utilizar el puerto especificado en la configuración de Bluetooth como entrantes. En las conexiones entrantes, la PC o el host escucha para una conexión entrante desde el dispositivo Bluetooth remoto, en este caso, el módulo realiza los siguientes pasos para establecer la conexión:

1. Se necesita la dirección MAC del modulo de radio Bluetooth de la PC para conectarse desde el modulo a la PC. Abrir la configuración avanzada de la PC para conexiones Bluetooth para encontrar la dirección MAC.
2. Asociar el modulo con la PC como explicamos antes.
3. Abrir la Terminal en la PC o host (llamaremos A en el ejemplo) y conectarse al modulo

4. Abrir una segunda terminal (terminal B en este ejemplo) en la PC para escuchar a la futura conexión de Bluetooth a través del número de puerto COM entrante.
5. Escribir c,<dirección MAC><cr> en la Terminal B para establecer una conexión SPP con la PC
6. Pruebe los siguientes comandos:
  - \$\$\$ para entrar en modo comando
  - SO,% para habilitar los mensajes de estado y ver la condición de conectado/desconectado
  - R,1 para reinicio
  - \$\$\$ para reentrar en modo comando
  - + para habilitar el eco local
  - C,<dirección MAC> para establecer la conexión con el dispositivo remoto

Los caracteres que se escriben en la Terminal B son enviados sobre bluetooth a la PC (Terminal A), todo lo ingresado en la Terminal A aparece en la B.

7. Para finalizar la conexión se debe escribir el comando k, 1 <CR> en la Terminal B.

### **Modos de seguridad**

El módulo Bluetooth admite la autenticación. Si el dispositivo Bluetooth local o remoto tiene habilitada la autenticación, se necesita un código pin la primera vez se intenta una conexión. El código pin es una serie de números o caracteres de 1 a 16 caracteres de longitud.

El código pin predeterminado para módulos Bluetooth Roving Networks es 1234. Después de introducir el código pin, los dispositivos Bluetooth los comparan. Si coinciden, se genera una clave de enlace y se almacenan en la memoria. Normalmente, pero no siempre, el dispositivo remoto almacena la clave de vínculo.

Para las conexiones posteriores, los dispositivos comparan las claves de vínculo. Si son correctos, no es necesario que vuelva a introducir el código pin.

## Modo de escucha

Modo escucha, un método de conservación de energía, se refiere sólo a una conexión activa. En los dispositivos Roving Networks, el modo de escucha se encuentra desactivada de forma predeterminada y la radio está continuamente activa cuando está conectado (consumo alrededor de 25 a 30 mA). En el modo de escucha, la radio se despierta a intervalos y se duerme en un modo de muy bajo consumo el resto del tiempo (aprox 2mA).

Para habilita el modo de escucha, se debe utilizar el comando SW,<valor en hex>

Ejemplo de intervalos:

0x0020 = 20 ms (32 decimal \* .625 = 20)

0x0050 = 50 ms

0x00A0 = 100 ms

0x0190 = 250 ms

0x0320 = 500 ms

0x0640 = 1 segundo

Cuando se establece una conexión, tanto el maestro como el esclavo deben admitir modo de escucha y acordar la ventana de escucha, de lo contrario, la radio sigue en modo activo.

**Nota:** El máximo intervalo es de 20 seg el cual es 0x7FFF.

## HABILITAR EL MODO SLEEP

Usted puede utilizar un modo de suspensión para obtener un consumo de energía extremadamente bajo. En este modo, el dispositivo se apaga por completo y sólo se consumen 300 µA de corriente aproximadamente.

Para habilitar el modo de suspensión, hay que establecer el bit más significativo del registro de escucha a 0x8000. Este bit NO se usa para determinar el intervalo, sólo se utiliza como un indicador para habilitar la suspensión.

Ejemplo: SW,8320 // ½ Segundo de suspension (0x0320)

En modo de suspensión normal de baja potencia, el firmware aún se está ejecutando en modo de inactividad, y se despierta cerca de 20 veces por segundo para comprobar el estado de los puertos, actualizar los led, etc. Durante la suspensión, el firmware en realidad deja de ejecutar algunas tareas. Por ejemplo, la actualización de los LED únicamente una vez por segundo (si se conectan a los pines). Hay 3 formas de activar la radio desde el modo de reposo:

- Enviar un carácter a la UART.
- Transición del pin RX. El peor de los casos, el tiempo de activación es de 5 ms. Por lo tanto, la radio generalmente pierde el primer carácter. La mejor manera de activar el radio es para cambiar la línea CTS de un estado bajo a uno alto, luego esperar 5ms y luego enviar los datos.
- La radio puede activarse automáticamente cada ciertas ventanas de tiempo (1 ventana de tiempo es de 625 µs) . La radio se despierta y escucha a ver si el otro lado de la conexión no tiene nada que enviar. Este tiempo de activación es de aproximadamente 5 ms (8 ventanas) incluso si no hay datos que se van a transferir.

Una vez que se despierta la radio, permanece activa durante exactamente 1 segundo de inactividad y, a continuación, duerme otra vez.

Para activar incluso reducir la utilización de energía, utilice el comando S|,<valor hex>, para establecer un ciclo de trabajo.

## Deshabilitar los drivers de salida

Utilice el comando S%,1000 para configurar todos los pines GPIO (0 al 11) como entradas.

## Potencia de transmisión

Todos los módulos Bluetooth Roving Networks poseen control de potencia. La potencia de salida de radio es controlada automáticamente por la banda base. Dependiendo de la modalidad (búsqueda, escaneo, conectado), la potencia se ajusta. Una vez que la conexión se ha establecido, las radios de los dos lados negocian un ajuste de energía en función de la potencia de la señal recibida (RSSI ).

La potencia de Tx puede ser ajustada para controlar:

- Reducción del rango efectivo por temas de seguridad
- Baja emisión de radio según regulaciones del estado.
- Reducir el consumo

Para configurar el nivel de potencia de transmisión, utilice el comando SY<valor hex>, donde el valor es un número sin signo. Ejemplos de valores:

### Potencia en Hex (dBm)

0010 16 (predeterminada)

000c 12

0008 8

0004 4

0000 0

FFFC -4

FFF8 -8

FFF4 -12

Si un valor distinto de cero se almacena en la variable, la potencia de la radio utiliza el valor más alto que es menor o igual a la variable.

Ejemplo: SY,FFF8 // -8 dBm

SY,0000 // 0 dBm

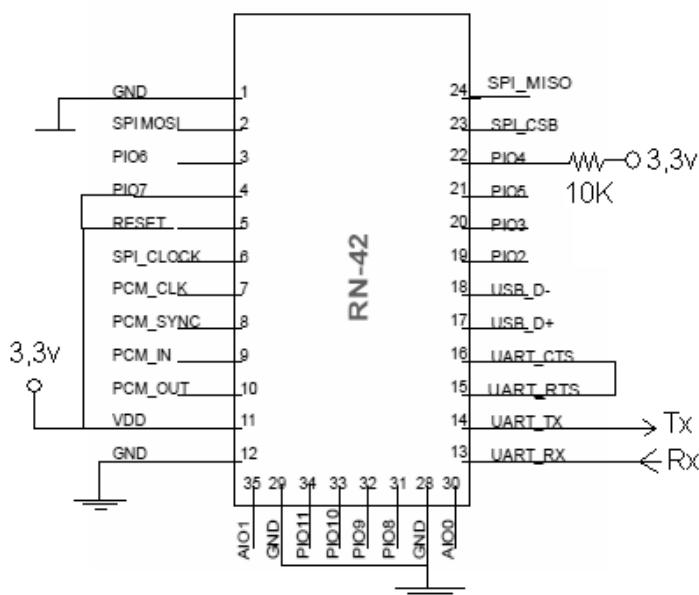
El ajuste del parámetro de la potencia tiene efecto después de un ciclo de apagado y encendido, reinicio o reset por medio del pin. Para comprobar la configuración de la potencia, utilice el comando O; el dispositivo devuelve TX=1 (u otro ajuste igual al valor introducido). Para ver la configuración actual, utilice el comando GY. El dispositivo devuelve sólo el valor real sin ceros iniciales. El valor GY puede no coincidir con el valor mostrado por el comando O debido a la forma en que el valor se utiliza como se describió anteriormente.

## CONECTANDO A UN MICROCONTROLADOR

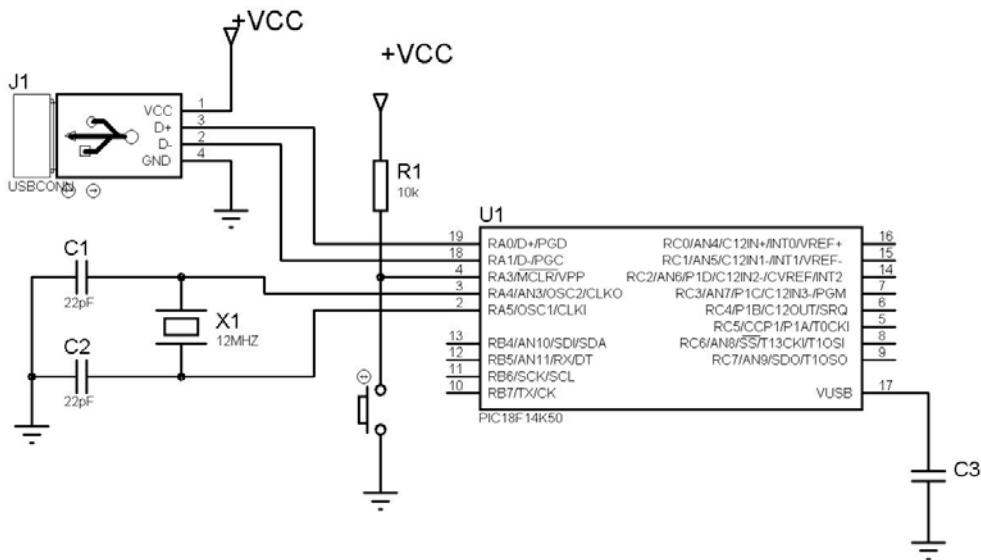
Los dispositivos Bluetooth Roving Networks pueden conectarse a microprocesadores de 3.3-V únicamente mediante la interfase UART. Cuando diseñe una conexión se deberán seguir los siguientes lineamientos:

- El dispositivo Bluetooth deberá tener conectado Vcc (3,3v), GND, RX, y TX . CTS debe ser conectada a un nivel bajo o unida a RTS
- El dispositivo Bluetooth puede ir al modo comando 500ms después de encender
- El microcontrolador deberá enviar \$\$\$ sin el enter o fin de línea para ingresar al modo comando

En la figura siguiente se aprecian las conexiones del modulo



Este debe ser vinculado al microcontrolador por medio de RX,TX y GND. Asimismo el microcontrolador deberá tener una configuración mínima para funcionar como la que se muestra a continuación.



En este caso el circuito está preparado para que el microcontrolador actúe como un puente entre el puerto USB de la PC y la UART del modulo.

El programa a cargar en el modulo puede ser obtenido del sitio de microchip dentro de las hojas de datos del microcontrolador bajo el nombre de Low Pin-Count USB Development Kit Project Labs, en la dirección:

[http://www.microchip.com/stellent/idcplg?IdcService=SS\\_GET\\_PAGE&nodeId=1406&dDocName=en536385](http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en536385)

Al finalizar podremos controlar el modulo desde la PC y practicar los comando enunciados anteriormente.

## **PERFIL HID**

Los módulos Bluetooth Roving Networks (Microchip) soportan una gran variedad de perfiles, incluyendo human interfase device (HID), serial port profile (SPP), DUN, HCI, y iAP para utilizar con dispositivos iPad, iPod y iPhone.

El perfil Bluetooth HID permite a los clientes desarrollar productos inalámbricos como teclados, trackballs, ratones y otros dispositivos señaladores, juegos (gamepads, joysticks, volantes, etc.).

Además, Roving Networks ha ampliado la capacidad básica del HID para permitir la programación y control de dispositivos como el iPad.

El perfil HID define el protocolo entre:

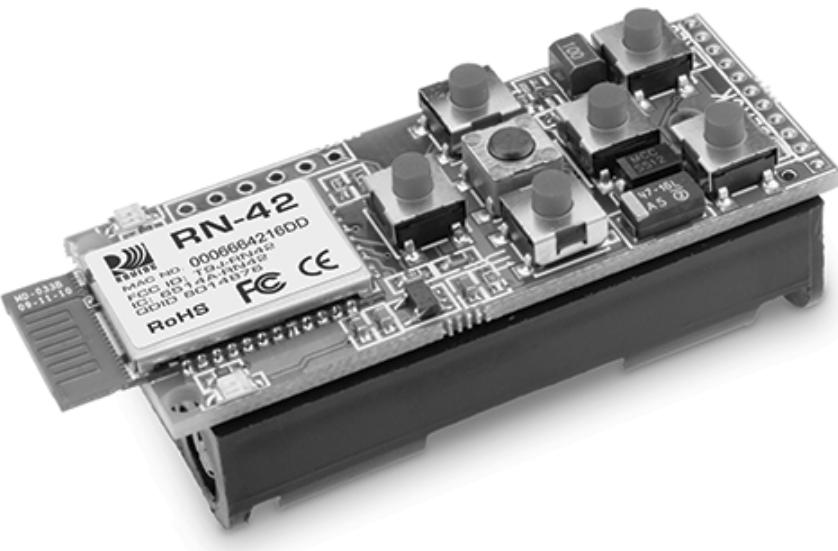
- *Dispositivo (HID)*— Servicios humanos de entrada y salida de datos hacia y desde el host.
- *Host*— Utiliza las solicitudes o los servicios de un dispositivo de interfaz humana.

El perfil Bluetooth HID permite a los usuarios controlar el descriptor HID, que define el conjunto de características del dispositivo, y el informe HID, que host utiliza para interpretar los datos como valores ASCII , movimiento, etc.

En un escenario típico, como un teclado, un dispositivo Bluetooth Roving Networks que utiliza un perfil HID reemplazara al cable USB. En este caso, el valor ASCII de una pulsación de tecla se convierte en un código de captura de un reporte HID que el modulo Bluetooth envía sobre el enlace Bluetooth al host. El software del controlador del host decodifica el dato y pasa los valores a la aplicación que se está ejecutando en el equipo. La figura muestra algunos de los típicos entornos HID

El tipo de dispositivo HID, tales como un teclado, un ratón o joystick, es definido por el descriptor HID, se debe utilizar una versión especial del firmware, la versión 6,03 o posterior

Para la realización de prácticas existe comercialmente un modulo el cual permite experimentar varios dispositivos HID como así también el acceso a la UART y realizar configuraciones personalizadas:



*Modulo de Microchip con RN-42 para realizar prácticas HID.*



# BIBLIOGRAFÍA



## **LIBROS:**

- Katsuhiko Ogata (2009). *Modern Control Engineering* (5th Edition). USA. Editorial: Prentice Hall
- Cheng, David K. (1998). *Fundamentos de electromagnetismo para ingeniería*. México. Editorial: Addison Wesley Longman
- Martín Cuenca, Eugenio (2003). *Microcontroladores PIC : la clave del diseño*. Madrid. Editorial: Thomson
- Bruno Saravia, Andres (2010). *Arquitectura y Programación de microcontroladores PIC*. Buenos Aires. Editorial: mcelectronics
- Airoldi, Alejandro Anibal (2011). *MPLAB X y Técnicas de Programación con librerías de Microchip*. Buenos Aires. Editorial: mcelectronics

## **HOJAS DE DATOS:**

- Microchip Technology Inc. (2011). Hoja de datos PIC18F26J50. USA. [www.microchip.com](http://www.microchip.com)
- Microchip Technology Inc. (2011). Hoja de datos PIC18F46K20. USA. [www.microchip.com](http://www.microchip.com)
- Microchip Technology Inc. (2009). Hoja de datos PIC18F97J60. USA. [www.microchip.com](http://www.microchip.com)
- Nilesh Rajbharti , Microchip Technology Inc. (2008). Microchip TCP/IP Stack Application Note. USA. [www.microchip.com](http://www.microchip.com)
- SIMCOM LTD. (2012). Hoja de datos módulo SIM908. China.
- SIMCOM LTD. (2012). SIM908\_AT Command Manual\_V1.01
- SIMCOM LTD. (2012). SIM900\_AN\_TCPIP\_V100.
- Micro Modular Technologies Pte. Ltd. (2008). Hoja de datos módulo MN5515HS. Singapore. [www.micro-modular.com](http://www.micro-modular.com)

***Todas las hojas de datos y notas de aplicación mencionadas se encuentran en el DVD que acompaña este libro.***

## **NOTAS:**

## **NOTAS:**

## **NOTAS:**



**ELEMON**  
Componentes Electrónicos

  
**MICROCHIP**

## Para estar un paso adelante, soluciones innovadoras Microchip

### Microcontroladores

La línea más completa del mercado en 8, 16 y 32 bits, incorporando una amplia gama de recursos de conectividad (USART, I2C, SPI, CAN, USB, ETHERNET), driver de LSD, y versiones de bajo consumo y baja tensión de operación.



#### Herramientas de desarrollo

Un único entorno de desarrollo integrado para toda la familia de micros, el MPLAB IDE (disponible sin costo en la web), Programadores actualizables vía web PICkit 3, In circuit debuggers MPLAB ICD 3, Kits de evaluación (Explorer 16).

#### Soporte on line

- **Notas de aplicación en:** [www.microchip.com/appnotes](http://www.microchip.com/appnotes)
- **Todo sobre tecnología USB en:** [www.microchip.com/usb](http://www.microchip.com/usb)
- **Soluciones de diseño sobre Ethernet en:** [www.microchip.com/ethernet](http://www.microchip.com/ethernet)
- **Conectividad inalámbrica en:** [www.microchip.com/zigbee](http://www.microchip.com/zigbee) | [www.microchip.com/mlwi](http://www.microchip.com/mlwi)
- **Centro de diseño para el control de motores en:** [www.microchip.com/motor](http://www.microchip.com/motor)
- **Todo sobre comunicación CAN en:** [www.microchip.com/can](http://www.microchip.com/can)

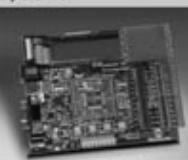
PICkit®3



MPLAB® ICD 3



Explorer 16



DISTRIBUIDOR  
AUTORIZADO EN LA  
ARGENTINA

[www.elemon.net](http://www.elemon.net)

**CASA CENTRAL**  
Capdevila 2707 | C1431FKA | Buenos Aires | Argentina  
Tel.: (5411) 4523 5555 | 3909 | 3910 | 4142 | Fax: (5411) 4522 7335  
[ventas@elemon.com.ar](mailto:ventas@elemon.com.ar)



**ELEMON**  
Componentes Electrónicos

 **Littelfuse®**

Expertise Applied | Answers Delivered



Confiabilidad  
a la hora  
de proteger  
sus circuitos  
electrónicos



# Líder mundial en componentes electrónicos

## Tecnología Switching

- TRIACS
- TIRISTORES
- SIDACs
- DIODOS
- DIACS

## Supresores de sobretensión

- DIODOS SUPRESORES DE TRANSITORIOS
- ARRAYS PROTECTORES DE SILICIO
- VARISTORES
- SUPRESORES ESD

## Protectores de sobrecorriente

- FUSIBLES Y PORTAFUSIBLES
- PTC RESETEABLES
- FUSIBLES DE MONTAJE SUPERFICIAL



DISTRIBUIDOR  
AUTORIZADO EN LA  
ARGENTINA

CASA CENTRAL  
Capdevila 2707 | C1431FKA | Buenos Aires | Argentina  
Tel.: (5411) 4523 5555 | 3909 | 3910 | 4142 | Fax: (5411) 4522 7335  
ventas@elemon.com.ar

[www.elemon.net](http://www.elemon.net)

# Equipos inteligentes decisiones inteligentes



## MÓDULO GSM SIMCom | Modelo SIM900

**Simcom, líder mundial en tecnología inalámbrica junto con Electrónica Elemon le ofrece la mejor relación costo beneficio en comunicaciones M2M.**

- Cuatro Banda 850/ 900/ 1800/ 1900 MHz
- GPRS multi-slot clase 10/B
- GPRS estación móvil clase B
- Cumple con GSM phase 2/2+
- Clase 4 (2 W @ 850/ 900 MHz)
- Clase 1 (1 W @ 1800/1900MHz)
- Dimensiones: 24mm x 24mm x 3mm
- Peso: 3.4 gramos
- Control vía comandos AT (GSM 07.07, 07.05 y comandos AT SIMCOM mejorados)
- SIM application toolkit
- Rango de Alimentación: 3.1 ... 4.8VDC.
- Bajo consumo de Energía: 1.5mA (sleep mode)
- Temperatura de operación: -40°C to +85 °C
- Group 3, class 1
- GPRS clase 10: max. 85.6 kbps (downlink)
- Soporta PBCH
- Esquemas de Codificación CS 1, 2, 3, 4
- CSD up to 14.4 kbps

<p>Accesorios para módulos GSM</p>	<p>Módulos GSM</p>	<p>Kits de desarrollo</p>
------------------------------------	--------------------	---------------------------

CASA CENTRAL Capdevila 2707 | C1431FKA | Buenos Aires | Argentina  
Tel: (5411) 4523 5555 | 3909 | 3910 | 4142 | Fax: (5411) 4522 7335 | [ventas@elemon.com.ar](mailto:ventas@elemon.com.ar)  
SUCURSAL CÓRDOBA Rufino Cuervo 1085 Local 5 | X5009GAA  
Barrio Las Rosas | Córdoba  
Tel: (0351) 489 8005 | 8600 | [ventascordoba@elemon.com.ar](mailto:ventascordoba@elemon.com.ar)

DISTRIBUIDOR  
AUTORIZADO EN LA  
ARGENTINA

**[www.elemon.net](http://www.elemon.net)**



TDK-EPC

Donde quiera que haya  
electrónica está TDK-EPC

TDK

Dos grandes  
marcas se unen

EPCOS



DISTRIBUIDOR  
AUTORIZADO EN LA  
ARGENTINA

CASA CENTRAL Capdevila 2707 | C1431FKA | Buenos Aires | Argentina  
Tel: (5411) 4523 5555 | 3909 | 3910 | 4142 | Fax: (5411) 4522 7335 | ventas@elemon.com.ar

SUCURSAL CÓRDOBA Rufino Cuervo 1085 Local 5 | X5009GAA

Barrio Las Rosas | Córdoba

Tel: (0351) 489 8005 | 8600 | ventascordoba@elemon.com.ar

[www.elemon.net](http://www.elemon.net)

**NOTAS:**

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Se termino de imprimir  
en Enero de 2014  
en Rolta, (4865-7337),  
Ecuador 334, Buenos Aires.  
[www.rolta.com.ar](http://www.rolta.com.ar)