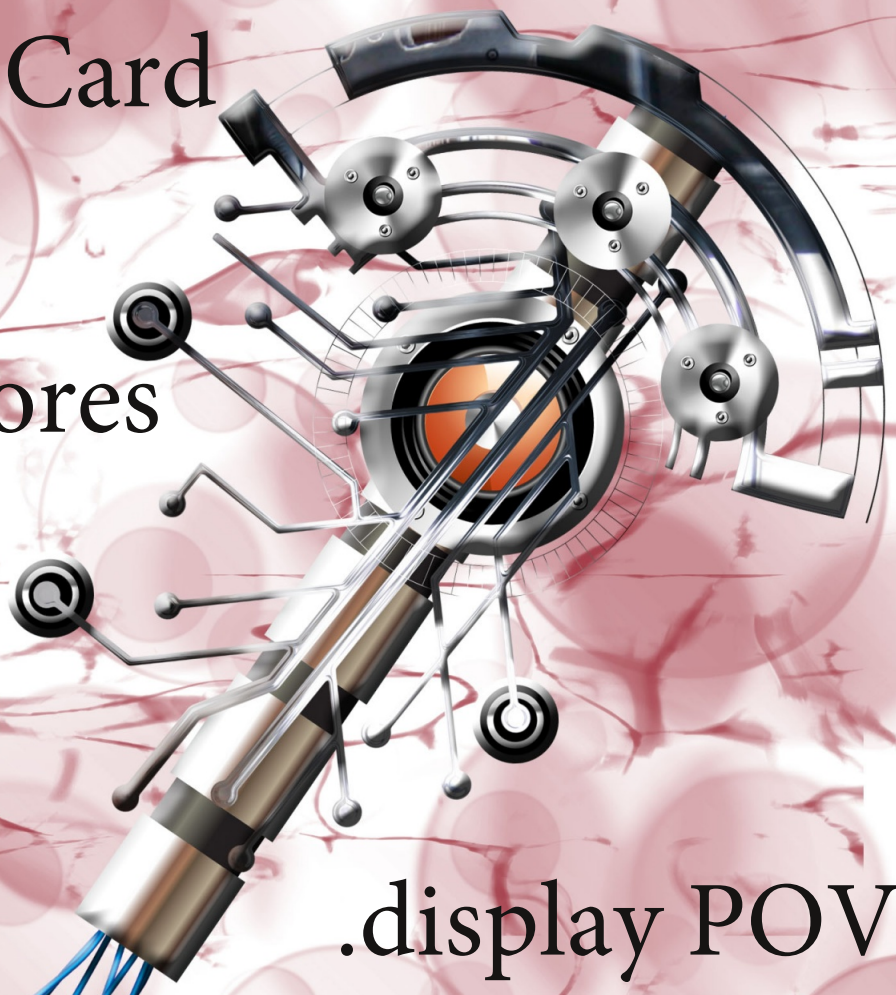
// DESCARGA GRATUITA //

"int año = 3, número 9 / AGOSTO 2010

µCONTROL

Electrónica en General Pics en Particular



.memoria SD Card

.tutorial semiconductores

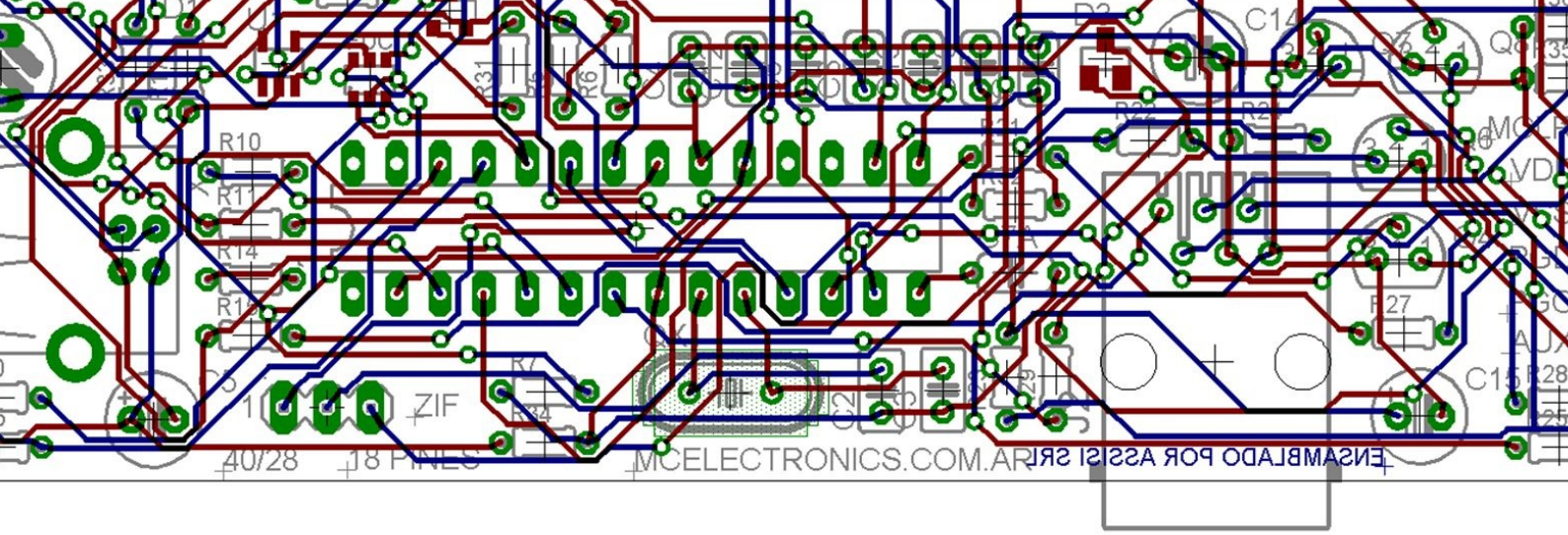
.tutorial MPLAB C18

.ASM desde cero

.display POV

.compuertas lógicas transistorizadas

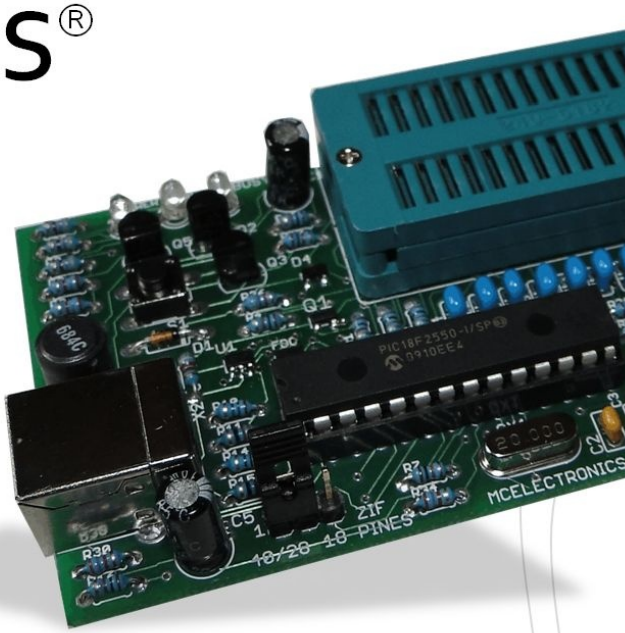
.seminario Microcontroladores



mc electronics®

Programador y Debugger Express

USB para PIC y dsPIC



El MCE PDX USB se puede utilizar como Programadador y Debugger Express. Además es un Analizador lógico de 3 canales. Incluye zócalo ZIF, conector RJ11 y EasyJack de 6 pines compatible con PICKit2 de Microchip.

$ 149,00 +IVA

USD 38,90

Incluye envío sin cargo a todo el país !

MCELECTRONICS
Austria 1760 - OF 8
Ciudad de Buenos Aires (1425)
BA. Argentina.

(011) 6091-4922/4581
www.mcelectronics.com.ar
info@mcelectronics.com.ar

PIN 796948766332890345678046587411

.staff

número = 9; año = 3;

Dirección, Redacción y Corrección:
Ariel Palazzesi
Argentina
arielpalazzesi@gmail.com

Diseño y Diagramación:
xocas y
Lucas M. Treser
lmtreser@gmail.com

Consejo Editorial:
Mario Sacco
service.servisystem@gmail.com

Alejandro Casanova
inf.pic.suky@live.com.ar

Emiliano Safarik
fluf@adinet.com.uy

David (Leon Pic)
david@meteorologiafacil.com.ar

Gabriel Gabarain
gabriel@gelab.com.ar

Germán Reula
gerreula@yahoo.com.ar

Martín Torres Fortelli
torres.electronico@gmail.com

.indice

.editorial

Nuevos colaboradores, nuevas ideas, nuevos proyectos. Este ejemplar de la Revista uControl contiene 62 páginas de material absolutamente sin desperdicio. Continuamos con el tutorial de Torres Fortelli destinado a desvelar todos los secretos de los semiconductores y ponemos a tu disposición una excelente guía para comprender e implementar puertas lógicas utilizando solamente componentes discretos. Si lo tuyo es el ASM, puedes seguir aprendiendo de la mano de David Persi, quien a partir de este número nos guiará en el apasionante camino de la programación de bajo nivel.

Los amantes de los microcontroladores de Microchip apreciarán el trabajo de Alejandro Casanova, quien nos propone una guía paso a paso para comenzar a utilizar el compilador cruzado MPLAB C18, basado en el ANSI C. Y si quieres seguir aprendiendo, que mejor que apuntarte al Segundo Congreso Virtual de Microcontroladores, que tendrá lugar entre el 18 y 29 de octubre del corriente año. German Reula nos cuenta todos los detalles de este evento gratuito, al que puedes apuntarte ya mismo. El año pasado participaron 805 congresistas, provenientes de 21 países repartidos por toda América y Europa, y los organizadores esperan que este año participe aún más gente. ¡No te lo puedes perder!

Uno de los más interesantes montajes de este número pertenece a Emiliano Safarik, y consiste en un un display de los llamados Persistence Of Vision (POV), que aprovecha la característica del ojo humano de seguir viendo una imagen durante una fracción de segundo luego de que esta ha desaparecido. Estamos seguros que querrás montarte uno sin perdida de tiempo.

A la hora de almacenar información, pocos dispositivos son más versátiles que las tarjetas de memoria Flash. ¿Alguna vez intentaste utilizarlas como medio de almacenamiento en tus proyectos? Si aún no lo has hecho, puedes aprovechar el otro trabajo de Alejandro e implementar sin problemas este sistema cada vez que lo necesites.

Esperamos sinceramente que este nuevo ejemplar de la revista sea de tu agrado, y esperamos tu participación en nuestro foro. Necesitamos de tus comentarios para que este proyecto pueda seguir avanzando.

!Hasta el próximo número!

Compuertas Lógicas transistorizadas

Si bien este articulo no trata un tema que sea especialmente novedoso, lo cierto es que siempre resulta interesante conocer la forma en funcionan los bloques constructivos que utilizamos cada día en nuestros proyectos.

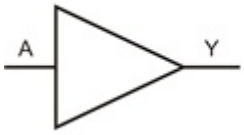
// por: Gabriel Gabarain //
gabriel@gelab.com.ar

A lo largo de las próximas páginas veremos un poco de historia, de los comienzos de la era de los semiconductores. Se trata, ni más ni menos, que de las famosas funciones que utilizamos a diario en los microprocesadores sin siquiera darnos cuenta, pero construidas con transistores. Todos los circuitos han sido probados.

Función IGUALDAD

Es aquella función en cuya salida (Y) existe el mismo valor que en su entrada (A).

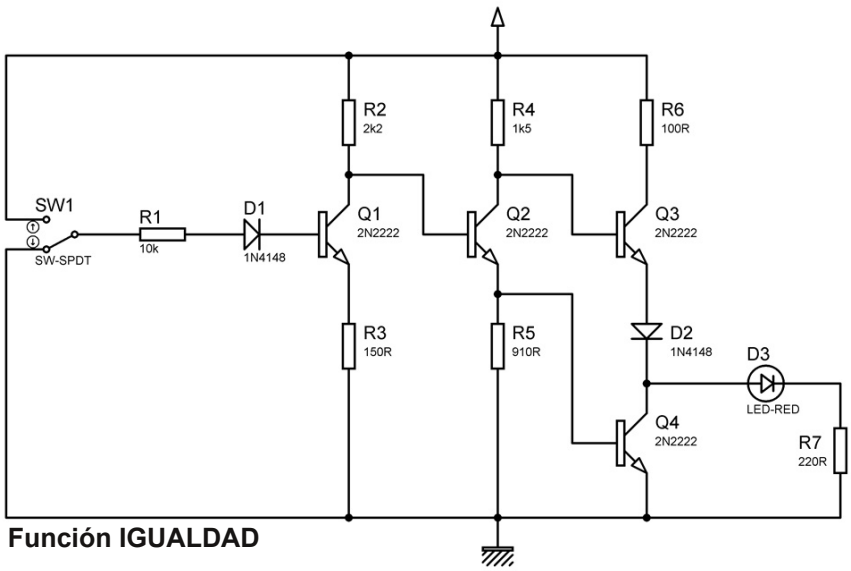
Símbolo y tabla de verdad:



A	Y
0	0
1	1

Si recorremos el diagrama de derecha a izquierda nos encontramos con lo que denominaremos la etapa de salida conformada por R6, Q3, D2 y Q4, cuya misión es la de suministrar un nivel de tensión alto o bajo en la salida (D3 y R7).

Supongamos que la salida esté a un nivel bajo, entonces el colector de Q4 deberá tener un nivel de tensión próximo a los 0V. Para que ello ocurra Q4 deberá estar en condición de saturación. Por el contrario, si la salida estuviese en un estado alto Q3 deberá estar en saturación haciendo que la corriente fluya a través de R6 y D2 respectivamente, de lo anterior se deduce que R6,Q3 y D2 constituyen la etapa de salida para los niveles altos y Q4 para los niveles bajos.



Función IGUALDAD

Un punto a tener en cuenta es que Q3 y Q4 no pueden ser activados simultáneamente, ya que de ese modo se produciría un cortocircuito en la salida, es por eso que se realiza la denominada etapa de excitación conformada por R4,Q2 y R5 para que esto no ocurra.

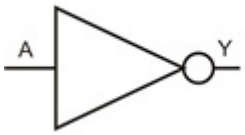
Observemos el circuito. Si no hay corriente de base en Q2 éste se encontrará en corte presentando una gran impedancia entre colector y emisor y haciendo que la corriente fluya a través de R4 hacia la base de Q3 poniéndolo en estado de saturación y, debido a que Q2 no se encuentra conduciendo, la caída de tensión en la resistencia R5 es 0V dejando al corte Q4. Por el contrario, si existiese corriente en la base de Q2 este pasaría al estado de saturación cayendo abruptamente la tensión de base de Q3 que lo llevaría al corte y aumentando del mismo modo la caída de tensión en R5 haciendo que Q4 se sature.

Por último quedaría por ver la etapa de entrada y adaptación compuesta por Q1, D1,R1 y R2. Como se puede apreciar, el funcionamiento con respecto de Q2 es mas que simple. El tema aquí está en la entrada, ya que si no existiese D1 y R3 cualquier tensión que superase los 0,25V sería tomada como nivel alto por Q1. Veamos, la tensión aplicada a la entrada atraviesa R1 y D1 reflejándose en la unión BE de Q1 produciendo caídas de tensión en cada uno de los elementos siendo la más importante aquí la producida en D1. Como no va a circular corriente hasta que el nivel de entrada supere los 0,9V Q1 se mantiene en estado de corte y Q2 en saturación, superada esa tensión se invierte el proceso recién cuando Q1 alcance los 1,2V aproximadamente (0,9V + 0.25V).

Hasta aquí les parece interesante?... sigamos entonces con la:

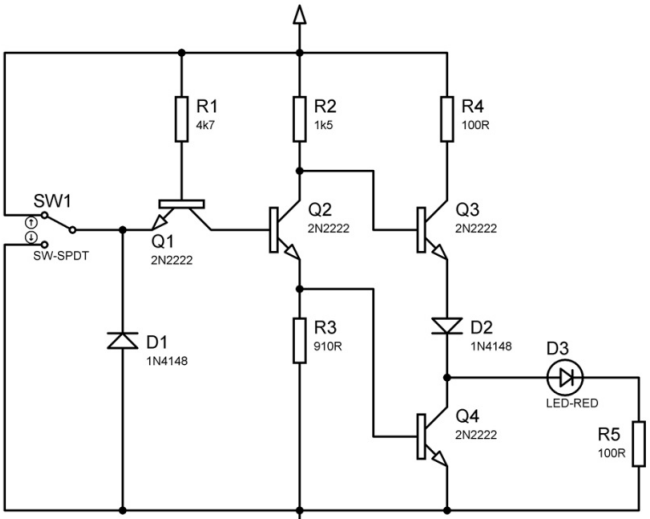
Función INVERSIÓN

Símbolo y tabla de verdad:



A	Y
0	1
1	0

Como podemos apreciar, tanto la etapa de salida (R4, Q3, D2, Q4) como la de excitación (R2, Q2, R3) son iguales al ejemplo anterior así que solo nos vamos a concentrar en la etapa de entrada y adaptación formada por D1, Q1 y R1.



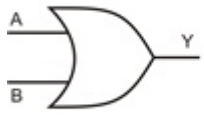
Función INVERSIÓN

A simple vista notamos que la disposición de Q1 cambió, la entrada de señal es ahora a través de su emisor; de esta forma circulará a través de R1 una corriente que será la suma de la corriente circulante por la unión BE de Q1 más la circulante por la unión BC de ese mismo transistor y que circula a su vez por la unión BE de Q2.

Ante la presencia de una señal lógica baja, la unión BE de Q1 quedará polarizada de manera directa haciendo que la circulación de corriente lo ponga en estado de saturación; como la tensión CE de Q1 será muy pequeña en este estado, Q2 estará al corte. Si por el contrario la señal de entrada posee un nivel lógico alto circulará corriente por la unión BC de Q1 haciendo que Q2 entre en estado de saturación. El diodo D1 está colocado para proteger la entrada de Q1 ante un eventual valor negativo de tensión; llegado el caso la entrada vería solamente el valor de tensión directa del diodo D1.

Función OR

Símbolo y tabla de verdad:

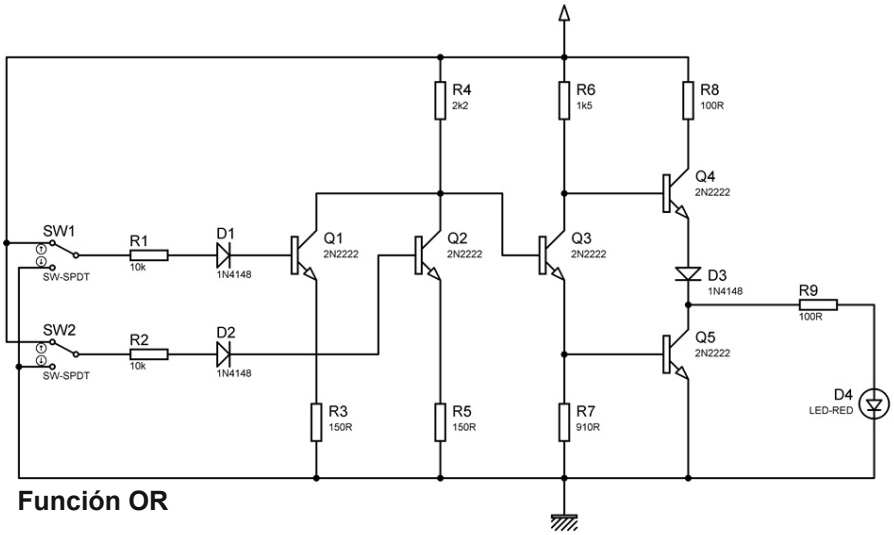


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Nuevamente, no hay modificaciones en las etapas de salida y excitación pero como esta función posee dos entradas (A y B) la etapa correspondiente se duplica; una entrada (A) esta constituida por R1, D1, Q1 y R3 y la otra (B) por R2, D2, Q2 y R5 quedando R4 que es común a ambas entradas.

Si la entrada A está a nivel alto circulará corriente por R1, D1, la unión BE de Q1 y R3 haciendo que Q3 quede al corte, lo mismo sucederá si la entrada B está en alto con la diferencia obvia que la corriente circulará esta vez por R2, D2, la unión BE de Q2 y R5.

En caso que ambas entradas estén a nivel alto Q1 y Q2 entrarían en saturación simultáneamente manteniendo el estado de corte de Q3, en el caso contrario (ambas entradas a nivel bajo) Q1 y Q2 entrarían en corte por no circular corriente en la unión BE de ambos lo cual hará que la tensión de colector sature a Q3.

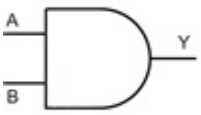


Función OR

Resumiendo podemos decir que:
Cuando A y B = 0 Q1 está en corte, Q2 está en corte, Q3 en saturación y la salida = 0
Cuando A = 0 y B = 1 Q1 está en corte, Q2 en saturación, Q3 en corte y la salida = 1
Cuando A = 1 y B = 0 Q1 está en saturación, Q2 en corte, Q3 en corte y la salida = 1
Cuando A y B = 1 Q1 está en saturación, Q2 está en saturación, Q3 en corte y la salida = 1

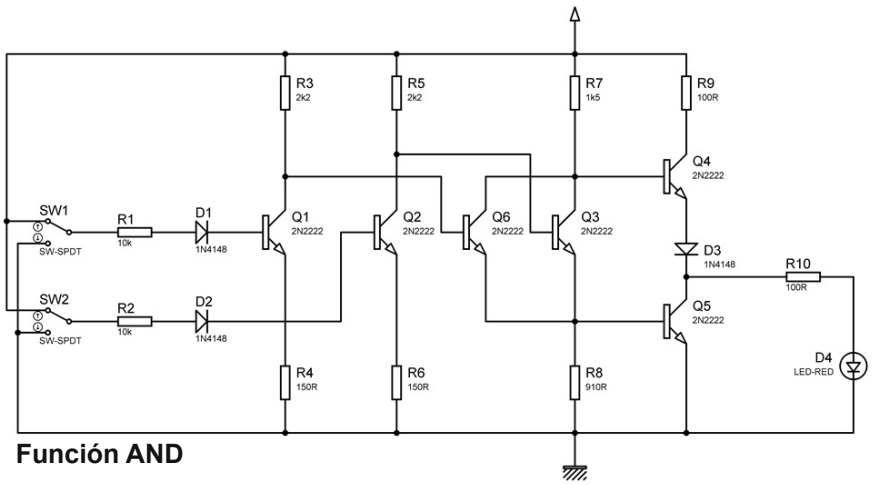
Función AND

Símbolo y tabla de verdad:



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Tal como ocurre en los casos anteriores, la etapa de salida no difiere, aunque si lo hace la etapa de excitación, que es doble (Q6 y Q3). Para que a la salida exista un nivel alto, tanto Q6 como Q3 deberán estar al corte. Si cualquiera de ellos entra en estado de saturación la salida pasará a nivel bajo, o sea que aquí es donde se produce la función AND.



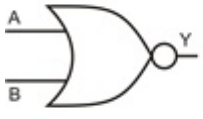
Función AND

Con respecto a las entradas ambas se comportan del mismo modo que lo explicado en la función OR e IGUALDAD, con la diferencia que las salidas por colector van a etapas independientes de excitación (Q1-Q6 y Q2-Q3) por lo tanto creo que no hay mayores problemas en entender el funcionamiento. Solo diremos que la entrada A estará compuesta por R1, D1, Q1, R3 y R4 y la entrada B por R2, D2, Q2, R5, y R6.

Resumiendo:
Si A = 0 y B = 0 Q1, Q2, D1, D2 y Q4 no conducen; Q6,Q3 y Q5 están en saturación y la salida = 0
Si A = 0 y B = 1 D1, Q1, Q3 y Q4 no conducen; D2, Q2, Q6 y Q5 si conducen y la salida = 0
Si A = 1 y B = 0 D2, Q2, Q6 y Q4 no conducen; D1, Q1, Q3 y Q5 si conducen y la salida = 0
Si A = 1 y B = 1 Q6, Q3 y Q5 no conducen; D1, D2, Q1, Q2, y Q4 si conducen y la salida = 1

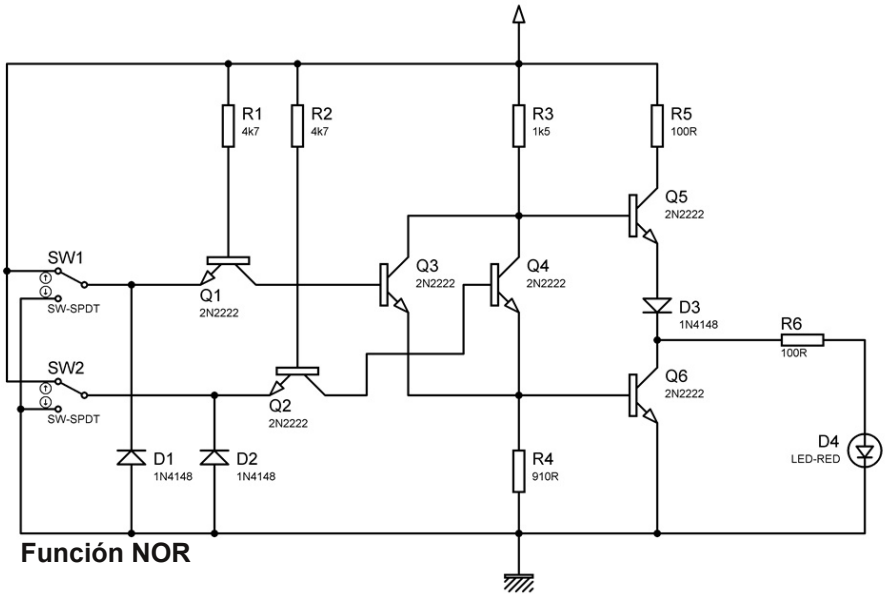
Función NOR

Símbolo y tabla de verdad:



A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Aquí las etapas de salida y de excitación son iguales a las mostradas en el circuito anterior. Las etapas de entrada son iguales a la explicada en el circuito de la función INVERSIÓN con la diferencia que aquí se duplican: la entrada A estará compuesta por Q1, D1 y R1 y la entrada B por Q2, D2 y R2.

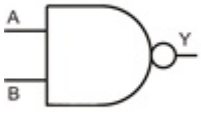


Función NOR

Resumen:
Si A = 0 y B = 0 BE Q1 en conducción, BC Q1 en corte BE Q2 en conducción, BC Q2 en corte Q3, Q4 y Q6 en corte; Q5 en conducción Salida = 1
Si A = 0 y B = 1 BE Q1 en conducción, BC Q1 en corte BE Q2 apenas conduce, BC Q2 en conducción Q3 y Q5 en corte; Q4 y Q6 en conducción Salida = 0
Si A = 1 y B = 0 BE Q1 apenas conduce, BC Q1 en conducción BE Q2 en conducción, BC Q2 en corte Q4 y Q5 en corte; Q3 y Q6 en conducción Salida = 0
Si A = 1 y B = 1 BE Q1 apenas conduce, BC Q1 en conducción BE Q2 apenas conduce, BC Q2 en conducción Q3, Q4 y Q6 en conducción; Q5 en corte Salida = 0

Función NAND

Símbolo y tabla de verdad:

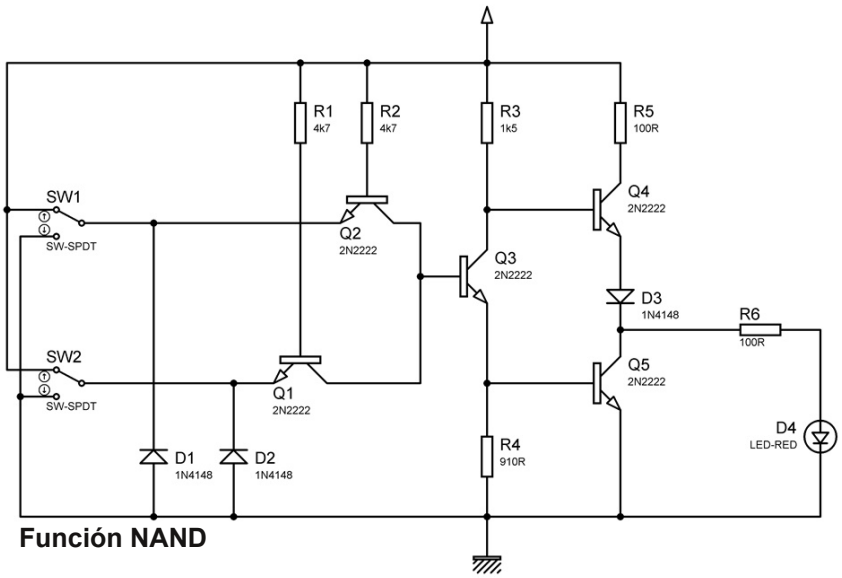


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A esta altura del artículo, el circuito no debería presentar mayores problemas de interpretación.

La etapa de salida ya es conocida, la etapa de excitación es la misma que utilizamos en las funciones OR, INVERSORA e IGUALDAD y la etapa de entrada en la función INVERSORA y NOR y solo difiere de la anterior en que los colectores de ambas entradas van unidos a un único punto, la base de Q3 debido a que la etapa de excitación es simple y no doble.

Resumiendo:
Si A = 0 y B = 0 BE Q2 en conducción, BC Q2 en corte BE Q1 en conducción, BC Q1 en corte Q3 y Q5 en corte; Q4 y D3 en conducción Salida = 1
Si A = 0 y B = 1 BE Q2 en conducción, BC Q2 en corte BE Q1 apenas conduce, BC Q1 en conducción Q3 y Q5 en corte; Q4 y D3 en conducción Salida = 1
Si A = 1 y B = 0 BE Q2 apenas conduce, BC Q2 en conducción BE Q1 en conducción, BC Q1 en corte Q3 y Q5 en corte; Q4 y D3 en conducción Salida = 1
Si A = 1 y B = 1 BE Q2 apenas conduce, BC Q2 en conducción BE Q1 apenas conduce, BC Q1 en conducción Q3 y Q5 en conducción; Q4 y D3 en corte Salida = 0

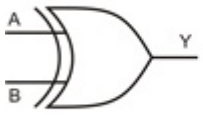


Función NAND

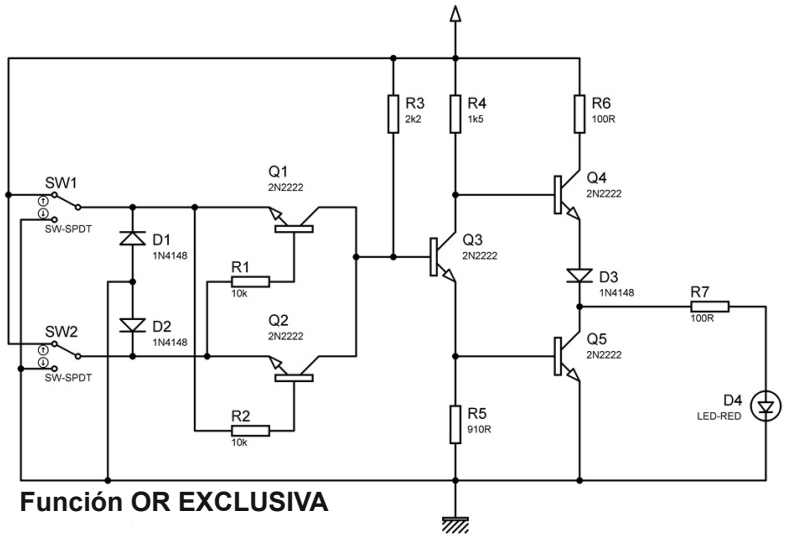
Si leyeron con atención el artículo y comprendieron los conceptos de las funciones anteriores, la función OR EXCLUSIVA no es necesario que la explique porque pueden deducirla solos.

Función OR EXCLUSIVA

Símbolo y tabla de verdad:



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



Función OR EXCLUSIVA

Saludos y Gracias por leer este humilde aporte.

¿Buscás LEDs?

D+
LED

Todos los LEDs, un solo lugar.

www.dled.com.ar

Soler 4949 Timbre "1" | Capital Federal | Argentina | Teléfono +54 11 4770 9366 y líneas rotativas.

Tutorial MPLAB C18

Se presenta un tutorial de C del compilador C18, en donde la idea es ir adquiriendo conceptos a medida que los utilicemos en el desarrollo de los ejemplos, de esta manera lo que se presenta teóricamente lo asociamos inmediatamente con la práctica. Aunque claro está que el lenguaje es muy amplio y no se pueden hacer ejemplos de cada uno de los conceptos. En esta primera entrega presentamos como configurar el entorno de trabajo y una primera introducción al lenguaje.

// por: Alejandro Casanova //
inf.pic.suky@live.com.ar

Entorno de programación

MPLAB C18 es un compilador cruzado que se corre en un PC y produce código que puede ser ejecutado por la familia de microcontroladores de Microchip PIC18XXXX. Al igual que un ensamblador, el compilador traduce las declaraciones humanas en unos y ceros para ser ejecutados por el microcontrolador. Sigue la norma ANSI C, salvo en particularidades de los microcontroladores y contiene librerías para comunicaciones SPI, I2C, UART, USART, generación PWM, cadena de caracteres y funciones matemáticas de coma flotante.

La interfaz gráfica del usuario es MPLAB IDE, que sirve como un único entorno para escribir, compilar y depurar código para aplicaciones embebidas. Permite manejar la mayoría de los detalles del compilador, ensamblador y enlazador, quedando la tarea de escribir y depurar la aplicaciòn como foco principal del programador (usuario)

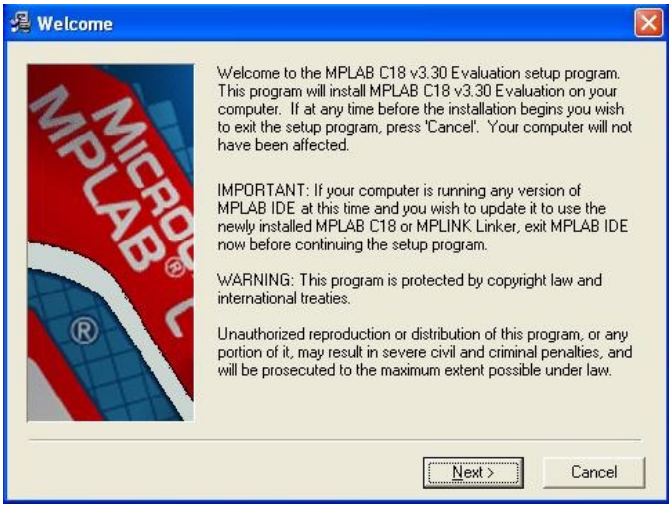
MPLAB IDE es gratuito y de MPLAB C18 se puede descargar una versión demo directamente de www.microchip.com

Instalación

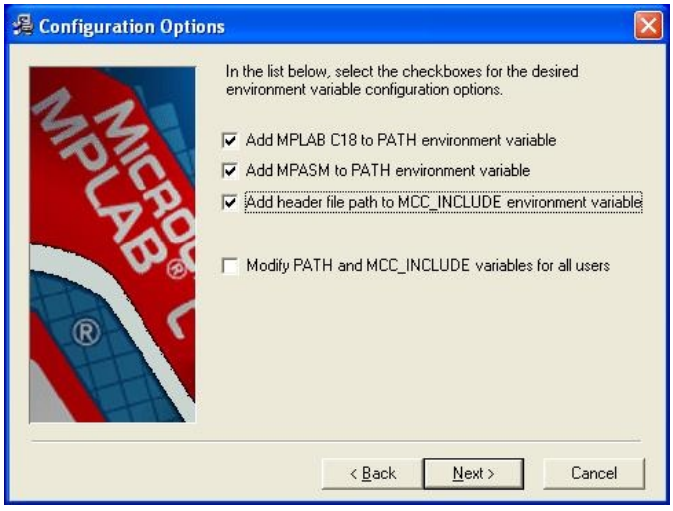
El programa se debe bajar directamente desde Microchip. Hay disponible una versión gratuita para estudiantes que es un demo de 60 días. Para poder descargarlo es necesario registrarse.

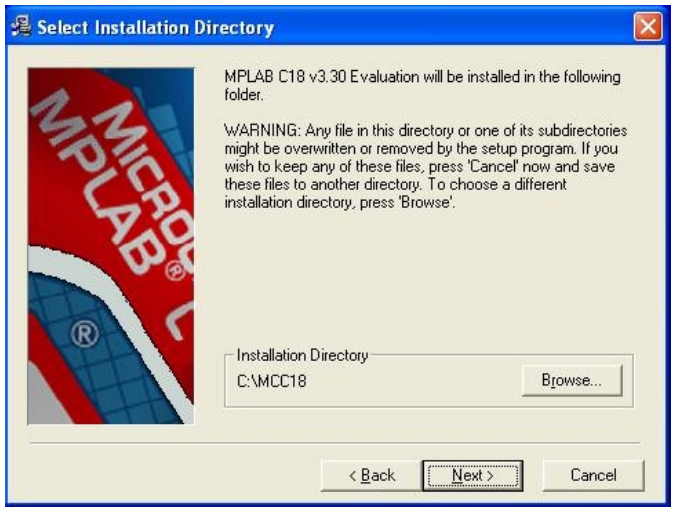
Una vez descargado, ejecutar el instalador MPLAB-C18-Evaluation-v3_30, versión actualmente disponible.

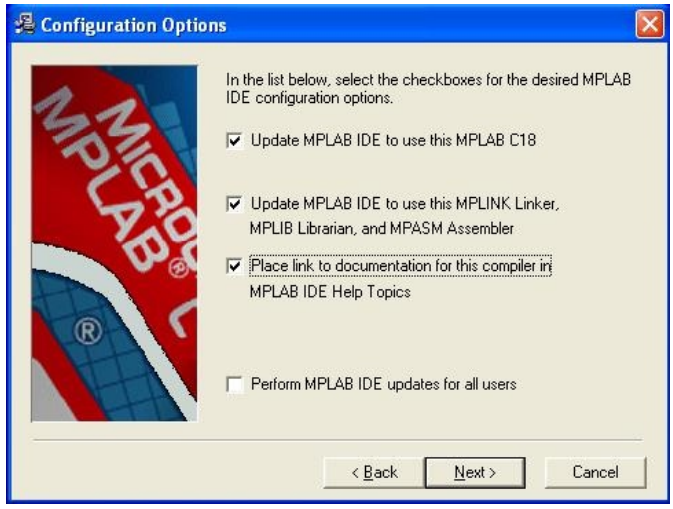
Para la instalación seguimos los siguientes pasos:

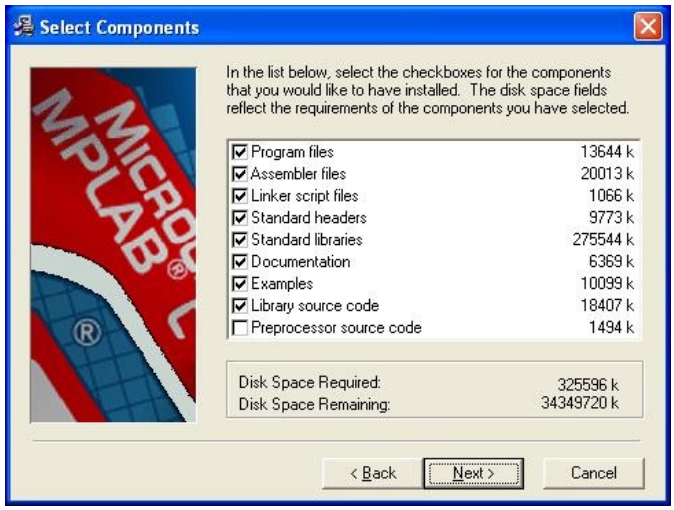


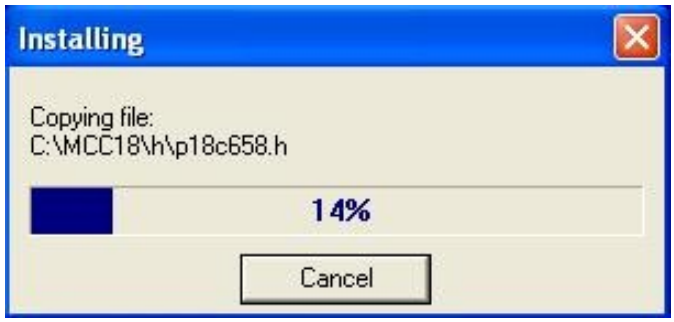

License Agreement
MICROCHIP
IMPORTANT
YOU MUST ACCEPT THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT TO RECEIVE A LICENSE FOR THE ACCOMPANYING SOFTWARE. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" OR OPEN THIS PACKAGE AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," OR DO NOT OPEN THIS PACKAGE, DOWNLOAD, OR INSTALL THIS SOFTWARE.
MPLAB(R) C18 COMPILER LICENSE
I Accept
I Do Not Accept
< Back
Next >
Cancel

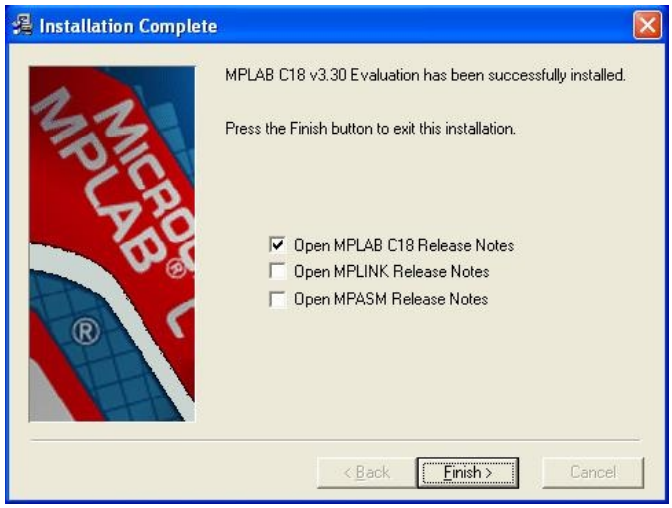

Configuration Options
In the list below, select the checkboxes for the desired environment variable configuration options.
Add MPLAB C18 to PATH environment variable
Add MPASM to PATH environment variable
Add header file path to MCC_INCLUDE environment variable
Modify PATH and MCC_INCLUDE variables for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


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


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

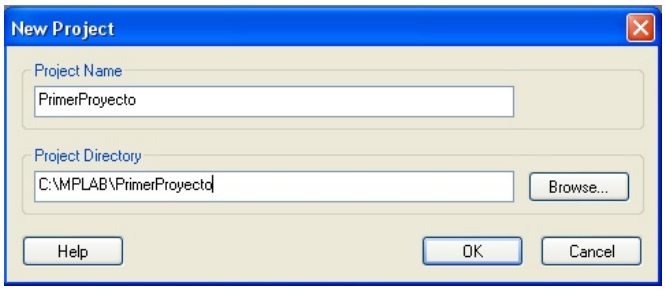

Select Components
In the list below, select the checkboxes for the components that you would like to have installed. The disk space fields reflect the requirements of the components you have selected.
Program files 13644 k
Assembler files 20013 k
Linker script files 1066 k
Standard headers 9773 k
Standard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


Installing
Copying file:
C:\MCC18\h\p18c658.h
14%
Cancel

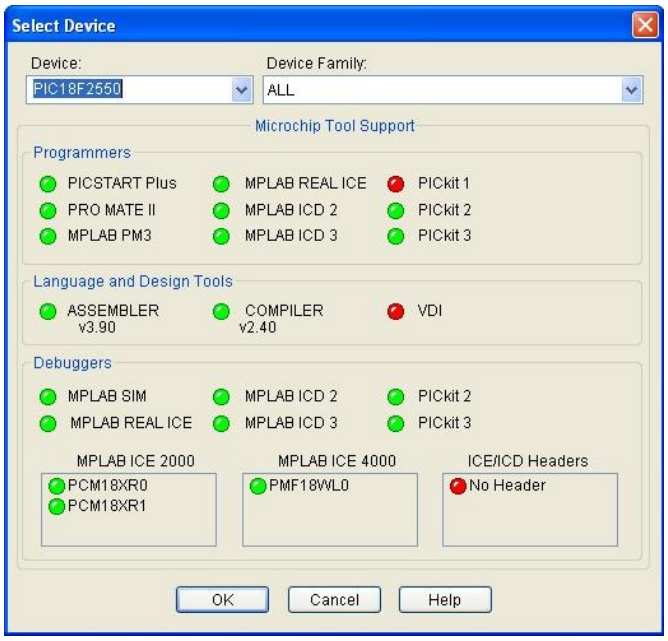

Installation Complete
MPLAB C18 v3.30 Evaluation has been successfully installed.
Press the Finish button to exit this installation.
Open MPLAB C18 Release Notes
Open MPLINK Release Notes
Open MPASM Release Notes
< Back
Finish >
Cancel

Creación de un nuevo proyecto

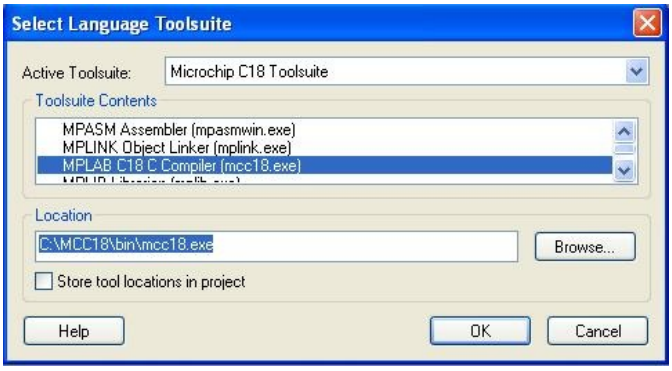
Projec/New: Nos aparecerá una pantalla donde indicamos el nombre de nuestro proyecto y la carpeta donde será guardado.



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



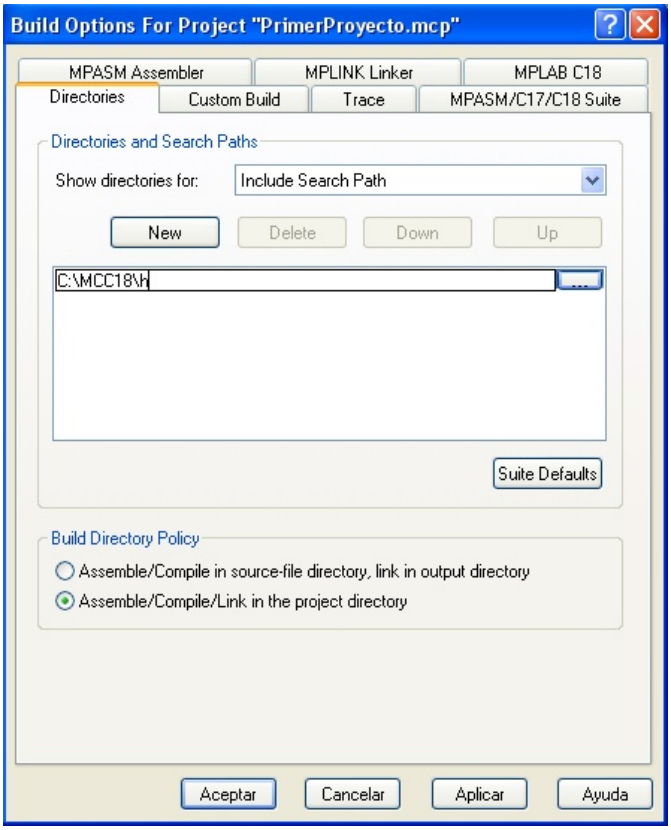
Seleccionamos el compilador: **Project/ Select Lenguaje Toolsuite** y nos aseguramos que todas las direcciones son correctas.

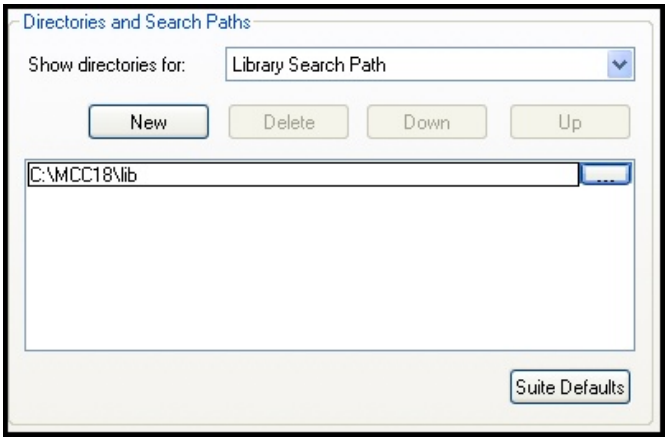


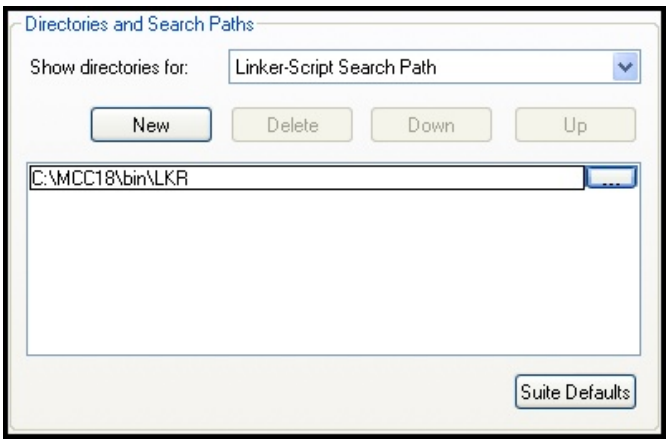
Configuramos los subdirectorios de trabajo: **Project/Build options/Project**

Seleccionamos la ubicación de ficheros de declaraciones, bibliotecas y script de enlazado.

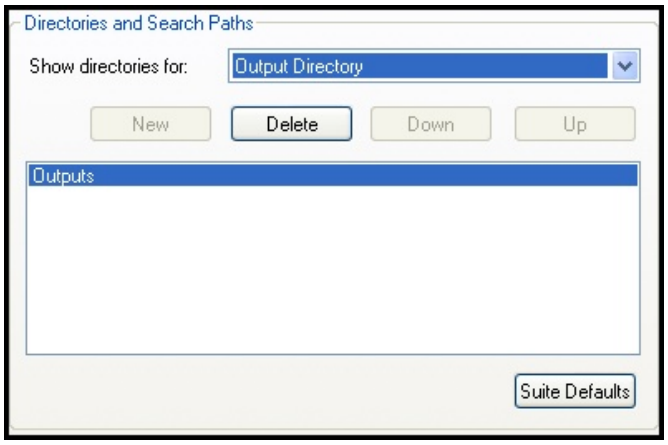
Show directories for:
- Include Search Path
- Library Search Path
- Linker-Script Search Path

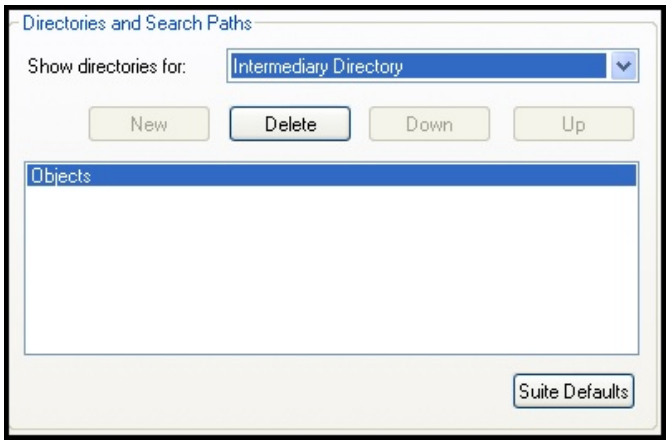




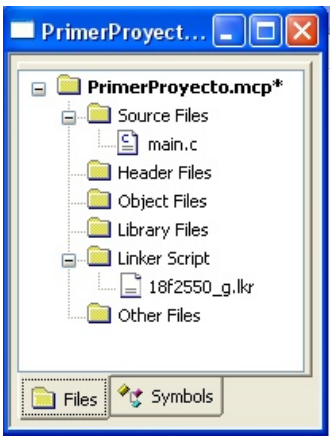


También podemos crear carpetas como Output y Objects para organizar en ellos los archivos intemedios y finales del proceso de compilación.

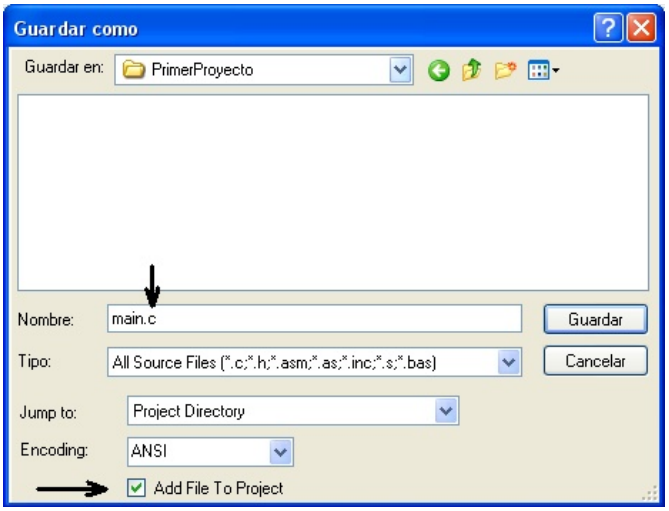




Nota: Según la versión también se debe agregar al proyecto el archivo (PIC18F utilizado).lkr ubicado en MCC18/lkr, si no produce error de compilación.



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



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

Introducción al C18

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

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

Creando el código

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

La librería que siempre incluiremos en el archivo principal será la del PIC a usar:

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

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

La sintaxis seria:
#pragma config Nombre_del_fuse=estado

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

Ahora viene el código de nuestro programa:

```
main{

}
```

El proceso de compilación

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

Ejemplo de estructura de un archivo fuente principal.

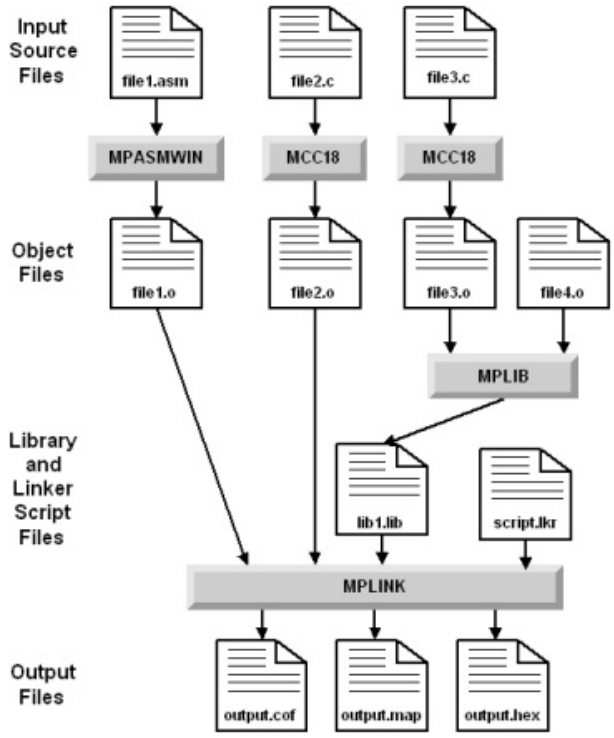
```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador  ** */
#pragma config FOSC=XT_XT, FCMEN=OFF, IESO=OFF, CPUDIV=OSC1_PLL2
#pragma config PWRT=ON, BOR=OFF, BORV=0, WDT=OFF, WDTPS=32768
#pragma config MCLRE=ON, LPT1OSC=OFF, PBADEN=OFF, CCP2MX=OFF
#pragma config STVREN=OFF, LVP=OFF, XINST=OFF, DEBUG=OFF
#pragma config CP0=OFF, CP1=OFF, CP2=OFF, CPB=OFF, CPD=OFF
#pragma config WRT0=OFF, WRT1=OFF, WRT2=OFF
#pragma config WRTB=OFF, WRTC=OFF, WRTD=OFF
#pragma config EBTR0=OFF, EBTR1=OFF, EBTR2=OFF, EBTRB=OFF

void main(void){

   // Sentencias.-
}
```

Flujo de la generación del hex

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



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

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

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

Operadores

Aquí definiremos todos los operadores utilizados por C18.

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

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

Operadores aritméticos:
Se utilizan para realizar cálculos matemáticos:

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

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

Operador	Descripción
&&	AND
||	OR
!	NOT

Operadores bitwise:
Son para modificar los bits de una variable:

Operador	Descripción
&	And
|	Or
^	Xor
~	complemento
<<	rotar izquierda
>>	rotar derecha

Estructuras

Estructura if:

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

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

// Es PORTA igual a cero(0)?
if(PORTA==0x00){
    LATB=0xFF; // Si
}
```

Estructura if-else:

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

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

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

Estructura while:

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

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

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

Estructura do-while:

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

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

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

Estructura For:

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

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

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

Estructura switch:

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

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

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

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

Accediendo a los bits de un registro

Para acceder individualmente a los bits de un registro se escribe la siguiente sentencia: **Registrobits.bit**

```
PORTBbits.RB0
LATAbits.LATA3
SSP1CON2bits.ACKDT
```

Variables

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

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

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

Modificadores de las variables:

MPLAB C18 utiliza los modificadores establecidos por ANSI:

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

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

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

Const: el contenido de la variable es fijo.

Volatile: el contenido de la variable puede cambiar.

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

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

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

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

Especificación de banco de memoria de datos:

Far: la variable puede ir en cualquier banco.
Near: la variable tiene que estar en el banco de acceso.

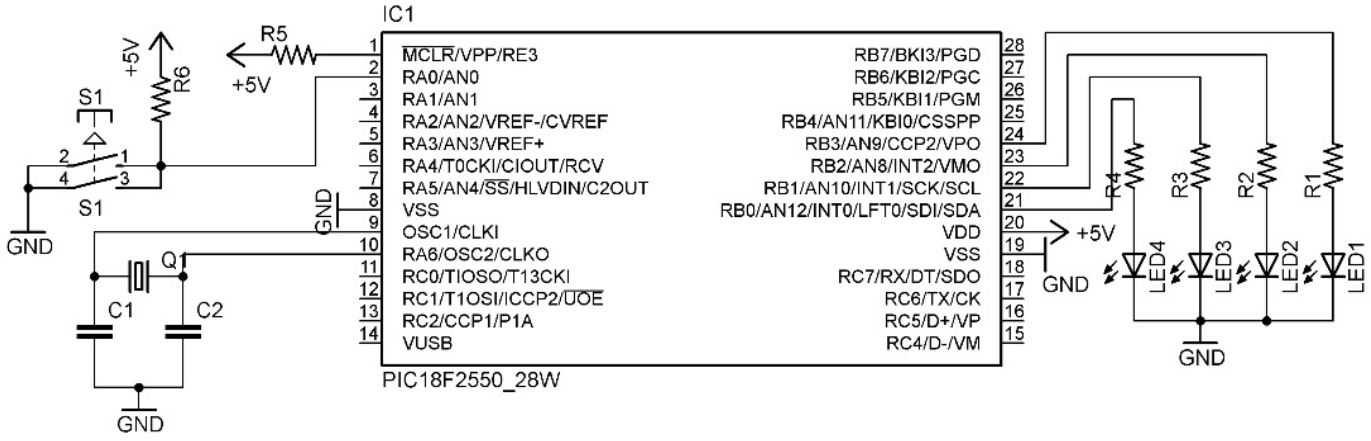
Ejemplo 1:

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

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador  ** */
#pragma config FOSC=XT_XT, FCMEN=OFF, IESO=OFF, CPUDIV=OSC1_PLL2
#pragma config PWRT=ON, BOR=OFF, BORV=0, WDT=OFF, WDTPS=32768
#pragma config MCLRE=ON, LPT1OSC=OFF, PBADEN=OFF, CCP2MX=OFF
#pragma config STVREN=OFF, LVP=OFF, XINST=OFF, DEBUG=OFF
#pragma config CP0=OFF, CP1=OFF, CP2=OFF, CPB=OFF, CPD=OFF
#pragma config WRT0=OFF, WRT1=OFF, WRT2=OFF
#pragma config WRTB=OFF, WRTC=OFF, WRTD=OFF
#pragma config EBTR0=OFF, EBTR1=OFF, EBTR2=OFF, EBTRB=OFF

void main(void){
	ADCON1=0x0F;		/* Todos entradas/salidas digitales */
	TRISA=0xFF;		/* Todos como entradas */
	TRISB=0xF0;		/* Nible bajo como salida */
	LATB=0x00;		/* Todos los leds apagados */
	while(1){		/* Bucle infinito */
		if(PORTAbits.RA0==1){	/* Se testea estado del pulsador */
			LATB=0x00;	/* Si esta en 1 logico apagamos leds */
		}else{
			LATB=0x0F;	/* Sino encedemos todos los leds */
		}
	}
}
```

Hardware:



Ejemplo de ubicación de variables

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador ** */
#pragma config FOSC=XT_XT, FCMEN=OFF, IESO=OFF, CPUDIV=OSC1_PLL2
#pragma config PWRT=ON, BOR=OFF, BORV=0, WDT=OFF, WDTPS=32768
#pragma config MCLRE=ON, LPT1OSC=OFF, PBADEN=OFF, CCP2MX=OFF
#pragma config STVREN=OFF, LVP=OFF, XINST=OFF, DEBUG=OFF
// Variables globales.-
unsigned char k;   // Variable ubicada en memoria RAM.
const rom long p;  //Variable ubicada en memoria de programa, constante.-
float Temp;

void main(void){
   // Variables locales:
   unsigned int r,s;
   // Sentencias.-
}
```

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

```
ram char k;
near rom char r;
far rom char s;
```

```
   08A        k          0x00
P  00FA       r          0x00
P  00FB       s          0x00
```

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

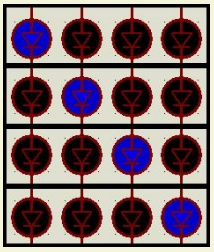
Demoras

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

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

Donde i puede tomar valores entre 0 y 255.

Ejemplo 2: LEDS secuenciales.



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

Hardware: Idem anterior.

Código:

```
/* ** Archivo con definicion de registros y bits del microcontrolador elegido */
#include <p18f2550.h>
/* ** Configuracion de los Fuses del microcontrolador  ** */
#pragma config FOSC=XT_XT, FCMEN=OFF, IESO=OFF, CPUDIV=OSC1_PLL2
#pragma config PWRT=ON, BOR=OFF, BORV=0, WDT=OFF, WDTPS=32768
#pragma config MCLRE=ON, LPT1OSC=OFF, PBADEN=OFF, CCP2MX=OFF
#pragma config STVREN=OFF, LVP=OFF, XINST=OFF, DEBUG=OFF
#pragma config CP0=OFF, CP1=OFF, CP2=OFF, CPB=OFF, CPD=OFF
#pragma config WRT0=OFF, WRT1=OFF, WRT2=OFF
#pragma config WRTB=OFF, WRTC=OFF, WRTD=OFF
#pragma config EBTR0=OFF, EBTR1=OFF, EBTR2=OFF, EBTRB=OFF

unsigned char k;    /* Variable utilizada para realizar efecto */

void main(void){
      ADCON1=0x0F;       /* Todos entradas/salidas digitales */
      TRISA=0xFF;        /* Todos como entradas */
      TRISB=0xF0;        /* Nible bajo como salida */
      LATB=0x00;         /* Todos los leds apagados */
      while(1){          /* Bucle infinito */
            if(PORTAbits.RA0==1){      /* Se testea estado del pulsador */
                  LATB=0x00;           /* Si esta en 1 logico apagamos leds */
            }else{
                  LATB=0x01;                  /* Encedemos primer Led */
                  for(k=1;k<=4;k++){          /* Rota 4 veces */
                        Delay10KTCYx(30);     /* Demora 300ms */
                        LATB=(PORTB<<1);      /* Rotamos Led encendido */
                  }
            }
      }
}
```


Toda la Television
el Audio y el Video
en un solo sitio web
SERVISYSTEM
Tutoriales, circuitos
reparaciones, software
tips, samples, consejos,
foros, blog, PICs ...
www.servisystem.com.ar

Para poder participar de un congreso, no siempre es necesario comprar pasajes de avión, viajar durante horas ni alojarse en costosos hoteles. Tampoco son indispensables los micrófonos, cañones proyectores ni lujosos auditorios. En muchos casos, como en el Segundo Congreso Virtual de Microcontroladores, solo es necesario valerse de la tecnología y las herramientas que esta nos pone a disposición.

// por: German Reula //
gerreula@yahoo.com.ar

El Congreso

"Lo que pretendemos, principalmente, al organizar un Congreso Virtual es que cualquiera pueda participar. Con esta metodología, eliminamos las restricciones de tiempo y dinero que implican el tener que trasladarse y muchas veces son factores condicionantes a la hora de decidir la participación en un evento presencial. De hecho, en el Primer Congreso Virtual de Microcontroladores, realizado en septiembre de 2009, contamos con la participación de congresistas de 21 nacionalidades."

"Intentamos a través de esta propuesta, romper las barreras geográficas y volver a unir en este segundo congreso a desarrolladores de cualquier región, reabriendo canales de participación para la comunicación de las numerosísimas experiencias realizadas en las distintas instituciones educativas, empresas y por desarrolladores particulares"

La iniciativa, gestada desde las Cátedras de "Técnicas Digitales" de la Universidad Tecnológica Nacional, Facultad Regional Paraná (UTN-FRP) surgió en 2009, año en el que se llevo a cabo el Primer Congreso Virtual de Microcontroladores y sus Aplicaciones.

"Cuando nos planteamos la idea de organizar el Primer Congreso, nunca imaginamos que tendría la repercusión que finalmente tuvo. Contamos con la participación de 805 congresistas, provenientes de 21 países repartidos por toda América y Europa. Esto nos enorgullece y es lo que nos impulsó a la organización de una segunda edición para este año."

El Primer Congreso en Números

El Primer Congreso Virtual de Micro-controladores, contó con la participación de más de ochocientos congresistas.

Estudiantes, profesores, técnicos, ingenieros, hobbystas y empresarios de España y toda Latinoamérica intercambiaron experiencias en lo que fue la primera edición de este Congreso.

La elección de la metodología virtual nos permitió extender el congreso a 21 países. Participaron congresistas de Argentina, Australia, Bolivia, Brasil, Chile, Colombia, Costa Rica, Cuba, Ecuador, El Salvador, España, Estados Unidos, Guatemala, México, Nicaragua, Panamá, Perú, República Dominicana, Uruguay y Venezuela.

Se presentaron cuarenta y siete ponencias, distribuidas en las seis áreas temáticas

propuestas. Comunicaciones, Educación, Hogar, Industria, Medio Ambiente y Transporte fueron los ejes temáticos del Primer Congreso Virtual de Microcontroladores.

Se dictaron 12 seminarios web de acceso libre para todos los participantes del congreso. Conceptos de programación, desarrollo de aplicaciones con herramientas gráficas, diseño y simulación de circuitos con microcontroladores, control de motores y otros temas fueron desarrollados en estos seminarios.

Metodologia

El 18 de octubre, a las 19:00 Hs (horario de Argentina), se realizará la apertura del congreso. Los inscriptos tendrán acceso a las publicaciones y quedarán abiertos los foros de debate, uno por cada línea temática.

Cada trabajo presentado tendrá un tópico específico en los foros de debate. Los autores de las ponencias responderán a las consultas que le realicen los participantes. Eso es un compromiso que deben asumir todos los ponentes. Todos los asistentes y ponentes, tendrán la posibilidad de participar de todos los espacios de debates así creados.

El 30 de Octubre se producirá el cierre del congreso. Se entregarán certificados a los asistentes, y a los ponentes.

Ponencias Virtuales

Como en todos los congresos, existe un plazo para la presentación de trabajos. Los participantes, autores y congresistas, deben preinscribirse al congreso y una vez registrados podrán enviar sus ponencias, teniendo como fecha límite para el envío, el 31 de Agosto. La organización evaluará las mismas y comunicara a los autores sobre su publicación.

Para esta segunda edición, los organizadores proponen enmarcar los trabajos en diez áreas temáticas, Bioelectrónica, Comunicaciones/Conectividad, Educación, Hogar/Entretenimiento, Industria/Potencia, Medioambiente, Robótica, Seguridad, Software y Transporte.

Quienes pueden participar

Para participar solo es necesario contar con acceso a internet, por lo tanto, cualquier persona, desde cualquier país o región, puede participar de este evento. A poco menos de cuatro meses del inicio del Congreso, el número de inscritos supera los seiscientos, entre los cuales hay ingenieros, técnicos, investigadores, docentes, estudiantes, empresarios y desarrolladores independientes de 23 países de América y Europa. Se espera que para la apertura del congreso el número de inscriptos supere los mil.

La participación puede ser en calidad de Asistente o de Ponente y para ello deberán realizar su inscripción antes del 10 de Octubre.

Seminarios Web

Para esta segunda edición, los organizadores han confirmado la realización de más de treinta seminarios web de capacitación. Estos seminarios de alto contenido técnico, serán totalmente gratuitos y podrán participar de ellos todos los inscriptos al congreso.

Seminarios:
• **Comunicación de aplicaciones mediante 802.15.4**. Presentación de módulos XBee 802.15.4
• **Comunicación de aplicaciones mediante ZigBee**. Presentación de módulos XBee ZB
• **Módulos XBee 802.15.4 y XBee ZB**: Configuración y utilización de los modulos. Ing. Sergio Caprile - Cika
• **Memorias FRAM y Processor Companions**. Ing. Sergio Caprile - Cika
• **VRS51L3xxx: core 8051 single-cycle de 40** Ing. Sergio Caprile - Cika
• **Módulos Rabbit para networking con TCP/IP sobre Ethernet y Wi-Fi**. Ing. Sergio Caprile - Continea
• **Módulos Digi ARM Cortex-A8 con Ethernet y Wi-Fi para desarrollo sobre Windows CE**. Ing. Sergio Caprile - Continea
• **Módulos Digi ARM9 con Ethernet y Wi-Fi para desarrollo sobre Net+OS, el RTOS de Digi basado en ThreadX**. Ing. Sergio Caprile - Continea
• **ARMando el rompecabezas de 32-bits**: arquitecturas ARM para microcontroladores y su jerga. Ing. Sergio Caprile - LDIR
• **DEMO_S08JM - El kit para trabajar con USB en pocos pasos**. Ing. Roberto Simone - EDUDEVICES
• **CORTEX - Una revolución en el mercado de los microcontroladores, desde M0 a M3**. Ing. Rafael Charro / Ing. Guillermo Jaquenod - ELKO/ARROW
• **MSP430 de Texas Instruments**. Ing. Rafael Charro / Ing. Guillermo Jaquenod - ELKO/ARROW
• **Microcontroladores de 32 bits ARM... O como no temerle al cambio!!** Ing. Marcelo Romeo - EDUDEVICES
• **Uso de las librerías graficas de Microchip Technology**. Mauricio Jancic FSE - Field Sales Engineer Artimar Ltda
• **Diseño de aplicaciones de ultra bajo consumo XLP con Microcontroladores PIC**. Mauricio Jancic FSE - Field Sales Engineer Artimar Ltda
• **Introducción a los PIC32**. Mauricio Jancic FSE - Field Sales Engineer Artimar Ltda
• **mTouch, sensado a través de metales**. Mauricio Jancic FSE - Field Sales Engineer Artimar Ltda
• **Conectividad USB sobre plataforma Freescale**. Ing. Ramiro Galloso, Electrocomponentes SA
• **Introducción al Sistema operativo de tiempo Real MQX**. Ing. Gabriel Soccodato, Electrocomponentes SA
• **Introducción al lenguaje C para MCUs**. Ing. Roberto Simone - EDUDEVICES
• **Sistemas didácticos en el aprendizaje con Microcontroladores**. Ing. Daniel Di Lella - EDUDEVICES
• **Un recorrido por los mundos de 8 a 32 Bits**. Ing. Daniel Di Lella - EDUDEVICES
• **Ethernet con PIC**. Ariel Coria – MC Electronics
• **WI-FI embebido con PIC y ZeroG**. Coria – MC Electronics
• **GSM hecho fácil con PIC**. Ariel Coria – MC Electronics
• **PIC18 Avanzado**. Andrés Bruno Saravia – MC Electronics – Microchip RTC Argentina
• **USB con PIC**. Jaime Fernández-Caro Belmonte, Director Ejecutivo - Microingenia, S.L.
• **Desarrollo de proyectos con Niple, novedades de la última versión**. Jorge Cano, Niple Software.
• **Plataforma Arduino**. Ing. German Tojeiro Calaza

Inscripción abierta

El congreso se realizará desde el 18 al 30 de octubre. La inscripción se encuentra abierta, pudiendo realizarla o recabar más información en www.areacapacitacion.com.ar o via e-mail a: congreso.microcontroladores@gmail.com.

Una vez registrados en el sitio, se podrán enviar las ponencias al congreso, participar en los foros de debate y recibir boletines informativos mensuales con las novedades del congreso, noticias sobre otros eventos de capacitación, lanzamiento de productos y artículos técnicos.

Segundo Congreso Virtual

MICROCONTROLADORES Y SUS APLICACIONES

www.areacapacitacion.com.ar

18 al 29 de Octubre

UTN

FICH - UNL

Facultad de Ciencia y Tecnología
Universidad Nacional de Entre Ríos

UAI

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Coeticor®

MICROCHIP
RTC ARGENTINA

niple

MICROCHIP

mc electronics®

mie microingenia electronics

ANCA srl
DIVISION INGENIERIA DE TRANSITO

www.edudevices.com.ar

ELKO/ARROW

freescale semiconductor

Electrocomponentes S.A.
www.electrocomponentes.com

CONTINEA

Organiza: Cátedras de Técnicas Digitales -UTN, Facultad Regional Paraná

Display POV

El funcionamiento de un display POV (Persistence Of Vision por sus siglas en ingles), se basa en la característica del ojo humano de seguir viendo una imagen por una fracción de segundo. El mejor ejemplo para entender esto es una película, la cual nos muestra 30 imágenes por segundo, pero nosotros vemos una imagen en movimiento, no cada cuadro.

// por: Emiliano Safarik //
fluf@adinet.com.uy

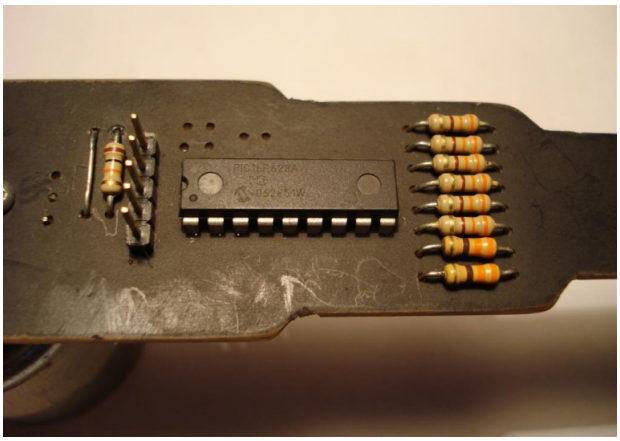
Construcción

Se utilizó un micro-controlador PIC16F628A, que es el cerebro y el encargado de escribir nuestro mensaje en el aire.

El conector ICSP permite programar el micro-controlador sin necesidad de sacarlo del circuito. Del mismo modo evitamos utilizar un zócalo y alivianar el brazo rotativo.

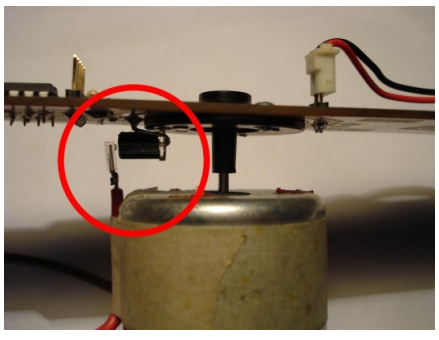
La matriz del display se forma por una columna de 8 leds, preferentemente de alta luminosidad para maximizar el brillo, ya que solo van a estar encendidos unos pocos milisegundos cada vez.

Un fototransistor y 1 led infrarrojo son los que van a indicar la posición 0 del brazo rotativo al micro-controlador, aunque se



podría utilizar un sensor de efecto hall y un pequeño imán, ya que en el circuito está pre-visto para su implementación.
Los fototransisto-res los podemos recuperar de algún Mouse en desuso y el sensor hall de cualquier viejo fan-cooler.

Una de las partes más complicadas en este tipo de proyectos (si no la más complicada) es unir el circuito impreso al motor. El plato de una vieja lectora va a servir



perfectamente para acoplar el motor al circuito impreso, ya que el diámetro de los ejes es el mismo.

Para alimentar al circuito se utilizo un pequeño pack de baterías recargables de 3.6v de un teléfono inalámbrico. Para hacer girar a nuestro display utilizamos un motor de un viejo casetero. Se puede utilizar cualquier otro similar.

Aparte de todo esto, solo hacen falta unos pocos componentes más.

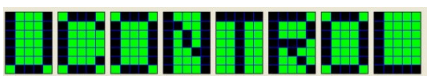
Un aspecto no menos importante es el peso del brazo, ya que por la velocidad que alcanza (cientos de RPM's) tiene que estar muy bien equilibrado.

Código

El programa para escribir nuestro mensaje esta escrito en Basic, y al contrario de lo que parece es por demás sencillo.

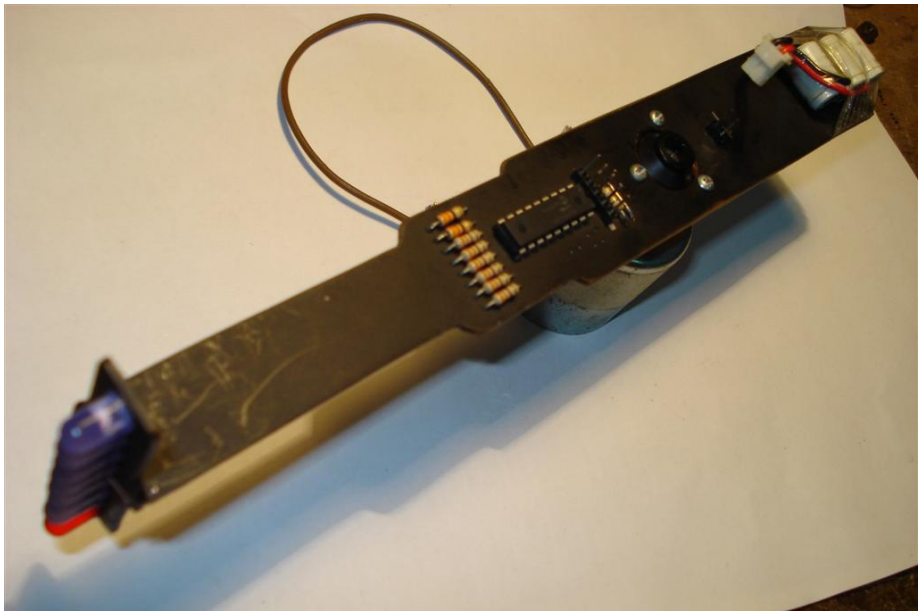
Nuestra columna va a estar enteramente en el puerto B del microcontrolador PIC16F628A.

Cada letra que compone el mensaje se compone por 5 columnas y 7 filas.



Estructura del programa

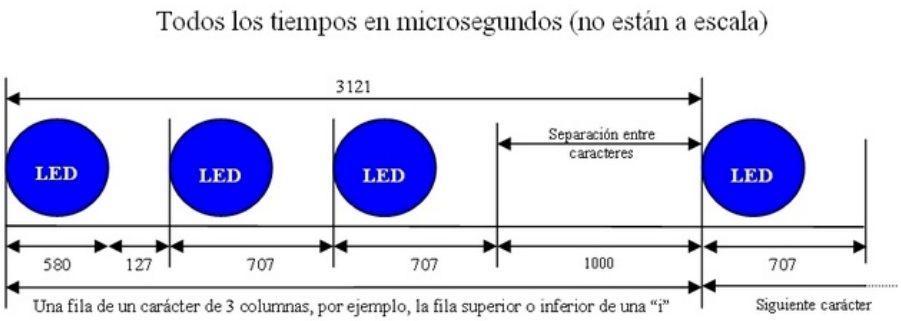
Después de haber definido que micro vamos a utilizar, si lleva o no cristal, cuales pines van a ser entrada y cuales salida, y después de



haber definido nuestras variables, etc.

Lo primero que vamos a hacer es esperar a que el sensor de posición nos diga que el brazo de nuestro display pasó por la posición 0.

Una vez que el brazo pasó por la posición inicial, vamos a escribir la primera columna, la cual va a estar encendida unos pocos milisegundos. Luego apagamos la columna completa y esperamos un tiempo hasta escribir la próxima, así hasta completar la 5 que integran cada letra. A su vez, el tiempo que separa las letras es 2 o 3 veces mayor que el que separa las columnas.



En este caso vamos a utilizar dos variables, una que va a darnos el tiempo que va a estar encendida una columna y otra variable que nos va a dar el tiempo que va a estar apagado entre columna y columna. Así cuando tengamos que modificar los tiempos va a ser mucho más fácil.

También podemos utilizar otra variable para el tiempo de separación entre los distintos caracteres.

Otra manera seria también crear tablas para cada carácter y hacer un GOTO a cada letra, esto facilitaría aun mas el cambiar los mensajes y desplazar el texto.

El tiempo que van a estar los leds encendidos y apagados va a depender de la velocidad de rotación.

```
0525
0526 'U*****************
0527
0528 PORTB = %11111100
0529 PORTB = %00000010
0530 PORTB = %00000010
0531 PORTB = %00000010
0532 PORTB = %11111100
0533
0534
```

La secuencia (a modo de ejemplo) muestra como sería la U de uControl

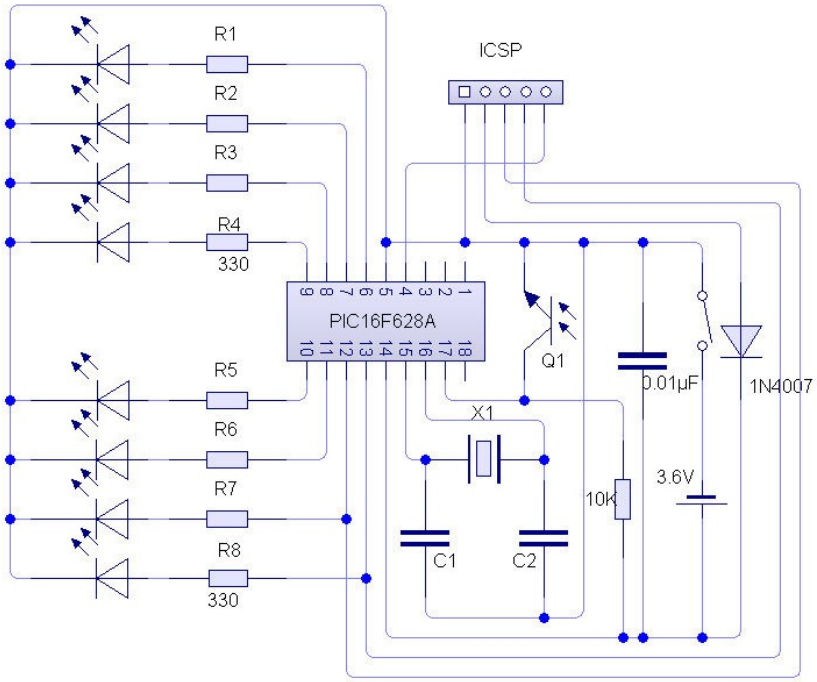
En el ejemplo cada columna esta en binario para que sea mas fácil comprender el funcionamiento, pero la forma corta y quizás la mas práctica sería enviar al puerto B un byte en binario para cada columna.

```
0525
0526 'U*****************
0527
0528 PORTB = %11111100
0529 WaitUs dur
0530 PORTB = 0
0531 WaitUs sep
0532 PORTB = %00000010
0533 WaitUs dur
0534 PORTB = 0
0535 WaitUs sep
0536 PORTB = %00000010
0537 WaitUs dur
0538 PORTB = 0
0539 WaitUs sep
0540 PORTB = %00000010
0541 WaitUs dur
0542 PORTB = 0
0543 WaitUs sep
0544 PORTB = %11111100
0545 WaitUs dur
0546 PORTB = 0
0547 WaitUs esp
0548 Return
```

Subrutina completa de la letra U

La variable “dur” es el tiempo que va a estar encendida una columna. La variable “sep” es el tiempo entre columnas (apagado), y por ultimo “esp” el tiempo entre letra y letra.

Este es el esquema del circuito de nuestro display, es más que simple y no requiere mucha explicación



La batería de 3,6 voltios alimenta el micro. Los 8 leds con sus respectivas resistencias de 330 Ohms están conectados al puerto b del micro.

Q1 es un fototransistor que dará un pulso negativo cuando es excitado por el led infrarrojo.

La implementación del cristal y sus condensadores es opcional, si usan un micro que tenga cristal interno como el de nuestro proyecto no lo necesitarán y ahorrarían componentes y peso adicional al circuito

Lo mismo pasa con el conector ICSP y el diodo 1N4007, en este caso se optó por utilizarlo por su practicidad.

Como verán es un proyecto sencillo y vistoso, además de muy barato, ya que la mayoría de los componentes los podemos recuperar de viejos artefactos o en desuso.

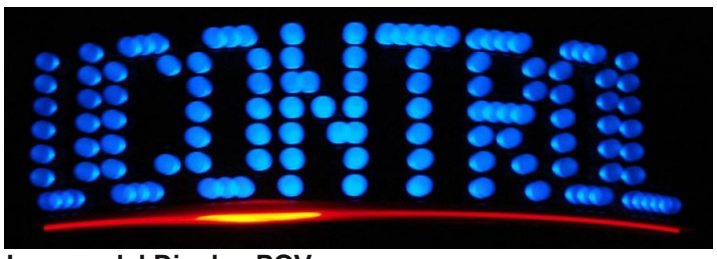


Imagen del Display POV


NEOTEO
Ciencia y tecnología al extremo

Memorias SD/MMC

Realizando una librería a nivel hardware

En muchas aplicaciones de sistemas embebidos es necesario almacenar grandes cantidades de datos en donde las memorias seriales EEPROM son insuficientes, además de que no son portables y leerlas en una computadora requiere de un hardware adicional. En este caso el uso de las memorias flash (SD/MMC) nos brinda una gran ventaja, otorgándonos gran capacidad de almacenamiento, una interface física/eléctrica sencilla y gran disponibilidad en el mercado a costos reducidos.

// por: Alejandro Casanova //
inf.pic.suky@live.com.ar

Introducción

Estas memorias poseen dos maneras de comunicación, la estándar denominado BUS SD que utiliza 4 líneas paralelas para la comunicación (1 Nible de ancho de bus) y comunicación serial SPI. Es esta última forma de comunicación la razón por la cual es sencillo utilizarlo en sistemas embebidos, donde la mayoría de los microcontroladores disponen de interface SPI.

Comencemos viendo su pinout:

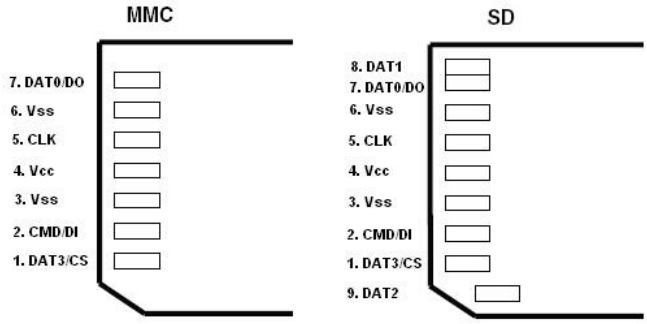


Fig 1: Pinout memorias MMC/SD

Estas memorias trabajan con un nivel de alimentación de 3.3V, por lo tanto si vamos a trabajarlas con un microcontrolador que es alimentado a 5V debemos realizar una interface que adecúe los niveles de tensión. Como hardware mínimo podemos simplemente realizar un divisor resistivo, pero lo adecuado para obtener buenos resultados en comunicaciones a altas frecuencias es utilizar buffer. Se puede utilizar buffer tri-estado como los 74HC125 y 74HC126 o implementar adaptadores de tensión como los que ofrecen Texas Instruments.

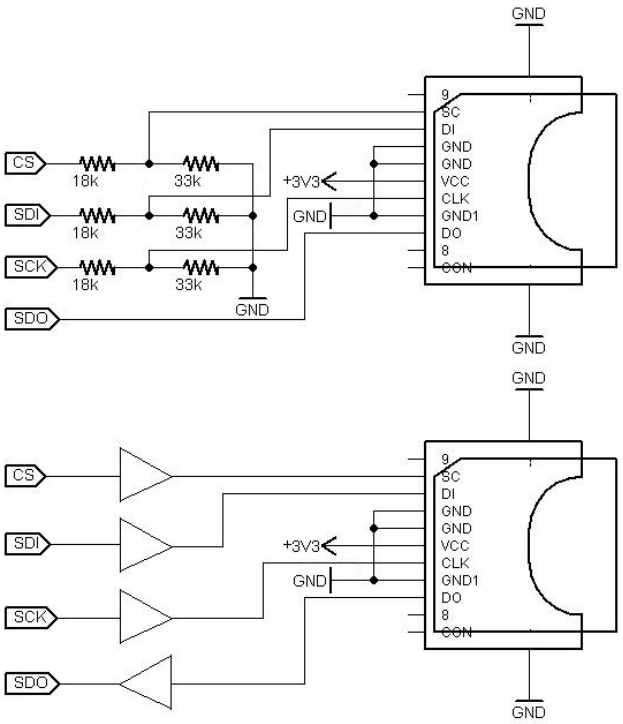


Fig 2: Posibilidades de Hardware

Librería a nivel hardware

Comenzamos con la creación paso a paso de una librería para comunicación a nivel hardware con la memoria, para ello nos guiamos según las especificaciones de las memorias SD de Scandisk (ProdManualSDCardv1.9.pdf). Vamos a crear funciones que puedan inicializar la memoria en modo SPI, escribir/leer sectores de la misma (En este caso 512 Bytes), y leer los registros CSD y CID.

Envío de comandos y respuestas

Los comandos tienen un tamaño fijo de 6 bytes, donde el formato los podemos observar en la figura 3.

Byte 1			Bytes 2—5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Command	Command Argument	CRC	1

Fig 3: Formato de comandos

CMD INDEX	SPI Mode	Argument	Resp	Abbreviation	Command Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets the SD Card
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its card-specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its card identification (CID).
CMD12	Yes	None	R1b	STOP _TRANSMISSION	Forces the card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).[1]
CMD17	Yes	[31:0] data address	R1	READ_SINGLE _BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.[2]
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE _BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command.
CMD24	Yes	[31:0] data address	R1[3]	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.[4]
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop transmission token is sent (instead of 'start block').
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28[1]	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card specific data (WP_GRP_SIZE).
CMD29[4]	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_ PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.[2]
CMD32	Yes	[31:0] data address	R1	ERASE_WR_BLK_START _ADDR	Sets the address of the first write block to be erased.
CMD33	Yes	[31:0] data address	R1	ERASE_WR_BLK_END _ADDR	Sets the address of the last write block in a continuous range to be erased.
CMD38	Yes	[31:0] don't care*	R1b	ERASE	Erases all previously selected write blocks.

Fig 4: Comandos utilizados

Comandos y argumentos utilizados

En la figura 4 podemos ver algunos de los comandos que se pueden enviar a la memoria, y que son suficientes como para trabajar con ella. Vemos que al utilizar solo los comandos mostrados vamos a recibir respuesta del tipo R1 y R1b.

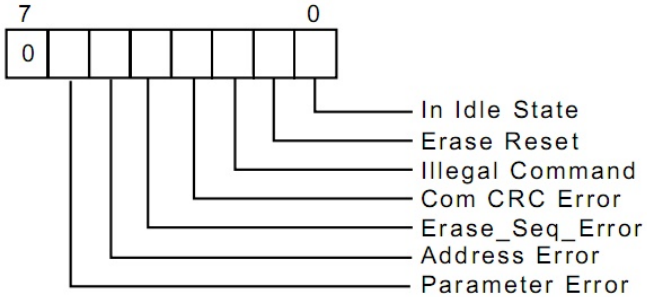


Fig 5: Respuesta R1

R1b es idéntica a R1, solo que luego de recibir la respuesta hay que esperar condición de memoria desocupada en donde la línea SDO de la memoria se lleva a nivel de 0 lógico.

Ya visto la estructura del comando y que respuesta vamos a recibir por parte de la memoria ya podemos desarrollar nuestra primera función que enviara un comando y esperará respuesta. Pero para ello vamos hacer algunas definiciones tales como las variables que vamos a utilizar para generalizar el código y otras para funcionamiento del código:

Card_int8 : Variable de 8 bit sin signo.
Card_int16: Variable de 16 bit sin signo.
Card_int32: Variable de 32 bit sin signo.

```
// Definición de pin que controla CS de la memoria.
#define SD_CS …
// Definición de comandos
#define CMD0 0        // Resetea la SD Card.-
#define CMD1 1        // Activa proceso de inicialización de SD Card.
#define CMD9 9        // Lectura registro CSD.-
#define CMD10 10      // Lectura registro CID.-
#define CMD16 16      // Selecciona largo del bloque para lectura/escritura (1 a 512)
#define CMD17 17      // Lectura de un único bloque.
#define CMD24 24      // Escritura de un único bloque.
#define CMD59 59      // Enciende/Apaga CRC.
#define CMD32 32      // Setea dirección  del primer bloque a borrar.-
#define CDM33 33      // Setea dirección del último bloque en un continuo rango a borrar.-
#define CMD38 38      // Borra todos los sectores entre las direcciones establecidas.-

#define BLOCK_SIZE 512 // Tamaño del bloque de lectura/escritura.

#define SDSelect()  {SD_CS=0;spi_write(0xFF);}
#define SDDeselect()  {spi_write(0xFF); SD_CS=1;}

/** \brief Envía comando a memoria SD.-

   \param cmd: Comando a enviar.-
   \param arg: Argumento del comando.-
   \return res: Respuesta al comando.-
   \return  0 a ocurrido algún error, 1 todo ok.-
*/
```

```
Card_int8 SDCard_send_command(Card_int8 cmd,Card_int32 arg,Card_int8 *res){

  Card_int8 Trama[6]; // Comando, Argumento y CRC
  Card_int8 ResTmp;
  Card_int16 i;

  // Construimos el paquete a enviar.-
  Trama[0] = cmd | 0x40;
  Trama[1] = (arg>>24);
  Trama[2] = (arg>>16);
  Trama[3] = (arg>>8);
  Trama[4] = (arg);

  if(cmd==0){ [5]=0x95;}
  else{Trama[5]=0xFF;}

  // Enviamos clock para sincronizar envío.
  for( i=0;i<16;i++) spi_xfer(SDCard,0xFF);

  // Transfiere el paquete.-
  spi_write(Trama[0]);
  spi_write(Trama[1]);
  spi_write(Trama[2]);
  spi_write(Trama[3]);
  spi_write(Trama[4]);
  spi_write(Trama[5]);

  //  Se espera respuesta R1
  i=0;
  do{
    ResTmp=spi_read(0xFF);
    i++;
  }while((ResTmp&0x80)!=0 && i<2000);// mientras no corresponda con 0xxxxxxx.-
  if(i==2000){SDDeselect();return(0);}
  *res=ResTmp;
  return(1);
}
```

Inicialización en modo SPI

Al iniciar la memoria se establece en modo SD. Para entrar en modo SPI se debe enviar el comando CMD0 con el pin CS a 0V, si la memoria reconoce la petición de cambio de protocolo responde con la respuesta R1.

Luego de haber fijado el protocolo SPI procedemos a encender la memoria y configurar algunos parámetros de trabajo, como largo del bloque para lectura y escritura, fijar modo CRC, etc.

```
/** \brief Inicialización de SD Card
   \param Ninguno
   \return 1 si se ha iniciado correctamente, 0 caso contrario.
*/
Card_int8 SDCard_init(void){
Card_int8 Respuesta;
Card_int16 I;

/* ** Configuramos Modulo SPI del microcontrolador ** */
// En CCS Compiler ->
SETUP_SPI(SPI_MASTER|SPI_CLK_DIV_4|SPI_H_TO_L|SPI_XMIT_L_TO_H );

// En C18 -> OpenSPI(SPI_FOSC_4, MODE_11, SMPMID);
   SD_CS=1;
   delay_ms(20);
   for(i=0;i<20;i++) spi_write(0xFF);   // Para sincronización.
   SD_CS=0;
   delay_ms(20);
   i=0;
   do{
      if(SDCard_send_command(CMD0,0x00,&Respuesta)==0){
         return(0);        // Si se recibe 0 error por timeout.-
      }else{i++;}
   }while(((Respuesta&0x01)==0x00) && i<2000);
   if(i>=2000){return(0);} // Timeout esperando desactivación de SD Card.
   // Pasamos a esperar condición de SD Card activa.
   // Si R1=0x01, SD Card en estado desactivado.
      i=0;
   do{
      if(SDCard_send_command(CMD1,0x00,&Respuesta)==0){
         return(0);     // Si se recibe 0 error por timeout.-
      }else{i++;}
   }while(((Respuesta&0x01)==0x01) && (i<2000)); // Mientras se reciba R1=0x01
   if(i>=2000){return(0);} // Timeout esperando activación de SD Card.

   // Se fija largo del bloque lectura/escritura.
   if(SDCard_send_command(CMD16,BLOCK_SIZE,&Respuesta)==0){
      return(0);
   }else if(Respuesta!=0){
      return(0);
   }
   // Se desactiva CRC.
   if(SDCard_send_command(CMD59,0,&Respuesta)==0){
      return(0);
   }else if(Respuesta!=0){
      return(0);
   }
   SDDeselect();
   return(1);
}
```

Operaciones de escritura y lectura

La figura 6 nos indica de forma clara que modos de escritura y lectura son soportados:

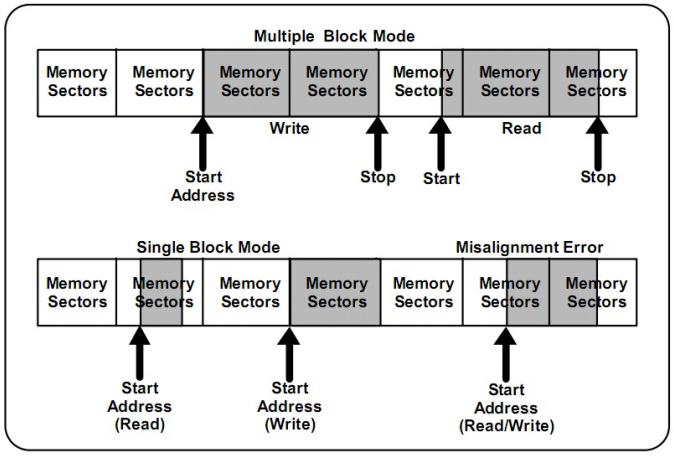


Fig 6: Modos de escritura/lectura

Las operaciones de lectura y escritura se realizan enviando el comando correspondiente junto a la dirección del primer byte del bloque con el largo indicado anteriormente (Comando CMD16). El largo del bloque puede ser desde 1 hasta 512, y no está permitido realizar operaciones en dos sectores a la vez, o sea que si el largo de bloque fijado en CMD16 es 512 la dirección para realizar lectura o escritura debe ser la del byte inicial del sector.

Realizando escritura de un sector

Para realizar escritura de un único bloque debemos enviar el comando CMD24 indicando la dirección del bloque en el argumento de la función. La memoria al reconocer el comando envía la respuesta R1, donde puede indicar si hay algún error. Si todo es correcto el microcontrolador debe enviar un token (0xFE) y luego los 512 datos del bloque más 2 bytes de CRC. Luego de enviado estos datos debemos quedar a la espera de una respuesta de la memoria indicando si los datos se han recibido correctamente o ha ocurrido un error, además de condición de desocupado.

En la figura 7 podemos observar el protocolo:

Fig 7: Protocolo de escritura

Respuesta:

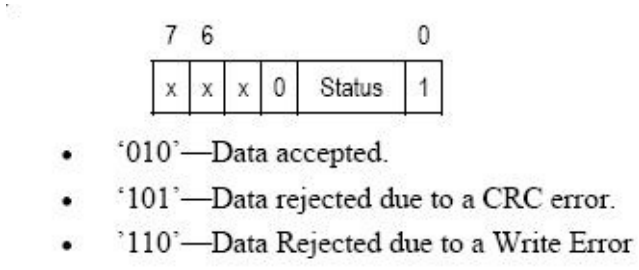


Fig 8: Respuesta de comando escritura

```
/** \brief Realiza escritura de un bloque.-

   \param Address: Dirección del sector.-
   \param Buffer: Buffer a escribir en memoria.-
   \return  0 a ocurrido algún error, 1 todo ok.-
*/
Card_int8 SDCard_write_block(Card_int32 Address,Card_int8 *Buffer){
Card_int8 Respuesta,ResTmp;
Card_int16 i;

   SDSelect();
   // Se envía comando para escribir bloque de bytes.-
   if(SDCard_send_command(CMD24,Address,&Respuesta)==0){
      return(0);
   }else if(Respuesta!=0){
      return(0);
   }
   // Enviamos Token.
   spi_write(0xFE);
   // Enviamos data.-
   for(i=0;i<BLOCK_SIZE;i++){
      spi_write(*Buffer++);
   }
   // Ignoramos CRC.-
   spi_write(0xFF);
   spi_write(0xFF);
   // Esperamos respuesta.-
   i=0;
   do{
      ResTmp=spi_read(0xFF);
      i++;
   }while(ResTmp==0xFF && i<2000);// Mientras sea 0xFF.-
   if(i==2000){SDDeselect();return(0);}
   if((ResTmp&0x1F)!=0x05){ // Si no se recibe ***00101.-
      SDDeselect();
      return(0);
   }
   // Esperamos a que baje la línea, indicando escritura completa.-
   while(spi_read(0xFF)==0);
   SDDeselect();
   return(1);
}
```

Realizando lectura de un sector

Para realizar una lectura debemos enviar el comando CMD17 indicando en el argumento la dirección del bloque. Luego se espera la respuesta R1 desde la memoria, si todo es correcto se pasa a recibir el token y luego los datos, la cantidad es establecida por el largo del bloque. (CMD16)

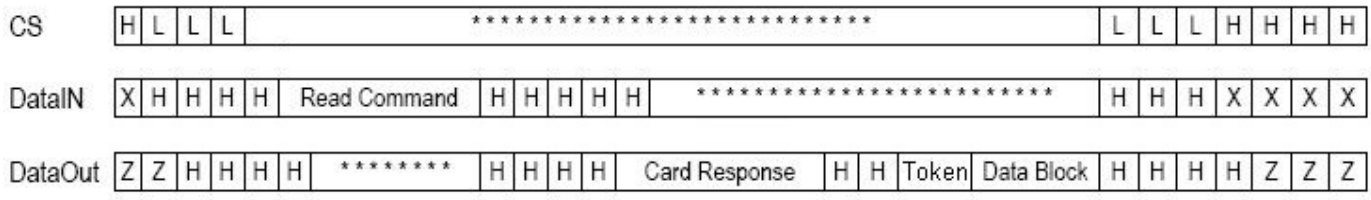


Fig 9: Protocolo de lectura

```
/** \brief Realiza lectura de un bloque.-

   \param Address: Direccion del sector.-
   \return Buffer: Buffer donde se almacena lectura de memoria.-
   \return  0 a  ocurrido algún error, 1 todo ok.-
*/
Card_int8 SDCard_read_block(Card_int32 Address,Card_int8 *Buffer){
Card_int8 Respuesta, TokenTmp;
Card_int16 i;

   // Se envía comando para leer bloque de bytes.-
      SDSelect();
   if(SDCard_send_command(CMD17,Address,&Respuesta)==0){
      return(0);
   }else if(Respuesta!=0){
      return(0);
   }
   // Pasamos a esperar Token.
   i=0;
   do{
      TokenTmp=spi_read(SDCard,0xFF);
      i++;
   }while(TokenTmp==0xFF && i<2000);// Mientras sea 0xFF.-
   if(i==2000){SDDeselect();return(0);}
   if((TokenTmp&0xE0)==0){ // Si se recibe 000xxxxx y no 0xFE.-
      SDDeselect();
      return(0);
   }
      // Todo ok, recibimos data.-
   for(i=0;i<BLOCK_SIZE;i++){
      *Buffer++=spi_read(SDCard,0xFF);
   }
   // Ignoramos CRC.-
   spi_read(SDCard,0xFF);
   spi_read(SDCard,0xFF);

   SDDeselect();
   return(1);
}
```

Registros CSD y CID

También podemos crear funciones dedicadas a la lectura de los registros CID y CSD. El registro CID es de 16 bytes de longitud y contiene información de identificación de la memoria, tales como fabricante, identificación del producto, ect. Luego el registro CSD contiene información de configuración para acceso a la memoria. En la página 37 de las especificaciones de las memorias SD de Scandisk podemos ver en detalle cada registro y su contenido.

```
Card_int8 SDCard_read_CSD(void){
Card_int8 Respuesta,TokenTmp, buf[16];
Card_int16 i;
SDSelect();
if(SDCard_send_command(CMD9,0,&Respuesta)==0){
  return(0);
}else if(Respuesta!=0){
  return(0);
}
// Pasamos a esperar Token.
i=0;
do{
  TokenTmp=spi_read(0xFF);
  i++;
}while(TokenTmp==0xFF && i<2000);// Mientras sea 0xFF.-
if(i==2000){SDDeselect();return(0);}
if((TokenTmp&0xE0)==0){ // Si se recibe 000xxxxx y no 0xFE.-
  SDDeselect();
  return(0);
}
// Todo ok, recibimos data.-
for(i=0;i<16;i++){
  buf[i]=spi_read(0xFF);
}
SDDeselect();

printf("\r\nCSD_STRUCTURE: %X", (buf[0] & 0x0C) >> 2);
printf("\r\nTAAC: %X", buf[1]);
printf("\r\nNSAC: %X", buf[2]);
printf("\r\nTRAN_SPEED: %X", buf[3]);
printf("\r\nCCC: %lX", ((((Card_int8)(buf[4])<<8)+ buf[5]) & 0xFFF0) >> 4);
printf("\r\nREAD_BL_LEN: %X", buf[5] & 0x0F);
printf("\r\nREAD_BL_PARTIAL: %X", (buf[6] & 0x80) >> 7);
printf("\r\nWRITE_BLK_MISALIGN: %X", (buf[6] & 0x40) >> 6);
printf("\r\nREAD_BLK_MISALIGN: %X", (buf[6] & 0x20) >> 5);
printf("\r\nDSR_IMP: %X", (buf[6] & 0x10) >> 4);
printf("\r\nC_SIZE: %lX", (((buf[6] & 0x03) << 10) | (buf[7] << 2) | ((buf[8] & 0xC0) >> 6)));
printf("\r\nVDD_R_CURR_MIN: %X", (buf[8] & 0x38) >> 3);
printf("\r\nVDD_R_CURR_MAX: %X", buf[8] & 0x07);
printf("\r\nVDD_W_CURR_MIN: %X", (buf[9] & 0xE0) >> 5);
printf("\r\nVDD_W_CURR_MAX: %X", (buf[9] & 0x1C) >> 2);
printf("\r\nC_SIZE_MULT: %X", ((buf[9] & 0x03) << 1) | ((buf[10] & 0x80) >> 7));
printf("\r\nERASE_BLK_EN: %X", (buf[10] & 0x40) >> 6);
printf("\r\nSECTOR_SIZE: %X", ((buf[10] & 0x3F) << 1) | ((buf[11] & 0x80) >> 7));
```

```
   printf("\r\nWP_GRP_SIZE: %X", buf[11] & 0x7F);
   printf("\r\nWP_GRP_ENABLE: %X", (buf[12] & 0x80) >> 7);
   printf("\r\nR2W_FACTOR: %X", (buf[12] & 0x1C) >> 2);
   printf("\r\nWRITE_BL_LEN: %X", ((buf[12] & 0x03) << 2) | ((buf[13] & 0xC0) >> 6));
   printf("\r\nWRITE_BL_PARTIAL: %X", (buf[13] & 0x20) >> 5);
   printf("\r\nFILE_FORMAT_GRP: %X", (buf[14] & 0x80) >> 7);
   printf("\r\nCOPY: %X", (buf[14] & 0x40) >> 6);
   printf("\r\nPERM_WRITE_PROTECT: %X", (buf[14] & 0x20) >> 5);
   printf("\r\nTMP_WRITE_PROTECT: %X", (buf[14] & 0x10) >> 4);
   printf("\r\nFILE_FORMAT: %X", (buf[14] & 0x0C) >> 2);
   printf("\r\nCRC: %X\r\n", buf[15]);
   return(1);
}

Card_int8 SDCard_read_CID(void){
Card int8 Respuesta,TokenTmp, buf[16];
Card_int16 i;

   SDSelect();
   if(SDCard_send_command(CMD10,0,&Respuesta)==0){
      return(0);
   }else if(Respuesta!=0){
      return(0);
   }
   // Pasamos a esperar Token.
   i=0;
   do{
      TokenTmp=spi_read(0xFF);
      i++;
   }while(TokenTmp==0xFF && i<2000);// Mientras sea 0xFF.-
   if(i==2000){SDDeselect();return(0);}
   if((TokenTmp&0xE0)==0){ // Si se recibe 000xxxxx y no 0xFE.-
      SDDeselect();
      return(0);
   }

   // Todo ok, recibimos data.-
   for(i=0;i<16;i++){
      buf[i]=spi_read(0xFF);
   }
   SDDeselect();

   printf("\r\nManufacturer ID: %X", buf[0]);
   printf("\r\nOEM/Application ID: %c%c", buf[1], buf[2]);
   printf("\r\nProduct Name: %c%c%c%c%c", buf[3], buf[4], buf[5], buf[6], buf[7]);
   printf("\r\nProduct Revision: %X", buf[8]);
   printf("\r\nSerial Number: %X%X%X%X", buf[9], buf[10], buf[11], buf[12]);
   printf("\r\nManufacturer Date Code: %X%X", buf[13] & 0x0F, buf[14]);
   printf("\r\nCRC-7 Checksum: %X\r\n", buf[15]);
   return(1);
}
```

Podemos hacer un pequeño programita donde se inicialice la memoria, leemos registro CID, CSD, escribimos un sector y lo leemos.

Al hacer un debug el resultado debería ser equivalente al de la figura 10.

```
******* Iniciando Memoria SD Card... *******

--> Se inicia sincronizacion

--> Se envia CMD0 (Se desactiva SD Card)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:1

--> Se envia CMD1 (Se activa SD Card)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0

--> Se envia CMD16 (Se fija largo del Bloque para escritura/lectura : 512)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0

******* Leemos registro CID: *******

--> Se envia CMD10 (Lectura de CID)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0
   >> Toquen recibido: 0xFE

Manufacturer ID: 00
OEM/Application ID:
Product Name: MMC03
Product Revision: 32
Serial Number: 01000000
Manufacturer Date Code: 0195
CRC-7 Checksum: 03

******* Leemos registro CSD: *******

--> Se envia CMD9 (Lectura de CSD)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0
   >> Toquen recibido: 0xFE

CSD_STRUCTURE: 02
TAAC: 26
NSAC: 00
TRAN_SPEED: 2A
CCC: 0D15
READ_BL_LEN: 09
READ_BL_PARTIAL: 00
WRITE_BLK_MISALIGN: 00
READ_BLK_MISALIGN: 00
DSR_IMP: 00
C_SIZE: FF
VDD_R_CURR_MIN: 06
VDD_R_CURR_MAX: 06
VDD_W_CURR_MIN: 06
VDD_W_CURR_MAX: 06
C_SIZE_MULT: 02
ERASE_BLK_EN: 00
SECTOR_SIZE: 00
WP_GRP_SIZE: 00
WP_GRP_ENABLE: 00
R2W_FACTOR: 04
WRITE_BL_LEN: 09
WRITE_BL_PARTIAL: 00
FILE_FORMAT_GRP: 00
COPY: 00
PERM_WRITE_PROTECT: 00
TMP_WRITE_PROTECT: 00
FILE_FORMAT: 00
CRC: 01

******* Escribimos sector 0x200 *******

--> Se envia CMD24 (Escritura de bloque)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0
   >> Se envia Token (0xFE)
   >> Se envia Bloque de datos
   >> Respuesta de recepci?n del bloque: 0x05  (***00101)
   >> Escritura Terminada

******* Leemos sector 0x200 *******

--> Se envia CMD17 (Lectura de bloque)
          >> Repeticiones para respuesta de CmdXX: 2; Respuesta recibida:0
   >> Toquen recibido: 0xFE
   >> Terminada la lectura
   >> Datos leidos:

00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,
20,21,22,23,24,25,26,27,28,29,2A,2B,2C,2D,2E,2F,30,31,32,33,34,35,36,37,38,39,3A,3B,3C,3D,3E,3F,
40,41,42,43,44,45,46,47,48,49,4A,4B,4C,4D,4E,4F,50,51,52,53,54,55,56,57,58,59,5A,5B,5C,5D,5E,5F,
60,61,62,63,64,65,66,67,68,69,6A,6B,6C,6D,6E,6F,70,71,72,73,74,75,76,77,78,79,7A,7B,7C,7D,7E,7F,
80,81,82,83,84,85,86,87,88,89,8A,8B,8C,8D,8E,8F,90,91,92,93,94,95,96,97,98,99,9A,9B,9C,9D,9E,9F,
A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,AA,AB,AC,AD,AE,AF,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,BA,BB,BC,BD,BE,BF,
C0,C1,C2,C3,C4,C5,C6,C7,C8,C9,CA,CB,CC,CD,CE,CF,D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,DA,DB,DC,DD,DE,DF,
E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,EA,EB,EC,ED,EE,EF,F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,FA,FB,FC,FD,FE,FF,
00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,
20,21,22,23,24,25,26,27,28,29,2A,2B,2C,2D,2E,2F,30,31,32,33,34,35,36,37,38,39,3A,3B,3C,3D,3E,3F,
40,41,42,43,44,45,46,47,48,49,4A,4B,4C,4D,4E,4F,50,51,52,53,54,55,56,57,58,59,5A,5B,5C,5D,5E,5F,
60,61,62,63,64,65,66,67,68,69,6A,6B,6C,6D,6E,6F,70,71,72,73,74,75,76,77,78,79,7A,7B,7C,7D,7E,7F,
80,81,82,83,84,85,86,87,88,89,8A,8B,8C,8D,8E,8F,90,91,92,93,94,95,96,97,98,99,9A,9B,9C,9D,9E,9F,
A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,AA,AB,AC,AD,AE,AF,B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,BA,BB,BC,BD,BE,BF,
C0,C1,C2,C3,C4,C5,C6,C7,C8,C9,CA,CB,CC,CD,CE,CF,D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,DA,DB,DC,DD,DE,DF,
E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,EA,EB,EC,ED,EE,EF,F0,F1,F2,F3,F4,F5,F6,F7,F8,F9,FA,FB,FC,FD,FE,FF,
```

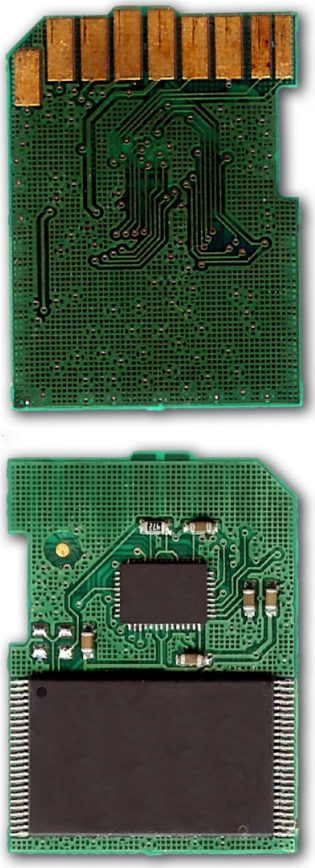


Fig 10:
Debug realizado a una memoria MMC


INFOPIC
www.infopic.comlu.com
+ Tutoriales
* Assembler
* MPLAB C18
* MPLAB C30
+ Proyectos
* USB
* SD Card, FAT
* Ethernet
* PICKIT2 Clone
+ Y mucho más...
R2
C16

Curso de semiconductores (2)

En la primera parte del curso de Introduccion a los Semiconductores, tratamos el componente DIODO; Recorriendo y Conociendo su funcionamiento basico, parámetros, condiciones de funcionamiento y empleos mas comunes en nuestros circuitos cotidianos.
En esta segunda entrega, conoceremos los principios basicos de funcionamiento que tienen los TRANSISTORES para poder ser empleados en nuestros circuitos diarios.

// por: Ing. Martín Torres //
torres.electronico@gmail.com

Válvulas Termoiónicas

El tubo fue inventado por el científico Británico John Ambrose Fleming en el año 1904, al utilizar una válvula diodo (el diodo Fleming) para pasar corriente alterna a corriente directa (proceso de rectificación). Muchos intentaron mejorar este diodo, pero no lo lograron hasta que en 1907, un inventor de Nueva York, Lee de Forrest, patentó el mismo diodo que Fleming, sólo que con un electrodo más, creando el primer amplificador electrónico verdadero, "El Triodo". Después vino el Tetrodo, el Pentodo y más, en muy diferentes versiones. Desde esta fecha hasta los años 60 su desarrollo fue continuo.

Fueron muy utilizados en las décadas de los 50 y 60 previos a la invención del famoso y ya mencionado transistor. Si quieres ver algún tubo, lo puedes encontrar en antiguos equipos de sonido, radios y televisión, que no estén en uso.

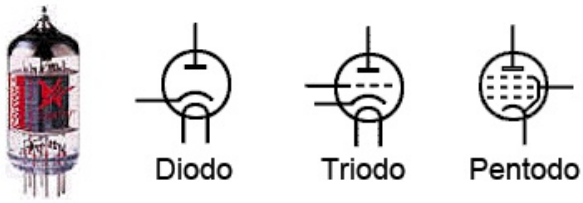


Imagen y símbolos de algunos tipos de válvulas

Los transistores, con su bajo consumo de energía y pequeño tamaño pueden utilizarse en equipos electrónicos portátiles que funcionen con pilas (baterías), algo muy difícil de obtener con los tubos, cuyas desventajas son: su tamaño y su alto consumo de energía.

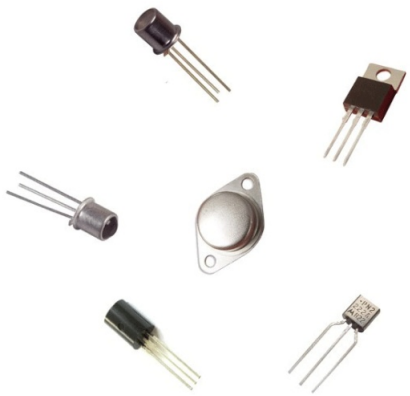
Pero a partir de los años 90 los tubos volvieron a hacer su aparición (en forma evidente).

Pero, ¿qué virtudes tiene el tubo para que hoy en día se los esté nuevamente tomando en cuenta?

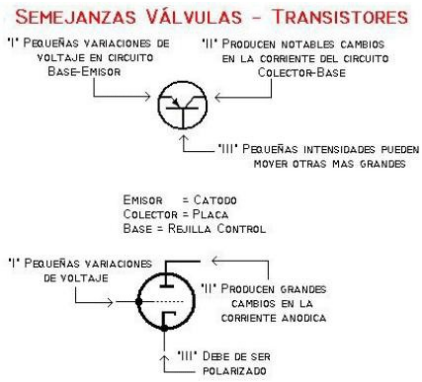
El tubo se puede utilizar para salidas de alta potencia en equipos de audio, amplificadores de guitarra, etc. Además si alguna vez ha visto un diagrama de un amplificador de tubos se habrá dado cuenta que son mucho más sencillos que uno similar de transistores y tienen una calidad de sonido superior a un equipo de alta fidelidad actual. Además de que hay grandes cantidades de tubos totalmente nuevos en existencia para la venta y países como Rusia, China y algunos países del este de Europa aún los siguen fabricando, así que, hay tubos para rato. Pronto en nuestra sección de tutoriales de electrónica habrá información sobre el funcionamiento de un tubo o válvula.

Transistores

Los transistores han facilitado en gran medida el diseño de circuitos electrónicos de reducido tamaño, gran versatibilidad y facilidad de control.

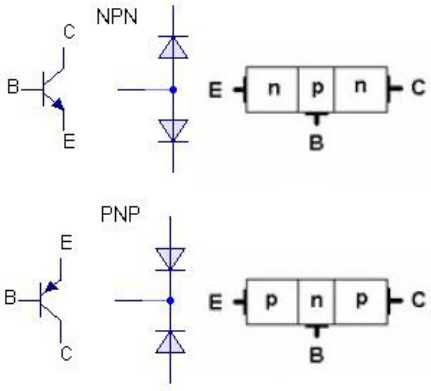


Estos vienen a sustituir a las antiguas “válvulas termoiónicas”.

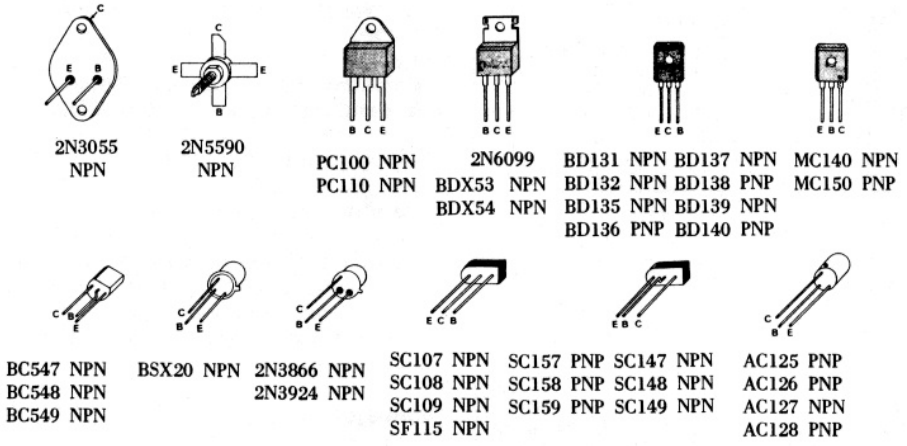


Gracias a los transistores se pudieron hacer equipos portátiles a pilas o baterías, ya que los aparatos valvulares, trabajan a tensiones muy altas, y tardaban en su momento hasta 30 segundos en empezar a funcionar.

El transistor es un elemento electrónico con tres terminales, formado por tres capas de material semiconductor que alternan el dopado tipo “N” y tipo “P”. Según la conformación de estas tres capas de semiconductores (que son cristal), nos podemos encontrar con dos tipos de configuraciones, o mejor dicho, nos podemos encontrar con dos tipos de transistores... Los transistores NPN, y los transistores PNP.



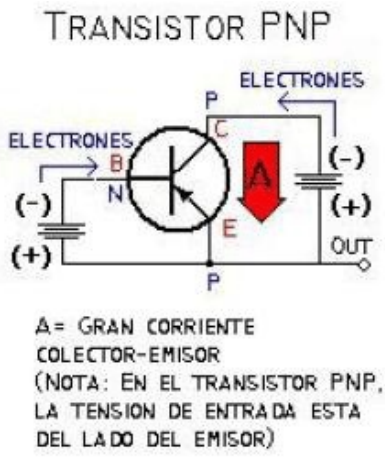
Capas de composición en un transistor NPN y un transistor PNP

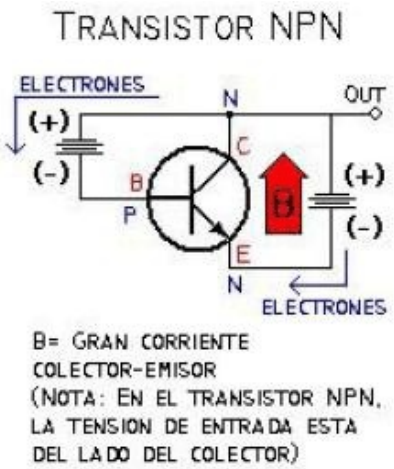


Tipos de encapsulados de transistores

Principio de funcionamiento de un transistor

Para que un transistor funcione correctamente, deben aplicarse unas tensiones continuas a sus terminales. Si mediante una fuente conformada por dos baterías aplicáramos unas tensiones positivas a la base y al colector de un transistor NPN, respecto del emisor y la base respectivamente, el diodo PN formado por la base y el emisor, estará polarizada directamente produciendo una corriente "Base-Emisor"; En el caso de los transistores PNP, seria la inversa.



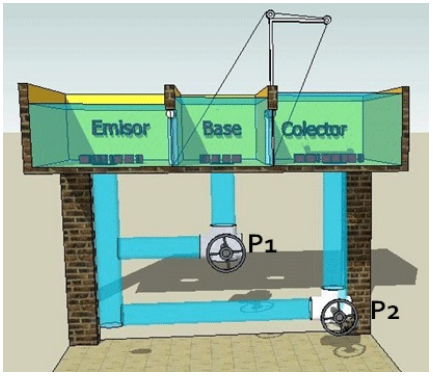


Seria de esperar que esta corriente de electrones que salen del emisor lleguen íntegramente a la base y drenen por ahí, pero esto no ocurre así, dado que en este caso son atraídos por la tensión positiva del colector, así que podemos decir que la mayoría se dirige al mismo; Para acentuar este efecto producido, la base se construye poco dopada y muy estrecha, osea, se aplica una pequeña corriente, para poder mover una mayor.

Veamos una comparación de un caso de la vida real para acentuar más la idea del principio de funcionamiento de los transistores...

Tratemos de Imaginar un tanque de agua con tres compartimentos internos separados entre si por dos compuertas mecánicas…

A estos compartimentos, los llamaremos Emisor - Base - Colector

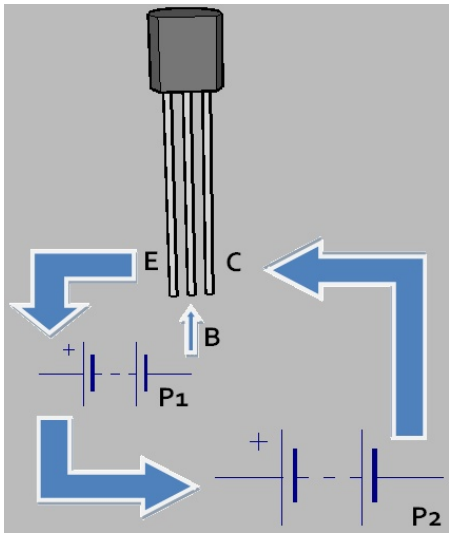


En la Base de los compartimentos Colector y Base, se encuentra una compuerta a la cual llamaremos D2. Puede subir y bajar, gracias a un dispositivo mecánico mandado por la otra compuerta (D1).

Una bomba P1 comunica Base con Emisor. No gira mas que en un solo sentido, aspirando el agua de Emisor e introduciéndola en Base.

Una bomba P2 comunica con Colector y Emisor, no gira nada mas que en un solo sentido, aspira agua de Emisor y la introduce en Colector.

Decimos que el sistema hidráulico presentado es el equivalente al funcionamiento de un transistor, porque un transistor es la asociación de tres elementos unidos que se denominan Emisor, Base y Colector, a las cuales corresponde los tres compartimentos descritos en el ejemplo anterior.



El Emisor y la Base de un transistor, constituyen un diodo, al cual corresponde la válvula D1, El Colector y la Base del transistor, constituyen otro diodo, comportándose en los montajes como lo hace la compuerta D2. Una pila P1 a la cual correponde la bomba P1, esta conectada entre la Base y el Emisor del transistor.

Una pila P2 a la cual corresponde la bomba P2, esta conectada entre el Emisor y el Colector del transistor…

Estudiemos ahora detenidamente el funcionamiento del mecanismo donde funciona P1 y P2….

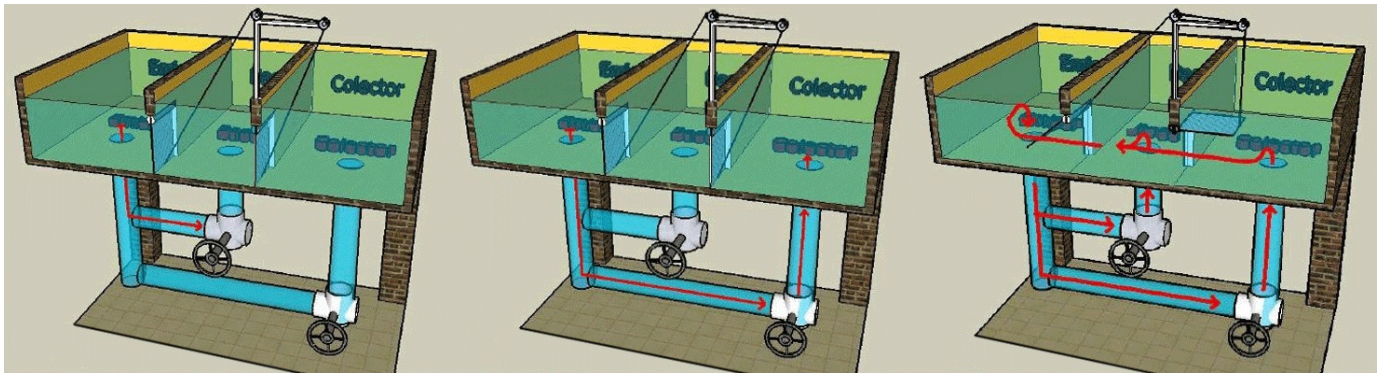
Cuando la bomba P1 no gira, la compuerta D1 esta cerrada y D2 lo esta también.

Si la bomba P2 gira, ninguna circulación se produce en el recipiente, porque hay dos compuertas herméticas entre los compartimentos Colector y Emisor.

Cuando la bomba P1 gira, aspira agua de Emisor y la introduce hacia el compartimento Base, cambiando todo, pues la presión del liquido en Base, ejerce una presión sobre la compuerta D1 y la abre. En el mismo instante, por el mecanismo que teníamos entre las dos compuertas, D2 se eleva y así como esta ofrece su apertura, P2 hace circular el liquido de Colector hacia Base y después hacia Emisor.

La corriente que circula entre Base y Emisor bajo la acción de la bomba P1, es siempre una corriente débil por que no hay esfuerzo que desarrollar para elevar la compuerta D2…

La corriente que circula entre Colector y Emisor bajo la acción de la bomba P2, es siempre una corriente intensa porque la compuerta abre la entrada a una gran circulación de agua.



Sistema hidráulico de ejemplo

Se puede entonces mandar una gran corriente por medio de una pequeña bomba:

P1 es una bomba de mando, pues ella abre la compuerta D2.

P2 es una bomba de alimentación, por que ella hace circular el agua.

Esto mismo es un transistor; La corriente intensa que circula entre Colector y Emisor bajo la acción de la pila P2, es mandada por una corriente débil que circula entre Base y Emisor, bajo la acción de la bomba P1. P1 es llamada pila o fuente de polarización y P2 pila de alimentación. El transistor, nos permite gobernar y/o digir grandes cosas, con pequeñas cosas…

Amplificación

Coloquemos una pequeña bomba Alterna en el circuito de P1 y la llamaremos P1´.

Su pistón interno, se mueve tanto en un sentido, como en el otro, tal como se muestra en la figura…

Con esta acción, la bomba alterna refuerza y disminuye la acción de P1 alternativamente.

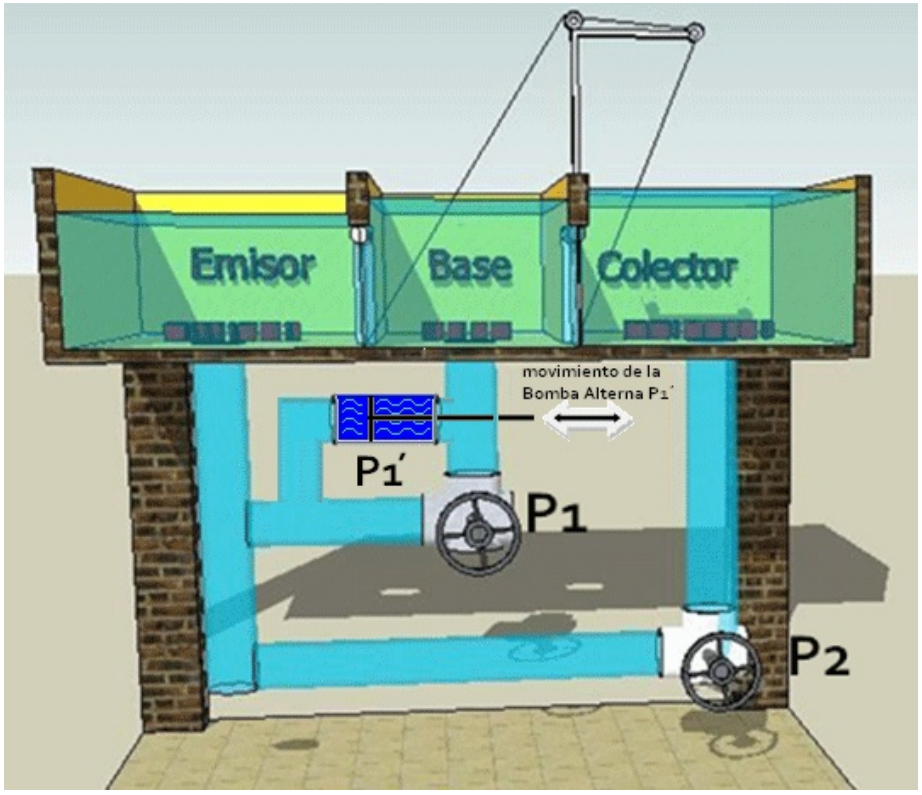
La compuerta D1 (la situada entre el E-B), sigue todas estas variaciones por que la corriente que lo eleva, unas veces aumenta, y otras disminuye.

La compuerta D2 hace exactamente la misma cosa, sube y baja alternativamente.

La corriente que va de C (Colector) hacia E (Emisor) bajo la presión de la bomba P2, aumenta también cuando la compuerta D2 se eleva y disminuye P2 cuando D2 baja.

La bomba P1´ gracias al mecanismo actúa sobre un circuito en el cual no está directamente colocada.

Ella hace aumentar o disminuir la corriente debido a P2, Como lo haría una Bomba P2´ mucho mas importante colocada en el circuito de la bomba P2.



Decimos “mucho mas importante”, por que las débiles variaciones de corriente de mando se traducen, gracias a este mecanismo por variaciones semejantes pero mas grandes de las corrientes debido a P2; Esto es exactamente lo mismo que

ocurre en un transistor. Se pueden obtener variaciones muy importantes de corrientes, debidas a P2, partiendo de variaciones semejantes, pero mucho mas débiles que una corriente debida a P1.

Para hacer esto, se utiliza una bomba alternativa P1´ que se coloca en el circuito de P1.

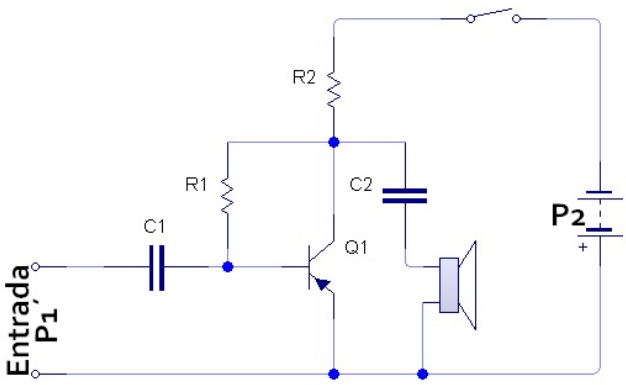
Este Montaje es llamado “AMPLIFICADOR”, y el trabajo que se efectúa de esta forma se llama “AMPLIFICACION”.

En la vida real, la bomba alterna de mando puede ser por ejemplo la cabeza de lectura de un toca-disco, o la antena de un aparato de radio.

Esta bomba, gracias al transistor, puede mandar corrientes relativamente importantes; Gracias a tan solo esto, se puede hacer vibrar por ejemplo la placa de un auricular o bien la membrana de un parlante.

Ahora, introduzcámonos un poco mas en la practica real, y observemos detenidamente un esquema de una etapa de amplificación.

Componentes:
- Un transistor (Q1)
- Dos resistencias (R1 y R2)
- Dos Capacitores (C1 y C2)
- Una Bomba alterna (P1´ - que puede ser la salida de nuestra radio o cualquier mp3 portátil)
- Un parlante
- Una fuente de 9Vcc



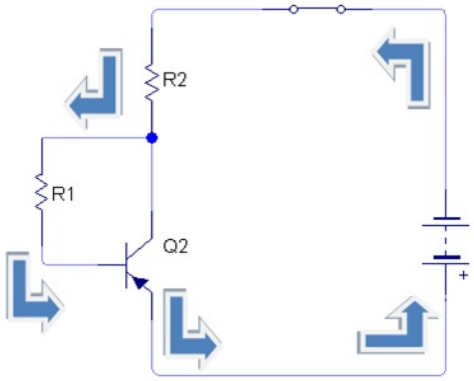
R1, R2, C1 y C2, son las novedades en este montaje… En el ejemplo de amplificación anterior, no teníamos un parlante, ya que era un sistema Hidráulico.

En el esquema que presentamos en modo novedoso, notamos que la Bomba P1´ a desaparecido… Estas novedades y supresiones, son toda la diferencia del esquema del principio, con el esquema del plano real… Pero sin dar mas vueltas, volvamos nuevamente al circuito del plano real…

Los circuitos:

Busquemos los caminos tratados por los electrones bajo la acción de las dos bombas del montaje, P2 y P1´…Comencemos en principio por P2.

A_ En el siguiente esquema, vemos el “**Circuito de Mando**"



Donde los electrones parten del polo Negativo de la fuente, atraviesan R2 y R1, después el espacio Base – Emisor del transistor y retornan al polo Positivo de nuestra fuente. Esta corriente que circula entre Base y Emisor, es pues, la corriente de mando del Transistor; Esto Prueba que P1 no es Indispensable, ya que como notamos en este caso, lo reemplazamos por P2.

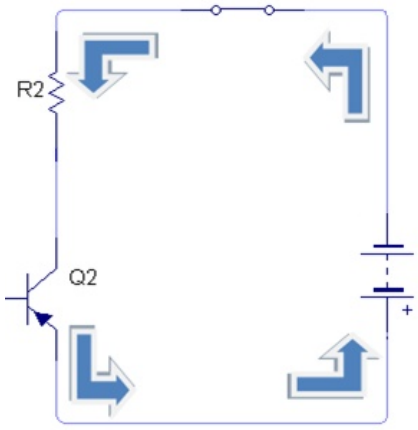
La resistencia R1, es la que le da a P2 esta posibilidad, ya que su posición permite a la corriente de mando circular; Su valor natural es muy grande, por tanto, esta resistencia moderadora permite la circulación de los electrones y obtiene por consecuencia la débil corriente de mando deseada (Recordemos que el transistor tiene como característica y consigna, mover grandes corrientes, con el empleo de otras pequeñas corrientes).

B_ **Circuito Gobernado**

Si circula una corriente entre base y emisor, el transistor conduce. Los electrones a la salida de R2 pueden, pues, no solamente pasar por R1, si no que también atravesar el colector, la base y el emisor del transistor, para volver al polo

positivo de P2. Este, pues, es el circuito gobernado.

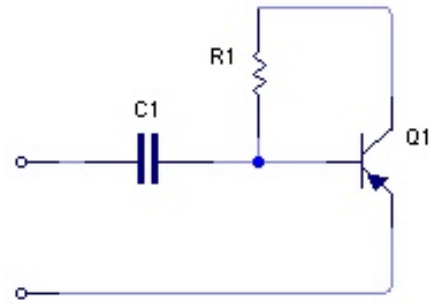
Observemos el siguiente esquema…



Pasemos ahora a P1´ dentro del circuito gobernado.

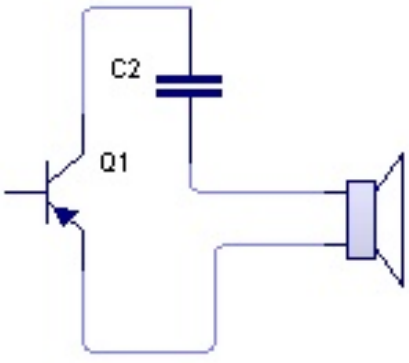
a) **Circuito Gobernado**:

P1´ una bomba alterna que sacude a los electrones en el circuito base-emisor del transistor a través del condensador C1



Ella, pues, unas veces refuerza y otras disminuye la corriente que circula entre la base y emisor del transistor.

b) **Circuito Gobernado**:

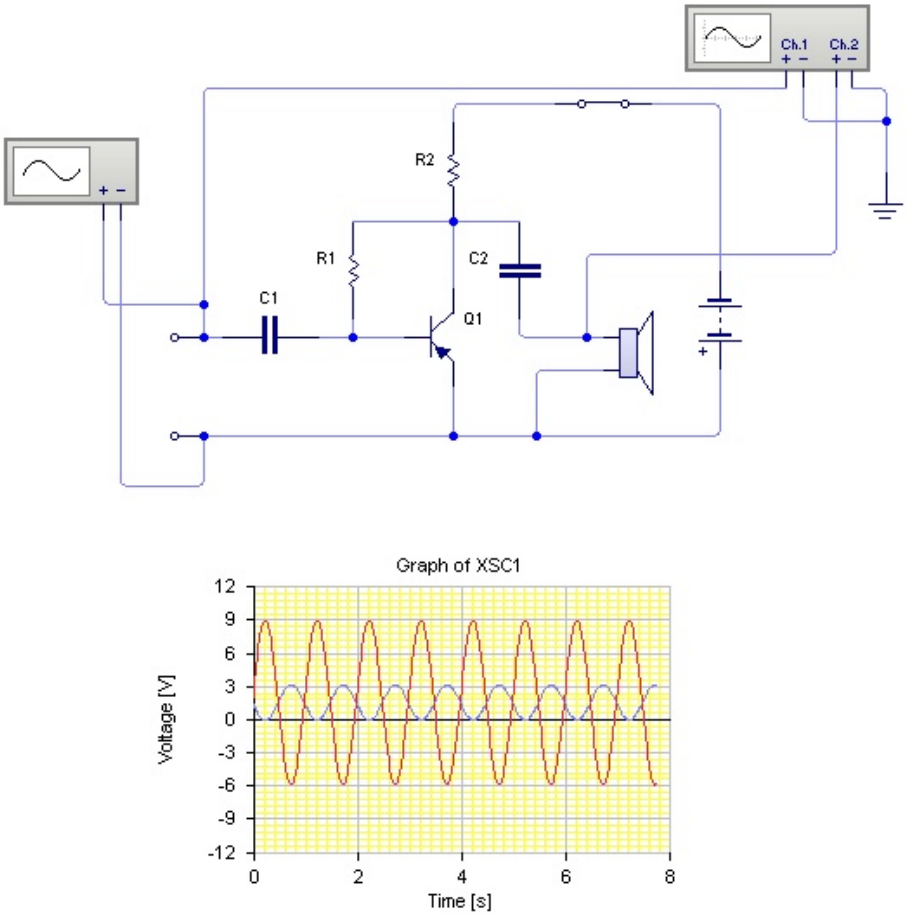


Gracias al mecanismo del transistor, la bomba P1´ hace aumentar o disminuir la corriente que atraviesa el Colector, la Base y Emisor del transistor.

El resultado es cargas y descargas sucesivas del condensador C2, que produce una corriente alterna en el circuito del parlante.

Por tanto, P1´ actúa sobre los electrones y el parlante, como lo haría una bomba P2´, mucho más importante, que se colocara directamente en su circuito.

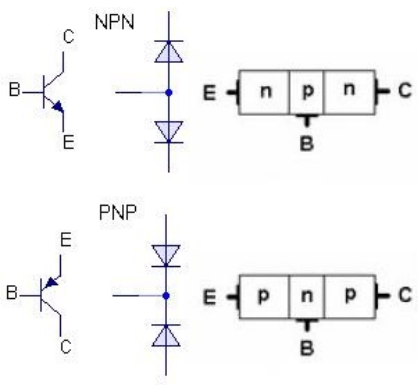
Cuando P1´ es demasiado débil, son necesarias varias etapas de amplificación. El transistor Q1 entonces no estaría al servicio directo del parlante, si no al servicio de un segundo transistor. En este caso, el parlante es reemplazado por el espacio Base-Emisor del que seria el segundo transistor (Q2).



Etapa de amplificación en pleno trabajo (Simulación en LiveWire)

Polarización del Transistor

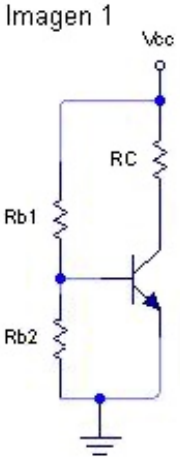
Internamente, podemos observar que el transistor está compuesto por dos diodos, según su juntura, podremos decir si es un transistor NPN, o un transistor PNP.



Para comprender el funcionamiento de estos semiconductores en modo de corte y saturación, recordemos que al principio de este capitulo, les mencione que transistor debe polarizarse para su funcionamiento; osea, la base se construye poco dopada y muy estrecha.

La imagen 1, muestra una típica polarización utilizando sólo una fuente de tensión Vcc., RB1 y RB2 son las llamadas resistencias divisoras de Base, y Rc la de Colector.

Imagen 1



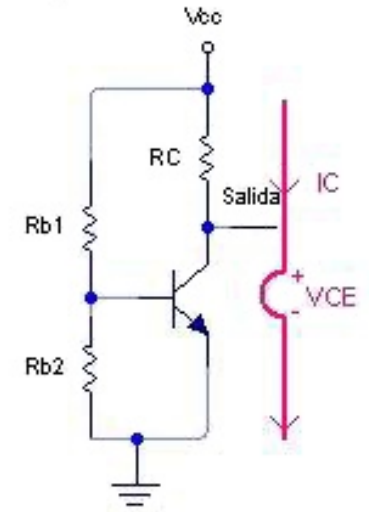
En este caso no se colocó ninguna resistencia en el emisor.

La salida se toma, en este caso, del Colector (cuando lleguemos al modo de funcionamiento MAD, veremos que la salida puede ser tomada de cualquiera de los terminales, y de esto depende el nombre de la configuración: Colector común, Emisor común o Base común).

Para saber en qué punto esta funcionando el transistor, es decir qué valor tiene IC y qué valor tiene VCE, se realiza un análisis llamado estático o análisis de continua, haciendo referencia a que sólo se tiene en cuenta las fuentes de alimentación de continua.

Para ello se recorre la malla de salida:

Imagen 2

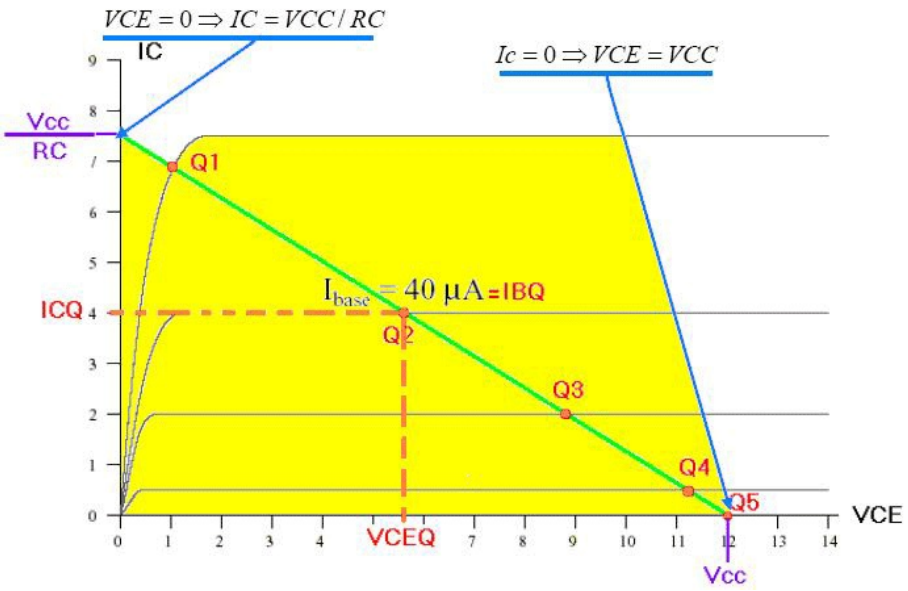


VCC= Ic . Rc + Vce

Ésta fórmula representa una recta, y es llamada Recta de carga estática.

Dibujamos ésta recta, en el gráfico ya visto de IC (corriente de Colector) en función de VCE (tensión Colector Emisor). De la intersección de dicha recta y las curvas, obtenemos el punto de funcionamiento del transistor.

Los puntos de la recta que cortan los ejes son:



Entonces, como dijimos, de la intersección de la curva y la recta, se halla el punto de funcionamiento del transistor. Este punto es llamado punto Q o punto de trabajo del transistor, y para dicho punto obtendremos un ICQ y un VCEQ, correspondiente a la corriente de colector y la tensión entre colector emisor a la cual se esta trabajando.

En el gráfico anterior se muestran diferentes puntos posibles de trabajo (Q1, Q2, etc.)

Recordando lo expuesto en la entrega anterior, vemos que, si el transistor se encuentra trabajando en el punto Q1, su funcionamiento es en modo saturación, de Q2 a Q4 en MAD y Q5 en corte.

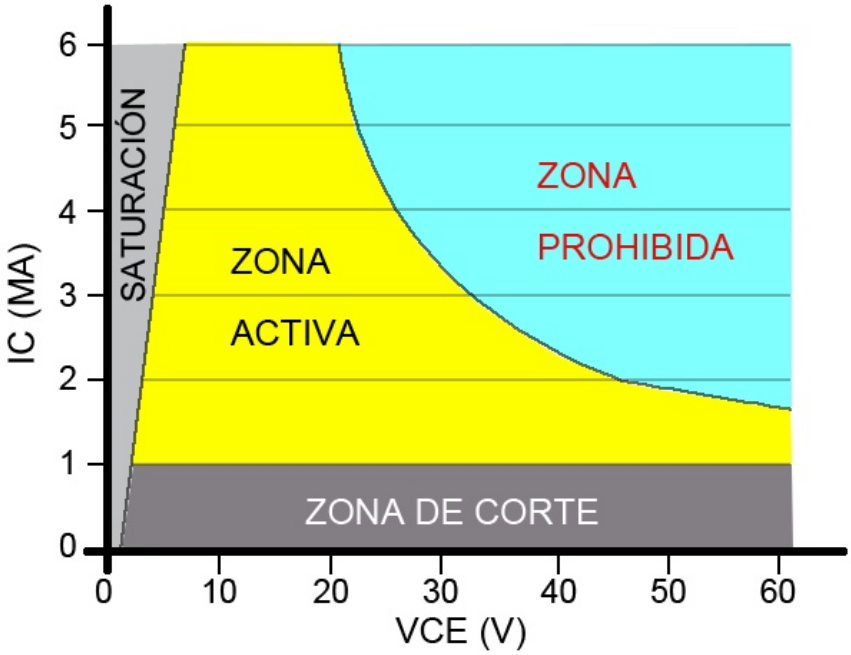
Parámetros “β”

Se define el parámetro “β” a una constante proporcional y como a la relación β= Ic/Ib, de la cual podemos derivar que **IC=β.Ib**

En la practica, “β” puede valer entre 50 y 300 y llegar a 1000 en algunos transistores, es decir, la intensidad del Colector (Ic), es unas veces mayor que la intensidad de Base (Ib) y proporcionalmente a la misma.

Las curvas mas interesantes del transistor, son las de salida.

En el siguiente grafico, está la representación de la intensidad de Colector en función de la tensión Colector-Emisor (VCE) para diferentes valores de intensidad de Base (Ib).



Ahora se preguntaran, ¿que significa y que son la “saturación”, “corte”, etc?

***Zona Activa**: En ella para cada intensidad de BASE, resulta una intensidad de Colector (Ic) “β” veces mayor, manteniéndose sensiblemente constante la tensión Colector-Emisor. La zona activa funciona como amplificador. Para que el transistor funcione correctamente, debe mantenerse su funcionamiento dentro de esta zona.

***Zona Saturación**: Al entrar en esta zona, la intensidad de Colector se dispara convirtiéndose el transistor en un circuito cerrado, osea que cuando por la Base circula una intensidad, se aprecia un incremento de la corriente de Colector considerable.

En este caso el transistor entre Colector y Emisor, se comporta como un interruptor cerrado; De esta forma se puede decir que la tensión de la batería se encuentra en la carga conectada en el Colector.

***Zona de Corte**: En esta zona la intensidad de Colector se anula y el transistor se comporta como un interruptor abierto.

Viéndolo de esta manera, el diodo formado por la corriente Base-Colector tiene una intensidad menor a la del Emisor; No circula intensidad por la Base, por lo que la intensidad de Colector y Emisor también es nula. La tensión entre Colector y Emisor es la de la batería o fuente.

***Zona Prohibida**: En ella, el producto de tensión por intensidad supera la potencia máxima tolerable por el transistor, osea que cuando se llega a esta zona, se puede llegar a destruir el transistor.

Cálculo para métodos de polarización

Hay diversas maneras de polarizar a un transistor, en este curso, veremos solo alguna de ellas y aprenderemos como realizar el calculo de polarización para un correcto funcionamiento.

Recordando que para que un transistor opere normalmente la unión Base-Emisor, debe tener polarización directa, en cambio, la unión Base-Colector debe de tener polarización inversa.

Nuevamente, recordemos la recta de carga citada anteriormente...

Como ya sabemos, la recta de carga es una línea diagonal, que se dibuja sobre la curva de salida del transistor, y que indica los extremos de operación del mismo.

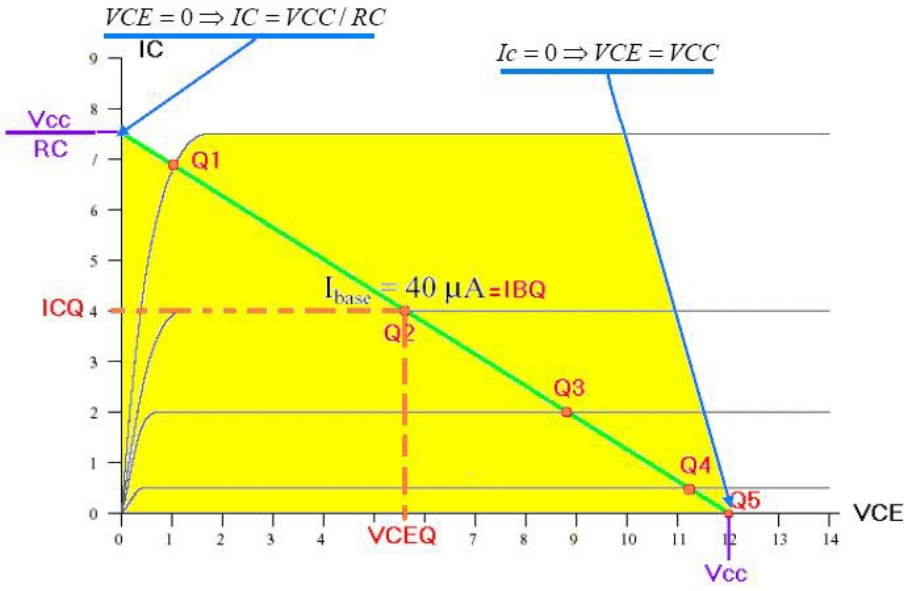
Cada uno de los puntos de esta recta, posee como coordenadas un valor de corriente de Colector y un voltaje Colector-Emisor, los cuales a su vez definen el valor de los componentes a ser utilizados en el circuito de polarización.

Las principales formulas a memorizar son:

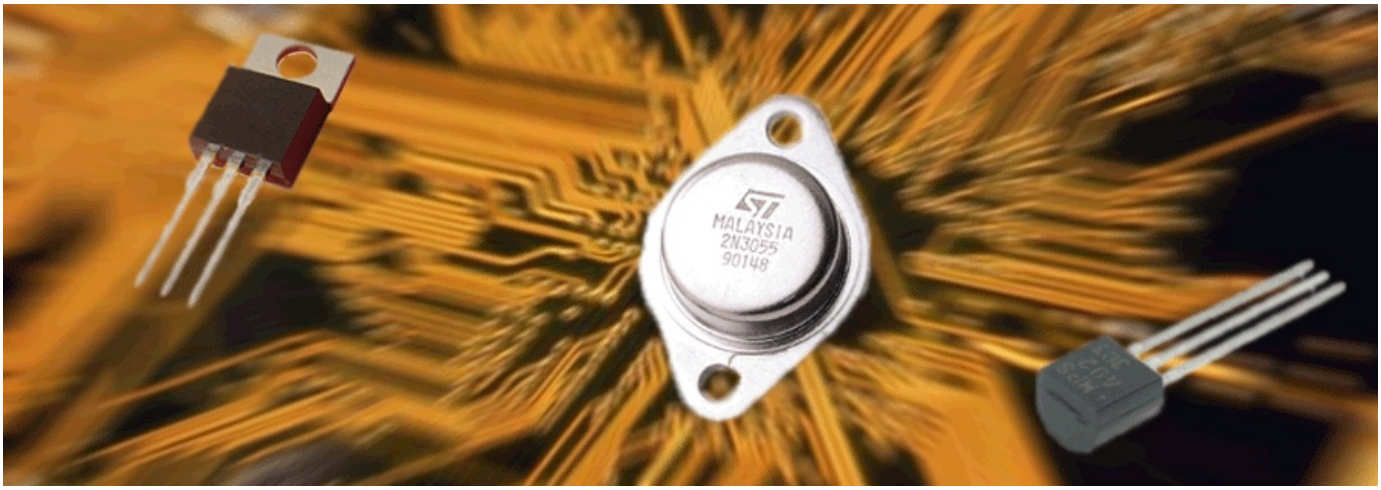
Vbe= 0,6v a 0,7v (dependiendo de si el transistor es de silicio o germanio. Esto se puede saber en la misma hoja de datos del semiconductor)

Ic= Ib . Hfe

Ie= Ib + Ic

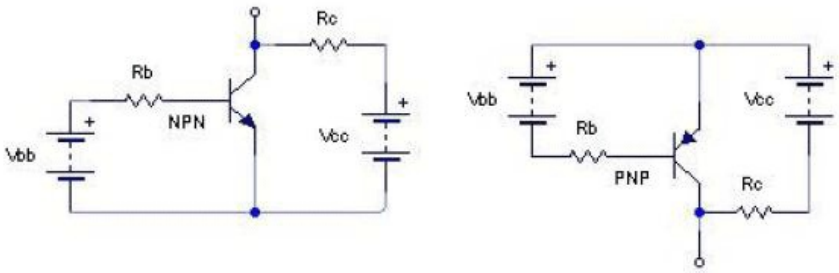


SABIENDO QUE...	
RC = Resistencia de Colector	Vcc= Fuente o Voltaje principal
Rb = Resistencia de Base	Vbb= Fuente o Voltaje secundario
Re = Resistencia de Emisor	Vbe= Voltaje Base-Emisor
RTH= Resistencia equivalente	Vbc= Voltaje Base-Colector
Ic= Intensidad de Colector	Vce= Voltaje Colector-Emisor
Ib= Intensidad de Base	Vc= Voltaje Colector
Ie= Intensidad de Emisor	Ve= Voltaje Emisor
HFE= Ganancia del transistor	Vb= Voltaje Base
PD= Potencia del Transistor	VTH= Voltaje Equivalente



Tenemos...

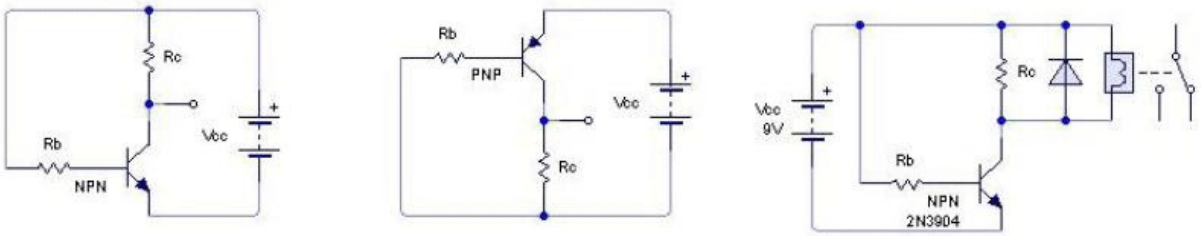
Calculo de Polarizacion Simple con dos Fuentes:



Calculo de circuito	Calculo de recta de carga
Ib=(Vbb - Vbe) / Rb **Ic=Ib . Hfe** **Vce=Vcc - (Ic . Rc)** **PT=Vce . Ic**	**Vce Max = Vcc** **Ic Max = Vcc / Rc**

Calculo de Polarizacion Simple con una Fuente:

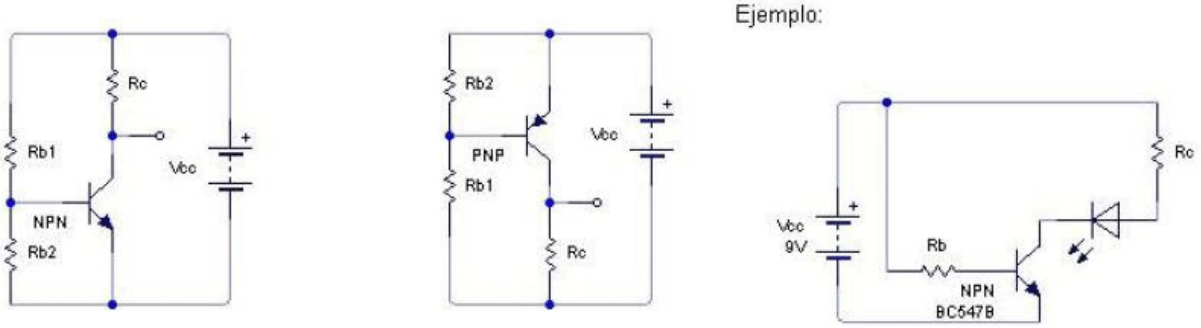
Ejemplo:



Calculo de circuito	Calculo de recta de carga
Ib=(Vbb - Vbe) / Rb **Ic=Ib . Hfe** **Vce=Vcc - (Ic . Rc)** **PT=Vce . Ic**	**Vce Max = Vcc** **Ic Max = Vcc / Rc**

Calculo de Polarizacion Simple con divisor de tension:

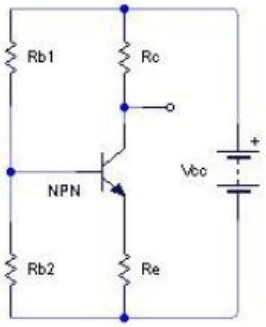
Ejemplo:

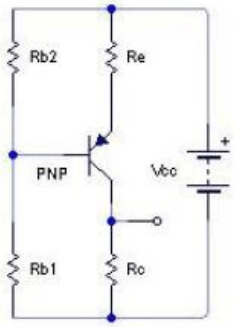


Calculo para polarización simple con divisor de tension:

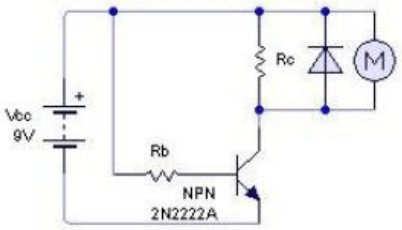
Calculo de circuito	Calculo de recta de carga
RTH= Rb1 . Rb2 / (Rb1 + Rb2) **VTH= Vcc . Rb2 / (Rb1 + Rb2)** **Ib= (VTH - Vbe) / RTH** **Ic= Ib . Hfe** **Vce= Vcc - (Ic . Rc)** **PT= Vce . Ic**	**Vce Max = Vcc** **Ic Max = Vcc / Rc**

Calculo de Polarizacion Simple con realimentacion de Emisor:





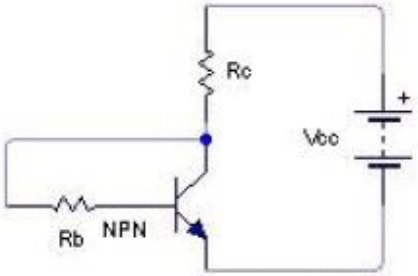
Ejemplo:

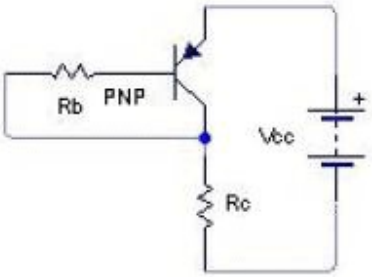


Calculo para polarización con realimentación en Emisor:

Calculo de circuito	Calculo de recta de carga
RTH= Rb1 . Rb2 / (Rb1 + Rb2) **VTH= Vcc . Rb2 / (Rb1 + Rb2)** **Ib= (VTH - Vbe) / (RTH + Re . (Hfe + 1))** **Ic= Ib . Hfe** **PT= Vce . Ic**	**Vce Max = Vcc** **Ic Max = Vcc / (Rc + Re)**

Calculo de Polarizacion Simple con realimentacion de Base:





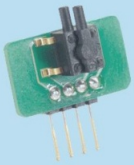
Calculo para polarización con realimentación en Base:

Calculo de circuito	Calculo de recta de carga
Ib= (Vcc - Vbe) / Rb + Rc . (Hfe + 1)) **Ic= Ib . Hfe** **Vce= Vcc - (Rc . Ic (1 + 1 / Hfe))** **PT= Vce . Ic**	**Vce Max = Vcc** **Ic Max = Vcc / Rc**

Antes de proseguir con lo que resta de la unidad, tendremos que ordenar e intentar tener en claro los conceptos básicos que vimos anteriormente. Mas que nada, para que al continuar con esta sección, se sigan sumando conceptos e ideas y no todo lo contrario…"Confusiones".





automatismos

MAR DEL PLATA

* noticias
* artículos
* proyectos
* ideas de diseño
* recursos didacticos
* y mucho, mucho más...

http://www.automatismos-mdq.com.ar/blog

Tutorial ASM
...desde 0

El lenguaje Assembly o Ensamblador (en español) es un lenguaje de bajo nivel que se utiliza para escribir programas o software. La misma es la representación directa al código de máquina y es la razón principal por su alto grado de complejidad a la hora de crear un programa. Básicamente se escribe en códigos de máquinas y esto conlleva a que el programador (el usuario) tenga que saber muy bien la arquitectura del microprocesador, microcontrolador o cualquier dispositivo a programar en este lenguaje.

// por: David (Leon Pic) //
david@meteorologiafacil.com.ar

Introducción

A diferencia de los lenguajes de alto nivel, el ensamblador permite un control total del Contador de Programa (CP a partir de ahora), pero esto también corresponde a una gran desventaja debido, a como se mencionó antes, el programador necesita saber como funciona el microcontrolador, o el procesador.

Cada instrucción en ensamblador corresponde a una instrucción del dispositivo pero en binario o lenguaje de máquina.

Por ejemplo la instrucción:
MOVF 0X20,F

Corresponde al código de máquina:
00100001000000

Queda de manifiesto que es más fácil acordarse el primer ejemplo que el segundo. Es por ello que se crea el ensamblador, propiamente dicho, que interpreta cada instrucción escrita en modo texto para pasarlo en binario, el cual, es el único lenguaje que entiende el microprocesador.

Este lenguaje posee varias ventajas y desventajas a saber. A la hora de migrar un programa en ensamblador de un microprocesador a otro, el programa debe ser modificado casi en su totalidad debido a la incompatibilidad directa de un microprocesador a otro. Posee un alto grado de complejidad a la hora de crear un programa y es complicado interpretar o leer un programa en este lenguaje. Lleva mucho tiempo escribir un programa entero y es el usuario que se encarga de hacer todo el esfuerzo para controlar distintos dispositivos y hardware. Esto conlleva a que el usuario debe saber manejarlos.

Pero no todo es desventaja en este lenguaje. Posee grandes ventajas que, hasta ahora, ningún otro lenguaje ha podido igualar. Los programas creados con estos lenguajes se ejecutan más rápidos, son óptimos, más pequeños y se logran mejores tiempos de demoras o de temporizadores con este lenguaje. Consumen menos recursos y se puede evitar o activar las interrupciones de uno o más eventos en lugares exactos.

El lenguaje ensamblador que veremos a continuación, será dedicado al PIC de 8 bits.

Lo primero que hay que saber para este lenguaje, es que cada línea de código, es una instrucción que realiza el CP o un paso que realiza el CP. A diferencia de otros lenguajes, como por ejemplo el C, una línea de

instrucción, puede llevar uno o más pasos que el CP debe realizar. Más adelante entenderán el porqué.

A esto último dicho, nos explica el porqué de un programa escrito en ASM (assembler) lleva menos línea de instrucción que el mismo programa realizado en otro lenguaje. Esto se traduce que, cuanto menos líneas de código, menor consumo de la memoria de programa.

Nota: Entendemos por el mismo código, al realizar un software para realizar una dicha tarea. Una tarea que debe realizar nuestro microcontrolador, puede ser escrita en diferentes lenguajes, y es el programa que se utilizó quien lo traduce a formato hex (1 y 0).

¿Qué es el CP?

El CP o PC es el Contador de Programa. Es el encargado de leer cada instrucción y realizar la acción solicitada por el software.

Para dar un ejemplo un poco más claro, digamos que el CP, es una persona. Esta persona se le da un papel que tiene diferentes tareas a realizar. Estas tareas, están una debajo de la otra y lee línea por línea y hace lo que le dice esa línea. Cada línea, posee una instrucción que se debe hacer. Cuando terminó de hacer esa instrucción, continúa con la línea de abajo.

Supongamos que tenemos un papel que dice:

Levantar la mano izquierda.
Bajar la mano izquierda.
Saltar tres veces en el mismo lugar.
Levantar la mano derecha.
Saltar una vez en el mismo lugar.
Bajar la mano derecha.

El CP, hará esas tareas sin negarse y lo hará fielmente a lo que está escrito. Por lo que, si hace una tarea mal, es porque le pusimos una o más instrucciones mal.

Si bien, el CP hará lo que nosotros le pidamos, nos pide que respetemos algunas cosas. Estas cosas, dependerán de cada microcontrolador y que debemos saber para poder programar en forma correcta al PIC.

¿Cómo es la estructura del lenguaje ASM?

Es muy fácil, posee 4 columnas bien diferenciadas y que no es problema acordarse.

* La primera columna se llama ETIQUETA y sirve para darle el nombre a una posición de la memoria del programa al que se necesita apuntar. Los que no se den cuenta, ya lo harán.

* La segunda columna se llama INSTRUCCIÓN y lo que justamente hace, es una instrucción a realizar por el CP.

* La tercera columna se llama OPERANDO y es el operando de una instrucción, o sea, de la segunda columna. Hay instrucciones que no tienen operando y las veremos más adelante.

* Y la cuarta columna se llama OBSERVACIONES y sirve solo para el programador, en otras palabras, sirve para el usuario que está programando el PIC. Siempre comienza con ; (punto y coma)

Si no sirve para el PIC, ¿Para qué complicarla más? Todo lo contrario, es para ayudar al programador de que no se olvide que intentó hacer. Todos los lenguajes, desde los más básicos hasta los más avanzados, tienen esta características ya que en el, pondremos que es lo que queremos hacer o explicar el programa.

Si bien estas observaciones se utilizan en la cuarta columna, no es obligación colocarla ahí, si no que puede ir al margen de la planilla en dónde estamos programando. Y es aquí dónde explicaremos la porción de una rutina para entenderlo más adelante.

Bien, veremos a continuación como se escriben las columnas:

```
ETIQUETAS    INSTRUCCIÓN    OPERANDO    ;OBSERVACIONES
```

O podemos verlo así:

```
;OBSERVACIONES
;OBSERVACIONES
;OBSERVACIONES (y la cantidad que necesitemos)

ETIQUETAS    INSTRUCCIÓN    OPERANDO    ;OBSERVACIONES
```

Antes de empezar a ver las instrucciones, debemos concentrarnos en los registros.

¿Qué son los registros?

Los registros, son posiciones de memoria que se utilizan para ir configurando el PIC mientras se corre el programa, cambiar de bancos para acceder a otras partes de memorias, hay banderas que nos van diciendo que está pasando con distintas operaciones, hay bits para habilitar o deshabilitar módulos que trae el PIC, por ejemplo el conversor Analógico/Digital, PWM, etcétera.

Estos registros, tienen una ancho de bit de acuerdo al PIC, que hay de 8 bit, 16 bit y 32 bit. Excepto la palabra configuración y que la veremos mas adelante. El tamaño de los registro depende del tipo de microcontrolador. Cada registro, posee un nombre, y cada bit o un grupo de bit, se puede utilizar para lo descripto anteriormente (habilitar/desabilitar, etc).

En la Figura_1 de la página siguiente podemos ver una posición de la RAM y sus 4 bancos. Este, es del PIC16F877

Como notarán, en las cuatro columnas, hay nombres. Estos nombres son todos los registros que posee este PIC. Verán que hay registros que se repiten. No quiere decir que están duplicados, triplicados o cuadriplicados, si no que se puede acceder a ellos no importa en que banco estemos trabajando. (Ya lo entenderán, no se preocupen).

Cada vez que se programe el PIC y se necesite trabajar con uno de estos registros, se debe acceder al banco en que esté dicho nombre. Es por eso, que esta tabla es muy importante a la hora de trabajar.

Cuando dé ejemplos de programación, será sobre este PIC. Microchip provee en forma gratuita estos datasheet y hay uno por cada PIC, por lo que si no lo tienen, descárguenlo de la web de Microchip. Cuando empecemos a programar, lo necesitarán.

Uno de los registros muy utilizado, es el registro de trabajo W, el cual, se utiliza para mover un dato/valor de un registro a otro, cargar valores en un registro, y con la ayuda de la ALU puede hacer operaciones matemáticas.

Observen con atención el Diagrama de Bloques dónde se encuentra W (Figura_2, página 0x3B)

Ahora bien, si W es un registro, ¿dónde está ubicado, ya que en el mapa de memoria no lo encuentro?

El registro W no está implementado en la memoria RAM, ni en la posición 0x00 ni en otra posición. W es un registro independiente que tiene un bus directo con la ALU (un camino privado). La ALU es la única entidad que puede leer o escribir este registro de trabajo.

El direccionamiento indirecto hace uso de los registros FSR e INDF. INDF es completamente ajeno a W. Un direccionamiento indirecto usa el registro FSR como apuntador al contenido de otros registros. Cualquier instrucción que hace uso de INDF (0x00) como dirección invoca un direccionamiento indirecto.

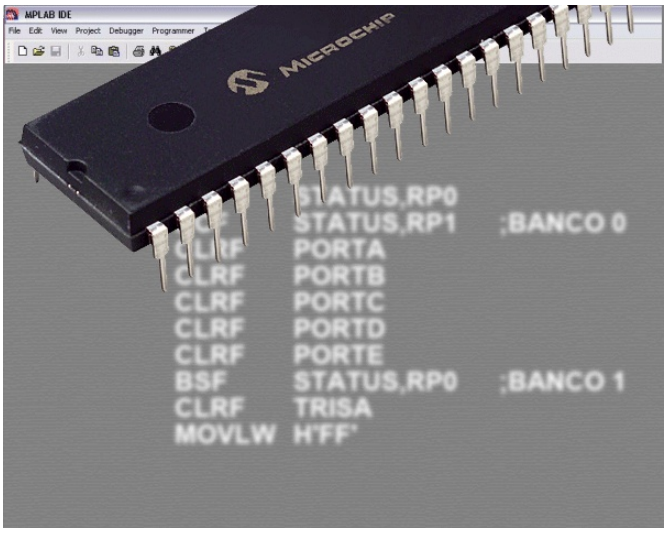
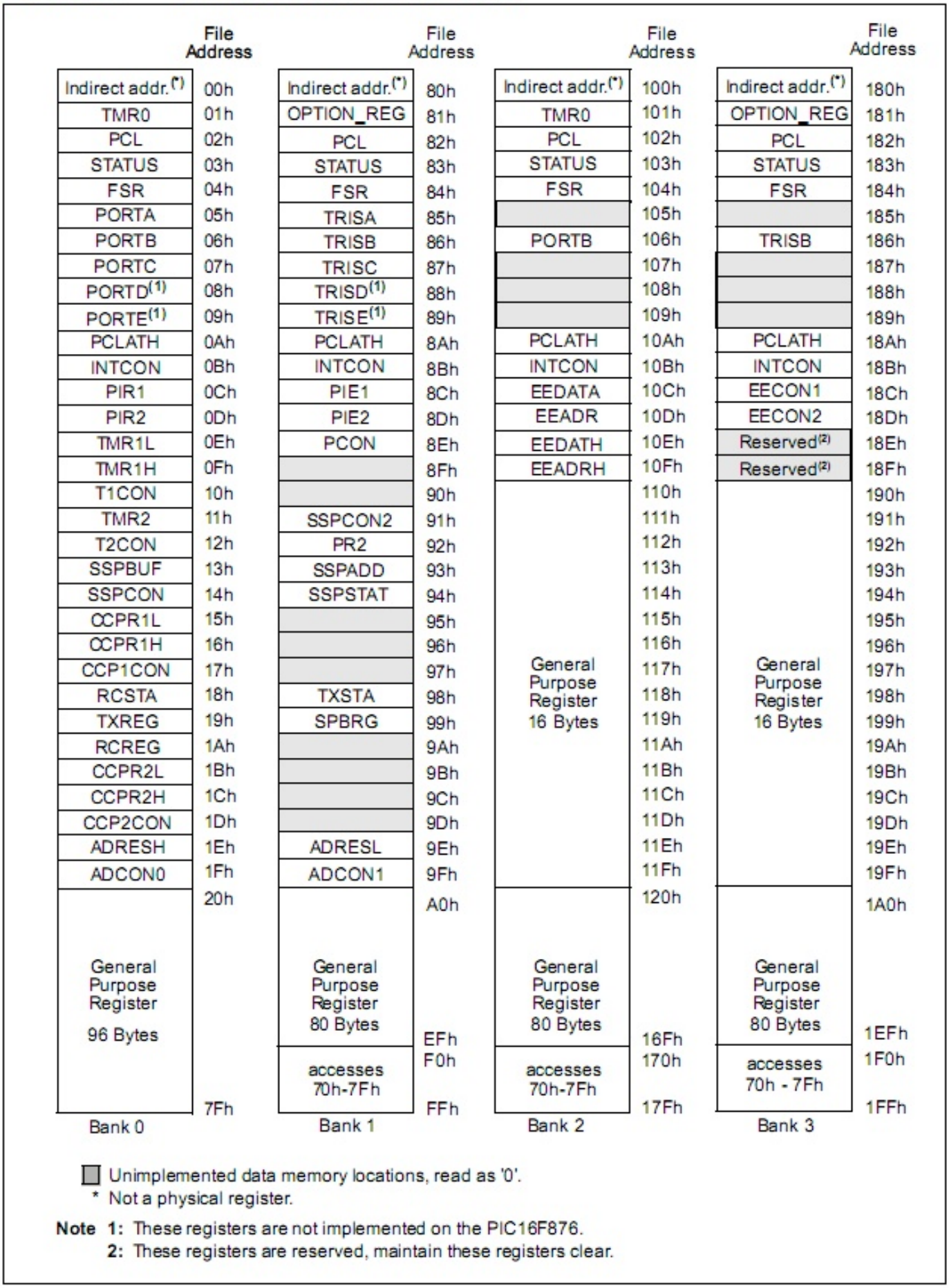


FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP



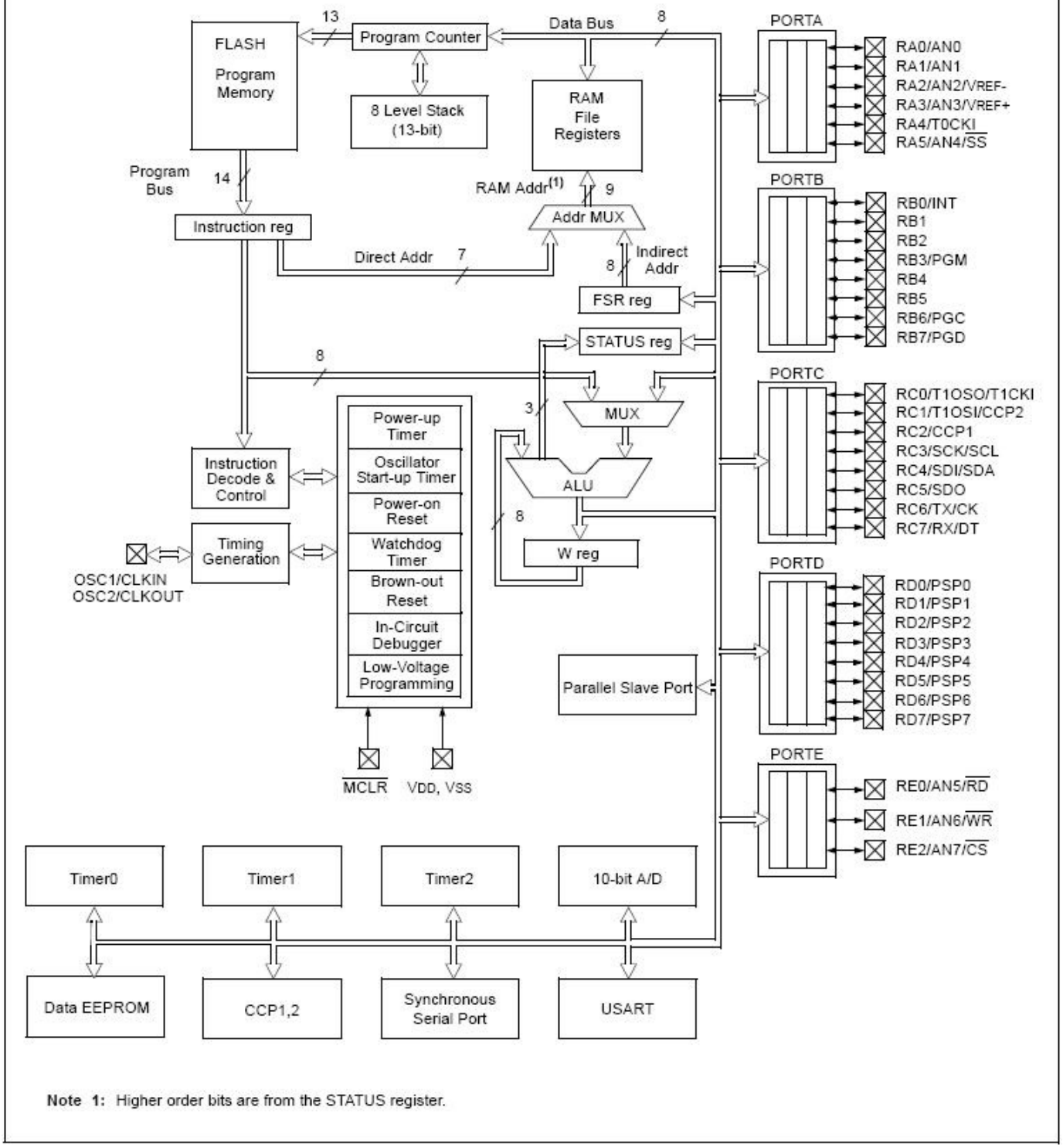
Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

Figura_1: Registros PIC

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

Figura_2: Diagrama de Bloques

Los microcontroladores PIC, poseen tres tipos de memorias.

* **Memoria de Programa**
Es la ubicación física dónde se guarda el firmware que hemos creado, o sea, nuestro programa. Y tiene un ciclo de 100.000 de lectura y/o escrituras antes de estropearse y es del tipo FLASH.

* **Memoria de datos de uso general**
Es la memoria RAM del PIC. Recordemos, que los registros están sobre la memoria RAM y la memoria de uso general, comienza después de los registros.

* **Memoria de datos EEPROM**
Es la memoria EEPROM que puede almacenar datos mas de 40 años sin energía y 1.000.000 de ciclos de escritura y lectura

El PIC 16F877, tiene una memoria de programa de 8Kb por un ancho de 14 bits, o sea que cada posición de memoria tiene 14 bits (B'11111111111111', o H'3FFF') Cada línea de instrucción ocupa una posición en la memoria de programa, asi que, podemos poner hasta 8.192 instrucciones.

La RAM de uso general (más los registros) trae 368 bytes con un ancho de 8 bits (B'11111111', o H'FF'). Esto quiere decir, que tenemos 368 posiciones para nuestro uso.

La EEPROM trae 256 bytes con un ancho de 8 bits. Tenemos 256 posiciones para nuestro uso.

Ahora, vamos a estudiar el registro STATUS y luego continuaremos con las instrucciones.

Registro STATUS

Si entendieron hasta acá, se acordarán que los registros poseen un ancho de 1 Byte u 8 bits. Cada bit puede ser un 1 o 0.

Vemos ahora en detalle el registro STATUS:
Este registro, tiene 3 bits dedicados a las operaciones matemáticas, 3 bits dedicados al cambio de banco de memoria y 2 bits dedicados a saber que o quién produjo un Power Up (despertar del micro). Y se puede leer y escribrir en él (cambiar datos).

Los analizamos desde el más significativo (MSB) hasta el menos significativo (LSB).

BIT 7:
Se llama IRP y sirve para el direccionamiento indirecto para cambiar el banco de memoria.
1 = Banco 2 y 3
0 = Banco 0 y 1

BIT 6 y **BIT 5**
Se llaman RP1 y RP0 respectivamente. Sirve para el direccionamiento directo para cambiar de banco de memoria.
00 = Banco 0
01 = Banco 1
10 = Banco 2
11 = banco 3

BIT 4
Se llama TO (neg). Este bit se utiliza para saber quién despertó al PIC.
1 = Después que despierta (power up) o por las instrucciones CLRWDT o SLEEP, se pone a 1 este bit.
0 = Se pone a 0 cuando el wachtdog o en castellano perro guardían (WDT) despierta al PIC.

BIT 3
Se llama PD (neg). Este bit se utiliza para saber si el PIC estaba durmiendo.
1 = Después de que despierta (power up) o por la instrucción CLRWDT, se pone a 1
0 = Se pone a 0 cuando se ejecuta la instrucción SLEEP

BIT 2
Se llama Z y al igual que los dos bits anteriores, es una bandera. Nos indica el resultado de una operación aritmética y lógica.
1 = La operación aritmética o lógica dio como resultado 0
0 = La operación aritmética o lógica no dio como resultado 0

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 **$\overline{TO}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$**: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

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

Figura_3: Registro STATUS

BIT 1

Se llama DC. Digit carry/borrow (dígito llevar/prestar). Es afectado por las instrucciones ADDWF; ADDLW; SUBLW; SUBWF (Para la resta, la polaridad es inversa).

1 = Hubo un acarreo del 4to bit menos significativo al 5to bit.

0 = No hubo un acarreo del 4to bit menos significativo al 5to bit.

BIT 0

Se llama C carry/borrow. Es afectado por las mismas instrucciones que afectan al bit DC.

1 = Hubo un acarreo del bit más significativo (Bit 7) o sea cuando se excede de H'FF'

0 = No hubo acarreo del bit más significativo


µCONTROL
Electrónica en General Pics en Particular